Reproductions

Published actuarial and numerical results, rebuilt with aggregate

Stephen J. Mildenhall

2026-10-22 19:31 commit 8012e55+

Table of contents

Preface 8
The structure of each chapter . . . 8
Three categories of paper . . . 9
Conclusions: correctness, flexibility, errors, and exactness . . . 9
The chapters . . . 10
Reading and rebuilding . . . 13

I. The short version 14

1. Five papers, five programs 15
1.1. How to read this chapter . . . 15
1.2. Venter (PCAS, 1983): Transformed Beta And Gamma Distributions And Aggregate Losses . . . 15
1.3. Bear and Nemlick (1990): six treaty features, six clauses . . . 18
1.4. Homer and Clark (2003): the joint distribution is a view, not a second model . . . 21
1.5. Bruno, Camerini, Manna and Tomassetti (2006): 92,832 convolutions . . . 22
1.6. Denuit, Dhaene, Ghossoub and Robert (2025): the check runs both ways . . . 24
1.7. Where the detail is . . . 28

II. Actuarial models 29

2. Venter and the transformed gamma distribution 30
2.1. Context and problem . . . 30
2.2. Paper solution and gold standard . . . 30
2.3. The `aggregate` reproduction . . . 32
2.4. Comments . . . 35

3. Mack and Fackler's test of Riebesell's rule 38
3.1. Context and problem . . . 38
3.2. Paper solution and gold standard . . . 38
3.3. The `aggregate` reproduction . . . 39
3.4. Theorem 2, end to end . . . 41
3.5. Comments . . . 44

4. Mata and Verheyen on trend by layer 45
4.1. Context and problem . . . 45
4.2. Paper solution and gold standard . . . 45
4.3. The `aggregate` reproduction . . . 46
4.4. The decisive test . . . 48
4.5. Comments . . . 53

5. Bodoff and excess of policy limits losses 55
5.1. Context and problem . . . 55

5.2. Paper solution and gold standard . . . 55
5.3. The aggregate reproduction . . . 56
5.4. Comments . . . 58

6. Homer and Clark's bivariate FFT 60
6.1. Context and problem . . . 60
6.2. Paper solution and gold standard . . . 60
6.3. Section 2: Per-Occurrence And Stop Loss . . . 62
6.4. Section 3: loss subject to an aggregate, plus unlimited ALAE . . . 69
6.5. Section 4: dollars against counts . . . 74
6.6. Comments . . . 79

7. Bear and Nemlick on adjustable treaty features 83
7.1. Context and problem . . . 83
7.2. Paper solution and gold standard . . . 84
7.3. The aggregate reproduction . . . 91
7.4. Comments . . . 102

III. Distortion pricing 105

8. Wang and risk loads by layer 106
8.1. Context and problem . . . 106
8.2. Paper solution and gold standard . . . 106
8.3. The aggregate reproduction . . . 108
8.4. Comments . . . 110

9. Wang and the pricing of weather options 113
9.1. Context and problem . . . 113
9.2. Paper solution and gold standard . . . 113
9.3. The aggregate reproduction . . . 115
9.4. Comments . . . 117

IV. Risk sharing and conditional expectations given the sum 119

10. Denuit and conditional mean risk sharing 120
10.1. Context and problem . . . 120
10.2. Paper solution and gold standard . . . 120
10.3. The aggregate reproduction . . . 122
10.4. Comments . . . 125

11. Denuit, Hieber and Robert on mortality credits 126
11.1. Context and problem . . . 126
11.2. Paper solution and gold standard . . . 126
11.3. The aggregate reproduction . . . 127
11.4. Comments . . . 131

12. Denuit et al. on comonotonic allocations 133
12.1. Context and problem . . . 133
12.2. Paper solution and gold standard . . . 133
12.3. The aggregate reproduction . . . 134
12.4. Comments . . . 141

13. Denuit, Ortega-Jiménez and Robert on conditional expectations given the sum 144
13.1. Context and problem . . . 144
13.2. Paper solution and gold standard . . . 144
13.3. The `aggregate` reproduction . . . 146
13.4. Comments . . . 150

V. Exact computation and convolution accuracy 152

14. Gerber and compound Poisson stop loss premiums 153
14.1. Context and problem . . . 153
14.2. Paper solution and gold standard . . . 153
14.3. The `aggregate` reproduction . . . 156
14.4. Comments . . . 159

15. Robertson and the sum of uniform claims 161
15.1. Context and problem . . . 161
15.2. Paper solution and gold standard . . . 161
15.3. The `aggregate` reproduction . . . 163
15.4. Comments . . . 165

16. Embrechts, Grübel and Pitts on discretization error 168
16.1. Context and problem . . . 168
16.2. Paper solution and gold standard . . . 169
16.3. The `aggregate` reproduction . . . 170
16.4. Comments . . . 173

17. Grübel and Hermesmeier on aliasing and exponential tilting 175
17.1. Context and problem . . . 175
17.2. Paper solution and gold standard . . . 175
17.3. The `aggregate` reproduction . . . 177
17.4. Comments . . . 180

18. Pittarello, Luini and Marchione's GEMAct package 182
18.1. Context and problem . . . 182
18.2. Paper solution and gold standard . . . 182
18.3. The `aggregate` reproduction . . . 186
18.4. Comments . . . 196

19. Embrechts and Frei's three discretization designs 197
19.1. Context and problem . . . 197
19.2. Paper solution and gold standard . . . 197
19.3. The `aggregate` reproduction . . . 199
19.4. Figure 1: the Poisson/Pareto aliasing picture . . . 200
19.5. Comments . . . 203

20. Schaller and Temnov on a severity with no mean 204
20.1. Context and problem . . . 204
20.2. Paper solution and gold standard . . . 204
20.3. The `aggregate` reproduction . . . 205
20.4. Comments . . . 208

21. Bruno et al. on thresholded direct convolution 210
21.1. Context and problem . . . 210
21.2. Paper solution and gold standard . . . 210
21.3. Table 6: Poisson, $\lambda = 504.814259$ . . . 211
21.4. Table 8: Poisson mixed by a generalized Pareto . . . 214
21.5. Table 7: $\lambda = 91{,}000$, the case the competitors could not do . . . 216
21.6. Comments . . . 219

22. Gollini and Rougier on a catastrophe model event loss table 221
22.1. Context and problem . . . 221
22.2. Paper solution and gold standard . . . 221
22.3. The `aggregate` reproduction . . . 227
22.4. Comments . . . 247

23. Wilson and Keich on accurate FFT convolution 252
23.1. Introduction . . . 252
23.2. What is reproducible . . . 252
23.3. The Section 1 example . . . 253
23.4. The Section 1 example, exactly . . . 255
23.5. Table 2: the FFT-C columns . . . 257
23.6. Discussion . . . 259

24. Wilson and Keich's sisFFT for small tail probabilities 260
24.1. Introduction . . . 260
24.2. What is reproducible . . . 261
24.3. Setup . . . 261
24.4. The true value, and what `aggregate` returns . . . 262
24.5. Where the pmf goes wrong . . . 263
24.6. The fix, applied . . . 263
24.7. An exact reference . . . 265
24.8. Discussion . . . 269

VI. Rare events and the far tail 270

25. Asmussen, Jensen and Rojas-Nandayapa on the lognormal left tail 271
25.1. The paper . . . 271
25.2. What is reproducible . . . 271
25.3. Setup . . . 272
25.4. Tables 5 and 10: $n = 4$, $\sigma = 0.25$ . . . 273
25.5. Table 3: `aggregate` as referee between first and second order . . . 274
25.6. Tables 6 and 11: $n = 64$, $\sigma = 0.25$ . . . 276
25.7. Tables 7 and 12: $n = 256$, $\sigma = 0.25$ . . . 277
25.8. Tables 8, 9, 13 and 14: shorter horizons . . . 279
25.9. Discussion . . . 281

26. Ben Amar et al. on state-dependent importance sampling 282
26.1. The paper . . . 282
26.2. What is reproducible, and what is not . . . 282
26.3. Setup . . . 283
26.4. Application 1: left tail of a log-normal sum . . . 284
26.5. Application 2: CDF of a ratio . . . 285

26.6. Figure 6 . . . 286
26.7. Where the FFT method breaks down . . . 288
26.8. Discussion . . . 289

27. Ben Rached, Hoel and Meo on the FFT accuracy floor 290
27.1. Introduction . . . 290
27.2. What is reproducible . . . 290
27.3. Setup . . . 291
27.4. Table 6: CDF and PDF . . . 291
27.5. Table 3: locating the floor . . . 292
27.6. Table 4: Lévy summands, and a harder limit . . . 294
27.7. Discussion . . . 296

28. Liu, Wu and Yao on k-moment tilting 297
28.1. Introduction . . . 297
28.2. What is reproducible . . . 298
28.3. Setup, and one parameterization trap . . . 300
28.4. Table 1: i.i.d. Pareto . . . 301
28.5. Table 2: i.n.d. Pareto, and where this stops working . . . 303
28.6. Table C.1: light-tailed, against an exact reference . . . 305
28.7. Discussion . . . 306

29. The left tail methods of Asmussen, Ben Amar and Ben Rached 307
29.1. Summary of Asmussen et al. (2016), Ben Amar et al. (2023), Ben Rached et al. (2024) 307

VII. A paper that did not reproduce 308

30. Jin, Provost and Ren's moment-based density approximations 309
30.1. Verdict . . . 309
30.2. What the paper does . . . 309
30.3. The one checkable univariate number . . . 310
30.4. The other five univariate examples publish nothing . . . 310
30.5. The bivariate tables are real, and are not `aggregate`'s dependence . . . 312
30.6. The joint law does reproduce, with a direct transform . . . 313
30.7. Why the marginals are not enough . . . 316
30.8. What would make this a reproduction . . . 316

VIII. Severity curves 318

31. Provenance of the named severity curves 319
31.1. Introduction . . . 319
31.2. Summary: paper to curve . . . 320
31.3. The curves . . . 321
31.4. Reference curves with no paper attached . . . 325
31.5. Using the curves . . . 326

IX. Appendix 327

32. Every program in the book 328
Five papers, five programs . . . 328

Mack and Fackler's test of Riebesell's rule . . . 330
Mata and Verheyen on trend by layer . . . 332
Bodoff and excess of policy limits losses . . . 332
Homer and Clark's bivariate FFT . . . 333
Bear and Nemlick on adjustable treaty features . . . 333
Wang and risk loads by layer . . . 339
Wang and the pricing of weather options . . . 339
Denuit and conditional mean risk sharing . . . 340
Denuit, Hieber and Robert on mortality credits . . . 341
Denuit et al. on comonotonic allocations . . . 341
Denuit, Ortega-Jiménez and Robert on conditional expectations given the sum . . . 342
Gerber and compound Poisson stop loss premiums . . . 343
Robertson and the sum of uniform claims . . . 343
Embrechts, Grübel and Pitts on discretization error . . . 344
Grübel and Hermesmeier on aliasing and exponential tilting . . . 344
Pittarello, Luini and Marchione's GEMAct package . . . 344
Embrechts and Frei's three discretization designs . . . 346
Schaller and Temnov on a severity with no mean . . . 346
Bruno et al. on thresholded direct convolution . . . 347
Gollini and Rougier on a catastrophe model event loss table . . . 347
Wilson and Keich on accurate FFT convolution . . . 348
Wilson and Keich's sisFFT for small tail probabilities . . . 348
Asmussen, Jensen and Rojas-Nandayapa on the lognormal left tail . . . 350
Ben Amar et al. on state-dependent importance sampling . . . 350
Ben Rached, Hoel and Meo on the FFT accuracy floor . . . 350
Liu, Wu and Yao on k-moment tilting . . . 351
Jin, Provost and Ren's moment-based density approximations . . . 352
Provenance of the named severity curves . . . 353

References 356

Preface

This book reproduces published figures, tables and results using `aggregate`. Every number in it is computed by the library, never transcribed. A chapter is recomputed when its text changes, and 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. A match therefore demonstrates the correctness of the library version named here, down to the git commit of its source.

```
Exception reporting mode: Minimal
Rendered 2026-10-22 19:31:20 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
```

A computational library must be correct. `aggregate` has an extensive set of *internal* correctness checks embodied in its validation system: these check for a match between the first three theoretical and empirical moments. However, a published number computed by somebody independent using a different method is a good *external* check. We work through results from twenty eight papers. Most of them confirm the library. Several of them corrected it, or located a limit in it that was not previously known. Nine of them turn out to contain an error, an internal inconsistency or a misprint, which is not a criticism of the authors so much as evidence that the check runs in both directions.

Solving external problems tests `aggregate` in a second way: it shows the library is flexible enough to solve problems posed *by other people*! Obviously, it can solve problems I think are important. The library's success on external problems bodes well when *you* come to use it.

The first chapter is the pith: five papers, each with the DecL program that reproduces the published exhibit and the answer set beside the paper's own numbers. It highlights how a simple DecL statement can answer an externally set question.

> ⚠ AI Generated Work!
>
> The mechanical work in this book was conducted by Claude Code Opus 5 model with high-level human review. In a sense, the work checks itself, but the reader should be aware it was essentially auto generated by AI.

The structure of each chapter

Each chapter follows the same four-part structure: the context and the problem the paper is solving, the paper's own solution and the gold standard it publishes, the `aggregate` reproduction, and comments. A simple reproduction can be very short. Something unusual, or unfamiliar to actuaries, needs more context.

When reviewing the code mentally discount the Python scaffolding and focus on the DecL nugget that does the work. Since the reproductions have to recreate the paper's work, they include a lot of code that is conceptually irrelevant other than helping to display correctness.

Jin et al. (2016) turned out not to be reproducible because of `aggregate` limitations, as the chapter on the paper records. It models a common shock count mode for bivariate frequencies.

Three categories of paper

The papers fall into three categories.

A **model paper** poses an insurance structure with explicit parameters. Reproduction means building it in DecL and matching the published numbers, and the paper is the authority. Venter (1983), Mack and Fackler (2003), Mata and Verheyen (2005), Bodoff (2017), Homer and Clark (2003) and Bear and Nemlick (1990) are of this kind, and they make up the first part.

A **method paper** proposes a competing numerical technique for the aggregate distribution. Here `aggregate` is the exact ground truth against which the paper's own approximation is measured, so the roles reverse: the paper is the thing being checked. Whether reproduction is possible at all depends on whether the paper published concrete input parameters and tabulated output. Several rare-event papers publish neither, reporting only sample counts and CPU seconds, and that is a large part of why the later parts read the way they do.

A **theory paper** is a third kind, and the two Denuit, Dhaene, et al. (2025) and Denuit, Ortega-Jiménez, et al. (2025) chapters are the instances. These prove results about the conditional expectation given the sum, $\mathsf{E}[X_i \mid S = s]$, and illustrate them with figures rather than tables. There is no insurance contract in either, and no compound distribution: each agent carries one claim with certainty, so what is under test is the allocation kernel and the grid it runs on. Success here means reproducing a *shape*, and it is checkable only because both papers state something numerical about the shape, a turning point in one case and a monotonicity failure in the other. Reproducing a figure is weaker evidence than reproducing a table, so these chapters lean on constraints the model must satisfy whatever the picture looks like: that shares add up, and that identically distributed agents get identical shares.

Conclusions: correctness, flexibility, errors, and exactness

The reproductions exercise confirms `aggregate`'s correctness and flexibility. Venter (1983)'s Exhibit 3 reproduces in all three columns, including the transformed gamma, which is `scipy.stats.gengamma` under a different name. Bruno et al. (2006)'s Tables 6, 7 and 8 come back exact at every printed decimal. Bodoff (2017)'s Exhibit 4 reproduces cell for cell. Homer and Clark (2003) reproduces every published cell of sections 2 and 4, and 136 of the 147 cells of section 3. Bear and Nemlick (1990)'s Table I reproduces in both of its reproducible columns, six adjustable features priced by six DecL clauses. Pittarello et al. (2024)'s accuracy table measures where the grid ends rather than how finely it is spaced, and on a long enough grid the error falls as h^2, in the opposite direction to the one the paper reports. Gollini and Rougier (2016)'s 32,060 event hurricane loss table, where the authors' Panjer recursion stopped at a \$100 rounding after 4,651 seconds, is computed unrounded in under a second, and the moment bound they recommend turns out to be four to ten times the exact tail probability.

The exercise uncovered small errors in six papers. Mata and Verheyen (2005)'s Table 1 limited expected values are not the exact lognormal ones, running 0.04% high at \$250,000 and drifting to 0.24%

low at $5M, while the closed form matches to eight figures. Homer and Clark (2003)'s Table 3.3 does not sum to one, carrying 99.97% of the mass. Asmussen et al. (2016) has a row where its two published saddlepoint columns and its Monte Carlo column disagree, and `aggregate`, being neither a saddlepoint expansion nor a sampler, can determine which one to trust. Denuit, Dhaene, et al. (2025)'s Figure 1 cannot hold at its right hand edge: agent 3's loss is capped at 30, so adding up forces agents 1 and 2 above 5 at a pool loss of 40, and their panels are drawn on an axis whose top tick is 4. Pittarello et al. (2024)'s Table 3 prints the same reference value in all four rows for four and five dimensions, a distribution function that would have to be flat from 10 to 10,000; Arbenz et al. (2011), where the values come from, has the right ones. Gollini and Rougier (2016) report Monte Carlo samples of 10^5 in the text, while their own design calculation, their summary and the script that drew their figures all use 10^4.

Finally, it reveals the FFT computational resolution floor. Ben Rached et al. (2024)'s Theorem 3 predicts that FFT convolution error is bounded absolutely rather than relatively, so below roughly 10^{-13} the output is noise regardless of grid refinement. `aggregate` sits on that predicted floor to within a factor of a few. Wilson and Keich (2016) shows the same failure on adversarial inputs and, more usefully, shows that it is not always detectable: a wrong answer can come back positive and plausible. Wilson and Keich (2017) exhibits it on a benign input and demonstrates the repair. Together they turned a vague caveat into a measured property, and explain why the far tail now has documented limits rather than optimistic ones.

Computational accuracy is discussed further in two other recent blog posts. Dice of Dice: The FFT Meets Exact Arithmetic compares FFT computed probabilities with an exact arithmetic probability calculation, and Dice of Dice: Exact Finite FFTs discusses using number-theoretic transforms which give exact rational number answers with no floating point noise. The latter distinguishes between the Fourier method which is exact and different numerical implementations. That method now ships in the library as the `nttagg` kind, and the Wilson and Keich (2016) and Wilson and Keich (2017) chapters use it: their exact reference values, previously hand algebra in one case and trusted naive convolution in the other, are computed in-library as exact rational numbers, and the float pipeline's error against them is itself computed exactly, per bucket. An optional compiled engine, the Rust add-in `aggregate-ntt` working over 64-bit primes, extends the reach of the exact method; this book runs the pure-Python reference path.

The chapters

Table 1 and Table 2 list the chapters in book order, split at the part boundary between the actuarial chapters and the numerical ones: the paper each chapter reproduces, where it was published and how often it has been cited, what the chapter targets, and how it came out.

Table 1.: The chapters on actuarial models, distortion pricing and risk sharing, in book order, with each paper's citation count, the chapter's target, and the outcome.

Part	Paper	Year	Journal	Citations	Target	Outcome
Actuarial models	Venter (1983)	1983	Proceedings of the Casualty Actuarial Society	109	Exhibit 3, three methods over 34 rows	Reproduced. The bucket size selects which of the two exact columns you land on
	Mack and Fackler (2003)	2003	Blätter der DGVFM	10	Table 1, and the threshold above which Riebesell is exact	Reproduced. All 26 cells within 8e-4, and the threshold is 5.186 $\mathsf{E}[X]$ against the paper's "about 5"
	Mata and Verheyen (2005)	2005	CAS Forum	10	Tables 1 to 14, trend and exposure adjustment by layer	Reproduced, and a discrepancy found in the paper's Table 1
	Bodoff (2017)	2017	Variance	0	Exhibit 4, excess of policy limits	Reproduced. Two unstated model choices inferred and confirmed by fit
	Homer and Clark (2003)	2003	Proceedings of the Casualty Actuarial Society	14	All three worked examples of the bivariate FFT	Reproduced. Every construction is a shipped DecL form
	Bear and Nemlick (1990)	1990	Proceedings of the Casualty Actuarial Society	10	Table I, six treaties with adjustable features	Reproduced. The lognormal column exactly, the collective risk column to the printed digit in five of six
Distortion pricing	Wang (2000)	2000	Journal of Risk and Insurance	676	Table 1, risk load by layer under two distortions	Reproduced to 0.31%, and the basic limit expected loss is misprinted
	Wang (2002)	2002	ASTIN Bulletin	268	Table 2, weather derivative calls off 22 observations	Reproduced. Prices to under half a cent
Risk sharing	Denuit (2019)	2019	ASTIN Bulletin	63	Figure 1 and the shares at $s = 1$, four participant pool	Reproduced. The horizontal line resolves a table that contradicts its own text
	Denuit et al. (2022)	2022	ASTIN Bulletin	18	Example 4.5, Figure 6, mortality credits at three pool sizes	Reproduced. Shares add up exactly at every pool size
	Denuit, Dhaene, et al. (2025)	2025	Insurance: Mathematics and Economics	5	Figure 1, the comonotonic repair of a conditional mean allocation	Reproduced, and the published figure cannot hold at its right hand edge
	Denuit, Ortega-Jiménez, et al. (2025)	2025	ASTIN Bulletin	2	Figures 2, 5 and 6, conditional expectation given the sum	Reproduced. The turning point lands at 64.46, which is what settles the index convention

Table 2.: The chapters on exact computation, rare events and the paper that did not reproduce, in book order, with each paper's citation count, the chapter's target, and the outcome.

Part	Paper	Year	Journal	Citations	Target	Outcome
Exact computation	Gerber (1982)	1982	Insurance: Mathematics and Economics	69	Tables 1 to 3, the exact stop loss premium columns	Reproduced. All 28 values inside the printed precision
	Robertson (1992)	1992	Proceedings of the Casualty Actuarial Society	25	Table 4, the piecewise uniform algorithm	Partial. Exact for the paper, $O(h)$ for a lattice, so the printed precision is out of reach
	Embrechts et al. (1993)	1993	Statistica Neerlandica	42	Figure 1, discretization error and Richardson extrapolation	Reproduced. The $O(h^2)$ rate measured at 3.998
	Grübel and Hermesmeier (1999)	1999	ASTIN Bulletin	50	Tables 1 and 2, the L1 sums, the Theorem 2 bound	Reproduced, including the bound at 0.4641
	Pittarello et al. (2024)	2024	Annals of Actuarial Science	1	Section 3.5 examples, Tables 1 to 3	Reproduced. Table 1 is a truncation study, Table 2 re-timed on one machine, Table 3's reference values misprinted
	Embrechts and Frei (2009)	2009	Mathematical Methods of Operations Research	94	Table 1, three arithmetization designs	Reproduced exactly, all twelve values
	Schaller and Temnov (2008)	2008	Computational Methods in Applied Mathematics	23	Table 4.3, operational risk quantiles	Reproduced to 2.2e-4, all twelve distinct values
	Bruno et al. (2006)	2006	Applied Mathematics and Computation	8	Tables 6, 7 and 8	Reproduced exact at six decimals. 92,832 convolutions in seconds against the paper's 1,950
	Gollini and Rougier (2016)	2016	The Journal of Operational Risk	1	Figures 1 to 3 and 5 to 7, bounds on a 32,060 event hurricane ELT	Reproduced. The exact answer the authors ruled out takes under a second, and the moment bound is 4 to 10 times high
	Wilson and Keich (2016)	2016	Computational Statistics and Data Analysis	8	Table 2, the FFT-C columns	Reproduced. The finding is detectability, not error size. The Section 1 example is now computed exactly in-library
	Wilson and Keich (2017)	2017	Journal of Computational and Graphical Statistics	5	The worst case, and its repair	Reproduced to the digit and the repair demonstrated, both now certified against exact arithmetic
Rare events	Asmussen et al. (2016)	2016	Scandinavian Journal of Statistics	36	Twelve tables, CDF and PDF	Reproduced from five build calls, and one bad row adjudicated
	Ben Amar et al. (2023)	2023	Statistics and Computing	7	The seven probabilities the paper actually quotes	Reproduced. The whole left tail curve costs what one estimate costs them
	Ben Rached et al. (2024)	2024	Statistics and Computing	0	Table 6, and Theorem 3's floor	Reproduced, and the floor confirmed on the library itself
	Liu et al. (2026)	2026	Operations Research Letters	0	Tables 1, 2 and C.1	Three of four reference points reproduced. The fourth is below the floor
	Commentary				The rare-event group as a whole	Where these methods earn their keep, and where they do not
Did not reproduce	Jin et al. (2016)	2016	Scandinavian Actuarial Journal	14	Nothing	Not a reproduction. The evidence, and the feature that would change it

Counts are from OpenAlex, retrieved 2026-08-08 (for Pittarello et al. (2024), 2026-10-11, and for Gollini and Rougier (2016), 2026-10-21), except for the Casualty Actuarial Society (CAS) literature. The CAS papers are not indexed and their counts come from a manual Google Scholar lookup. The counts are a snapshot and do not auto-update on rebuild. They should be read relative to the year of publication, and they say how widely each result has been taken up, which is a different question from whether it reproduces.

The two most cited papers here by an order of magnitude, Wang (2000) and Wang (2002), are the two whose reproductions are the shortest and least eventful. The papers that changed the library, Ben Rached et al. (2024), Wilson and Keich (2016) and Wilson and Keich (2017), have 0, 8 and 5 citations between them. Thus, being useful to an implementation and being cited are close to independent!

The final chapter is reference rather than reproduction: it records where each published severity curve shipped in `library.agg` came from, what the source paper uses it for, and every transcription decision behind the DecL.

Reading and rebuilding

Each chapter stands alone and they can read independently. Two pairs reward being read together: Homer and Clark (2003) and Jin et al. (2016) sit on opposite sides of the same feature boundary, dependence through a joint severity under a shared count against dependence through a common shock on the counts, and Wilson and Keich (2016) and Wilson and Keich (2017) are companion papers on the same failure, the first on adversarial inputs and the second on ordinary ones.

Rendering requires a Python environment with `aggregate` and its documentation extras installed, since every chapter executes. The two working documents in this folder, `README.md` and `plan-examples.md`, are notes rather than chapters and are deliberately outside the book: the first is a folder index and the second is the candidate assessment, which carries the tier ratings for the papers that were considered and rejected.

Part I.

The short version

1. Five papers, five programs

1.1. How to read this chapter

This section presents a DecL solution to problems from five published results. Each has the same four parts: what the paper was doing, the exhibit it printed, the DecL program that reproduces it, and the answer beside the paper's own numbers.

The program on each page is not typed into the page. It is read back off the object that produced the numbers, so the two cannot drift apart. Nothing else is shown, because nothing else is needed: no discretization, no grid loop, no convolution. The bucket size and the grid, the two numerical inputs that matter, are the library's self-computed defaults in four of the five reproductions. Choosing them well is itself one of the library's jobs, and these pages show it doing it. Venter alone states them, through the `hints{}` clause visible in its program, and for two reasons: his exhibit tabulates at multiples of $25,000, which the default bucket of 80 does not divide, and the argument of that section turns on comparing two deliberately chosen resolutions.

The five results are chosen to make five different points. Venter (1983) is about accuracy against a paper that computed the same thing two other ways. Bear and Nemlick (1990) shows how the library can model standard reinsurance variable features: it models six treaty features in six clauses. Homer and Clark (2003) is about reach, a joint distribution that is a view of a program rather than a second model. Bruno et al. (2006) is about speed. Denuit, Dhaene, et al. (2025) is about the reproduction checking the paper rather than the other way round, and it is the one target here that is not an insurance pricing problem at all.

The rest of this book is the detail: the transcription decisions, the readings that had to be inferred, and the cases that did not work. The detail includes a lot of Python scaffold coding that distracts from the simple DecL solution. This chapter gives just the DecL highlights and omits the scaffolding.

```
Exception reporting mode: Minimal
```

1.2. Venter (PCAS, 1983): Transformed Beta And Gamma Distributions And Aggregate Losses

Venter (1983) introduced the transformed gamma to actuarial work and closed by testing it against the exact aggregate distribution. His Exhibit 3 computes the cumulative probability distribution of one compound Poisson three ways over $25,000 to $850,000: by characteristic function inversion, by the Adelson and Panjer recursion, and by a transformed gamma matched to three moments, see Figure 1.1. Frequency is Poisson with $\lambda = 13.7376$ and severity is a piecewise linear claim size CDF to a $250,000 policy limit, tabulated at 23 points.

EXHIBIT 3
PART 1

AGGREGATE LOSS DISTRIBUTIONS
COMPARATIVE SUMMARY

Aggregate Loss ($000)	Characteristic Function Method		Recursive Method		Transformed Gamma	
	Cumulative Probability	Excess Ratio	Cumulative Probability	Excess Ratio	Cumulative Probability	Excess Ratio
25	.0508	.9016	.0516	.9016	.0621	.9031
50	.1291	.8107	.1298	.8107	.1260	.8125
75	.2009	.7273	.2015	.7272	.1895	.7283
100	.2676	.6507	.2683	.6507	.2520	.6503
125	.3289	.5806	.3295	.5806	.3129	.5786
150	.3843	.5163	.3848	.5163	.3717	.5129
175	.4341	.4573	.4346	.4573	.4280	.4529
200	.4788	.4030	.4793	.4029	.4817	.3984
225	.5189	.3529	.5193	.3529	.5324	.3491
250	.5548	.3066	.5552	.3066	.5801	.3047
275	.6034	.2642	.6040	.2642	.6245	.2650
300	.6556	.2273	.6561	.2273	.6658	.2295
325	.7008	.1951	.7013	.1951	.7039	.1981
350	.7405	.1672	.7408	.1672	.7388	.1702
375	.7749	.1431	.7752	.1431	.7707	.1457
400	.8047	.1221	.8049	.1221	.7995	.1243
425	.8303	.1039	.8305	.1039	.8255	.1055
450	.8524	.0880	.8526	.0880	.8488	.0893
475	.8714	.0742	.8716	.0742	.8696	.0752
500	.8878	.0622	.8879	.0622	.8881	.0631
525	.9045	.0518	.9047	.0518	.9043	.0528
550	.9201	.0430	.9203	.0430	.9186	.0439
575	.9332	.0357	.9333	.0357	.9310	.0364
600	.9442	.0296	.9443	.0296	.9418	.0301
625	.9534	.0245	.9535	.0245	.9511	.0247
650	.9611	.0202	.9611	.0202	.9592	.0203
675	.9675	.0167	.9675	.0167	.9660	.0165
700	.9728	.0137	.9729	.0137	.9718	.0134
725	.9773	.0112	.9773	.0112	.9768	.0109
750	.9810	.0091	.9810	.0091	.9809	.0088
775	.9844	.0074	.9844	.0074	.9844	.0070
800	.9873	.0060	.9873	.0060	.9873	.0056
825	.9897	.0048	.9897	.0048	.9897	.0045
850	.9916	.0039	.9916	.0039	.9917	.0035

Figure 1.1.: Venter (1983), Exhibit 3, Part 1.

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
Venter: aggregate PMF deficit 1.302e-04 (Σp = 1 − 1.302e-04 < 1); forwards and backwards S diverge
by the deficit (forwards > backwards). Further occurrences this session are suppressed; each
object records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

```
agg Venter
  13.7376 claims
  250000 xs 0
  sev chistogram xps [0 1000 5000 6000 7000 8000 9000 10000 12500 15000 17500 20000 25000 35000
  50000 75000 100000 125000 150000 175000 200000 225000 250000 300000] [0.38935 0.38935 0.00568
  0.00543 0.00517 0.00495 0.00473 0.01098 0.00989 0.00896 0.00815 0.01426 0.02237 0.02353 0.02459
  0.01517 0.01021 0.00732 0.00547 0.00423 0.00337 0.00274 0.0241]
  poisson
  hints{log2=16; bs=500; normalize=False; padding=1}
```

The small bucket size does not have full extent and triggers the (harmless) DefectiveDistributionWarning. The validation explains.

```
print(textwrap.fill(v5.validation_explanation, 72))
```

```
Validation of the realized grid against the analytic moments: fails pmf
deficit 1.302e-04. Each of mean, CV and skewness is compared for
severity and for the aggregate, and a relative error above 0.0001 fails;
only the lowest-order failure is reported, since a mean that is wrong
makes the higher moments uninformative. The realized law does not sum to
1: 1.302e-04 of the probability ran off the top of the grid and was
dropped rather than wrapped. Forwards S = 1 - cumsum carries the missing
mass as a tail blob and backwards S drops it, so the two pricing routes
disagree by exactly this amount and neither is wrong. Raise log2, or
widen the grid.
```

Table 1.1.: Validation of part of Venter Exhibit 3, Part 1.

aggregate loss	Venter char fn	Venter recursive	agg bs=500	agg bs=5	agg bs=100.0
225,000	0.5189	0.5193	0.5193	0.5189	0.5190
250,000	0.5548	0.5552	0.5552	0.5548	0.5549
275,000	0.6034	0.6040	0.6040	0.6034	0.6035
300,000	0.6556	0.6561	0.6561	0.6556	0.6557
325,000	0.7008	0.7013	0.7013	0.7008	0.7009
350,000	0.7405	0.7408	0.7408	0.7405	0.7405
375,000	0.7749	0.7752	0.7752	0.7749	0.7750
400,000	0.8047	0.8049	0.8049	0.8047	0.8047

Table 1.1 shows eight of the thirty four rows, at the middle of the range. Table 1.2 measures the fit over all thirty four.

Table 1.2.: Largest absolute difference from each of Venter's two exact columns, over the full range.

rows compared	agg bs=500 vs his recursive column	agg bs=5 vs his characteristic function column
34	5.1e-05	7.3e-05

Venter presents two exact *methods* that differ by up to 0.0008, and reads that gap as the cost of discretizing severity. Forty years later we can reproduce both with one method using two resolutions. Change `bs=500` to `bs=5`, one token in one program, and `aggregate` stops agreeing with his recursion and starts agreeing with his characteristic function instead. The 0.0008 was never a difference between the methods. It was the $500 bucket, and it is the same error `aggregate` incurs when asked for the same bucket.

1.3. Bear and Nemlick (1990): six treaty features, six clauses

Bear and Nemlick (1990) price six reinsurance treaties that each carry one adjustable feature, comparing a lognormal approximation of the aggregate against the collective risk model. Their Table I, Figure 1.2, shows the six answers. The subject book is two excess casualty classes written at $160,000 xs 0 with Pareto severities and gamma mixed Poisson counts. Treaty I puts an annual aggregate deductible of $360,000 against $450,000 of expected layer loss, an entry ratio of 0.800, and the indicated rate is the loss cost surviving the deductible loaded 100/75.

TABLE I

COMPARISON OF KEY ITEMS

Example	Unadjusted Rate	Item Compared	Lognormal Model	Collective Risk Model $c = 0$ $b = 0$	Collective Risk Model $c = .05$ $b = .05$	Collective Risk Model $c = .05$ $b = .10$	Collective Risk Model $c = .10$ $b = .10$
I) Aggregate Deductible	5.00%	Adjusted Rate	1.47%	1.58%	1.68%	1.73%	1.77%
II) Limited Reinstatement	25.00	Adjusted Rate	19.53	19.89	19.72	19.55	19.52
III) Loss Corridor	5.00	Adjusted Rate	4.02	3.67	3.71	3.74	3.73
IV) Retro Rating Plan	5.00	Expected Rate After Retro Adjustments	5.02	5.20	5.18	5.14	5.14
V) Profit Commission		Expected Profit Commission	8.37	8.24	8.50	8.69	8.75
VI) Sliding Scale Commission		Expected Sliding Scale Commission	31.04	30.31	30.90	31.22	31.33

Figure 1.2.: Bear and Nemlick (1990), Table I.

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  poisson
  aggregate net of
    360 xs 0
```

An aggregate deductible is an aggregate layer the cedant keeps, so the recovery is the aggregate net of it, and the whole feature is the last clause. Nothing else in the program changes.

Table 1.3.: Treaty I, the annual aggregate deductible, against the two columns of Table I.

expected layer loss, 000s	insurance charge, exact	indicated rate %, agg	Table I, collective risk	Table I, lognormal
450.0	0.3151	1.5755	1.580	1.470

In Table 1.3 the exact answer is 1.5755% against 1.58% printed, and the lognormal shortcut the paper was testing reads 1.47%. The other five treaties are one clause each, as Table 1.4 sets out, and the detail chapter prices all six.

Table 1.4.: The adjustable feature of each Bear and Nemlick treaty, and the DecL clause that expresses it.

Treaty	Feature	DecL clause
I	annual aggregate deductible	`aggregate net of 360 xs 0`
II	three free reinstatements	`reinstatements 3 free`
III	loss corridor	`corridor`
IV	retrospective rating	`retro`
V	three year profit commission	`pc`
VI	sliding scale commission	`slide`

Six treaties, six clauses, and no arithmetic outside the loading factors the slips specify. The detail chapter reproduces the lognormal column exactly at every printed digit and the collective risk column to the printed digit in five of the six.

Here are all six programs, at Poisson counts, as the detail chapter writes them. They are worth reading side by side: every one is the same shape, a subject book with a feature clause on the end, and the differences between the six treaties are differences of one clause. Five wrap in an `xpnl`, the walk face, which books each cession as its own step so the recovery, the ceded premium and any commission come back as separate legs. Treaty IV uses `pnl` instead, because a retro plan varies the gross premium rather than ceding anything, so there is no cession to walk and the plan replaces the premium at the head of the program.

These are shown as authored rather than as parsed, which is the one place in this chapter where that distinction matters. The canonical form the parser renders back spells several of these clauses differently: `reinstatements 3 free` is stored as a vector of three zero charges and comes back `reinstatements [0 0 0]`, percentages resolve to decimals so `corridor 100% po 100% xs 100%` comes back `corridor 1 po 1 xs 1`, and `exposure at rate` and `premium at lr` are one construct with `premium` as the canonical spelling. Everything downstream is driven by the parse, not by this text. The slip language is what is worth reading here.

Treaty I: annual aggregate deductible

```
agg BN.One [9000 3000] exposure at [0.04 0.03] rate 160 xs 0
    sev 40 * pareto [0.90 0.95] - 40
    poisson
    aggregate net of 360 xs 0
```

Treaty II: three free reinstatements

```
xpnl BN.TreatyII 1172 premium less
    agg BN.Two [2000 2000 2000] exposure at [0.10 0.14 0.21] rate 700 xs 0
        sev 300 * pareto [1.5 1.3 1.1] - 300
        occurrence net of 700 xs 0 deposit 1172 reinstatements 3 free as "Layer"
        poisson
```

Treaty III: loss corridor

```
xpnl BN.TreatyIII 10000 premium less
    agg BN.Three [4500 4500 1000] exposure at [0.032 0.038 0.035] rate 400 xs 0
        sev 100 * pareto 1.1 - 100
        poisson
        aggregate net of 10000 xs 0 deposit 350
                       corridor 100% po 100% xs 100% as "Corridor"
```

Treaty IV: retrospective rating

```
pnl BN.TreatyIV retro basic 0 lcm 1.3333333333 min 360 max 1200 premium less
    agg BN.Four [9000 3000] exposure at [0.04 0.03] rate 160 xs 0
        sev 40 * pareto [0.90 0.95] - 40
        poisson
```

Treaty V: three year profit commission

```
xpnl BN.TreatyV 18000 premium less
    agg BN.Five [6000 6000 6000] exposure at [0.10 0.14 0.21] rate 700 xs 0
        sev 300 * pareto [1.5 1.3 1.1] - 300
        poisson
        aggregate net of 80% po 100000 xs 0 deposit 5625
                       pc 25% after 20% as "PC layer"
```

Treaty VI: sliding scale commission

```
xpnl BN.TreatyVI 25000 premium less
    agg BN.Six 25000 exposure at 0.10 rate 900 xs 0
        sev 100 * pareto 0.98 - 100
        poisson
        aggregate net of 100000 xs 0 deposit 5000
                       slide 40% at 35% and 25% at 55% and 20% at 65% as "Slide"
```

Three details these programs make visible that the prose above glosses over. The nominal wide layers in III, V and VI (`10000 xs 0`, `100000 xs 0`) are not terms of the treaty: a decorator clause has to sit on an aggregate layer, so the layer is written high enough to reach everything the aggregate does and the feature supplies the only binding term. The `deposit` in each is the base premium the feature prices against, which is why the corridor's `100% po 100% xs 100%` reads in loss ratio points exactly as the slip does. And the `80% po` in Treaty V is a placement rather than a cession: premium clauses are quoted at 100% and scaled by the fraction placed, so the slip's $4,500 on an 80% placement is a `deposit` of $5,625.

The one genuine constraint is that each program carries exactly one feature. Three of the five features read a ceded loss ratio, and that ratio is only a deterministic rescaling of the ceded loss while the denominator stays fixed. Put two features on one program and the second reads a random denominator. The paper reaches the same conclusion from the other side in its section 5, and proposes running the collective risk model a second time on the post-coinsurance distribution: a two stage construction, not a single decorated layer.

1.4. Homer and Clark (2003): the joint distribution is a view, not a second model

Homer and Clark (2003) introduces the two dimensional FFT to actuarial work, for the case where two covers on the same book are driven by the same losses and so cannot be priced separately. Section 2 retains $600,000 per occurrence, buys $400,000 xs $600,000 per occurrence, and puts a $5,000,000 xs $3,000,000 stop loss on the retention. The paper builds the joint law of retained and ceded by hand, as its Table 2.3, Figure 1.3.

TABLE 2.3

SINGLE CLAIM PRIMARY & EXCESS LOSS BIVARIATE DISTRIBUTION

Loss Capped at 600,000	Loss Excess of 600,000					
	0	200,000	400,000	600,000	800,000	1,000,000
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
200,000	**37.80%**	0.00%	0.00%	0.00%	0.00%	0.00%
400,000	**23.50%**	0.00%	0.00%	0.00%	0.00%	0.00%
600,000	**14.60%**	**9.10%**	**15.00%**	0.00%	0.00%	0.00%
800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 1.3.: Homer and Clark (2003), Table 2.3.

```
agg HC
  5 claims
  dsev [200000 400000 600000 800000 1000000] [0.378 0.235 0.146 0.091 0.15]
  occurrence net of
    400000 xs 600000
  mixed gamma 0.2
```

That program declares the treaty and nothing two dimensional. Retained and ceded are the net and ceded views of an occurrence layer already in it, so the joint severity is the cession map applied to the gross severity, and the joint distribution is one call away: `a.occ_bivariate(views=('net', 'ceded'))`. The paper's Table 2.3 falls out as a by-product rather than an input.

Table 1.5.: The five statistics Homer and Clark read off Table 2.3, taken instead from the bivariate view of the program.

statistic	**aggregate**	**paper**
expected loss to the stop loss layer	123,529.2617	123,529.0000
probability the stop loss is hit	0.1508	0.1508
mean stop loss given it is hit	819,209.9114	819,210.0000
expected loss to the per-occurrence layer	390,999.9993	391,000.0000
expected per-occurrence loss given the stop loss is hit	830,334.0200	830,334.0000

Table 1.5 reproduces all five to the digit, and the last two are the paper's point. The per-occurrence layer costs $391,000 on average but $830,334 in the scenarios where the stop loss also pays. Writing both covers is more than twice as bad as the two expected values suggest, and that is invisible to anyone pricing them separately.

1.5. Bruno, Camerini, Manna and Tomassetti (2006): 92,832 convolutions

Bruno et al. (2006) proposes thresholded direct convolution as a cheaper alternative to Panjer recursion, De Pril transforms and the FFT. Table 7, Figure 1.4, is the showcase: a Poisson with $\lambda = 91{,}000$ against a fourteen point discrete severity, which their formalization reaches in 92,832 convolutions. Table 4 records that Panjer recursion and De Pril both underflow at λ as low as 1,004.8, that their own method took 1,950 seconds here, and that their FFT comparison stopped at $\lambda = 5{,}004.8$ after 3,850 seconds.

Table 7
Aggregate claims cumulative distribution using a Poisson counting distribution with parameter $\lambda = 91{,}000$

Aggregate claims amount[a]	Cumulative distribution values
$\mu - 5.00\sigma \cong 3{,}820{,}935$	0.000000
$\mu - 4.00\sigma \cong 3{,}835{,}002$	0.000030
$\mu - 3.00\sigma \cong 3{,}849{,}069$	0.001327
$\mu - 2.50\sigma \cong 3{,}856{,}102$	0.006149
$\mu - 2.00\sigma \cong 3{,}863{,}135$	0.022643
$\mu - 1.90\sigma \cong 3{,}864{,}542$	0.028604
$\mu - 1.80\sigma \cong 3{,}865{,}949$	0.035817
$\mu - 1.75\sigma \cong 3{,}866{,}652$	0.039943
$\mu - 1.70\sigma \cong 3{,}867{,}355$	0.044447
$\mu - 1.65\sigma \cong 3{,}868{,}059$	0.049359
$\mu - 1.60\sigma \cong 3{,}868{,}762$	0.054687
$\mu - 1.50\sigma \cong 3{,}870{,}169$	0.066705
$\mu - 1.00\sigma \cong 3{,}877{,}202$	0.158658
$\mu - 0.75\sigma \cong 3{,}880{,}718$	0.226704
$\mu - 0.50\sigma \cong 3{,}884{,}235$	0.308708
$\mu - 0.25\sigma \cong 3{,}887{,}752$	0.401538
$\mu - 0.00\sigma \cong 3{,}891{,}268$	0.500249
$\mu + 0.25\sigma \cong 3{,}894{,}785$	0.598941
$\mu + 0.50\sigma \cong 3{,}898{,}302$	0.691642
$\mu + 0.75\sigma \cong 3{,}901{,}818$	0.773450
$\mu + 1.00\sigma \cong 3{,}905{,}335$	0.841343
$\mu + 1.50\sigma \cong 3{,}912{,}368$	0.933083
$\mu + 2.00\sigma \cong 3{,}919{,}402$	0.977145
$\mu + 2.50\sigma \cong 3{,}926{,}435$	0.993730
$\mu + 3.00\sigma \cong 3{,}933{,}468$	0.998627
$\mu + 4.00\sigma \cong 3{,}947{,}535$	0.999967
$\mu + 5.00\sigma \cong 3{,}961{,}602$	1.000000
$\mu + 8.00\sigma \cong 4{,}003{,}802$	1.000000

[a] $\mu = 3891268.954300$ and $\sigma = 14066.631372$. They are, respectively, the mean value and the standard deviation of the aggregate claims distribution.

Figure 1.4.: Bruno et al (2006), Table 7.

```
agg B7
  91000 claims
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.0103301 0.030799 0.0293511 0.0103301
  0.0730414 0.0111568 0.0264554 0.1002133 0.0815418 0.0252146 0.0212857 0.0254214 0.0991756
  0.4556837]
  poisson
```

The program carries no `hints{}` clause, so the grid is the library's own. That is possible because the aggregate is computed in its support window rather than from zero, and at $\lambda = 91{,}000$ that window is a narrow interval around a mean of 3.89 million. Table 1.7 reports the grid the library chose for it, and the whole of Table 7 comes back exact on that grid.

Table 1.6.: Eight of the twenty eight aggregate loss points Table 7 prints, against the program's CDF.

aggregate loss	**agg CDF**	**Table 7 CDF**
3,877,202	0.158658	0.158658
3,880,718	0.226704	0.226704
3,884,235	0.308708	0.308708
3,887,752	0.401538	0.401538
3,891,268	0.500249	0.500249
3,894,785	0.598941	0.598941
3,898,302	0.691642	0.691642
3,901,818	0.773450	0.773450

Table 1.6 shows eight of the printed points; Table 1.7 takes in all of them, and the clock.

Table 1.7.: Accuracy over all twenty eight rows, and the run time beside the paper's own.

rows compared	largest CDF difference	aggregate seconds	paper seconds	grid points	mean	paper mean
28	4.8e-07	0.10000	1,950	262,144	3,891,268.9543	3,891,268.9543

All twenty eight published values are exact at the six decimals printed. The paper counted the FFT as one transform per convolution and summed over 92,832 of them, which is not how a compound distribution is computed: the frequency enters through its probability generating function applied pointwise to the severity transform, so the whole distribution costs two transforms whether λ is 500 or 91,000. Some of the timing gap is twenty years of hardware but that structural part is not.

1.6. Denuit, Dhaene, Ghossoub and Robert (2025): the check runs both ways

Denuit, Dhaene, et al. (2025) settle a question about risk sharing rather than insurance pricing. Given a pool of losses $S = X_1 + \cdots + X_n$, the conditional mean risk sharing rule allocates $\kappa_i(s) = \mathsf{E}[X_i \mid S = s]$ to agent i. That rule is not always **comonotonic**: an agent's share can *fall* as the pool's total loss rises, which violates the no-sabotage condition and is not Pareto optimal for the convex order. The paper's contribution is an explicit algorithm, from majorization theory, that converts any such allocation into a comonotonic one that improves every agent in convex order while preserving every expectation. Its section 3.2 example is three independent agents: agents 1 and 2 carry a truncated exponential on $[0, 10]$ with $\beta = 1/2$, and agent 3 a truncated equal mixture of an exponential and a $\Gamma(8)$ on $[0, 30]$.

This is a natural reproduction target because κ_i is not a special quantity here. It is the `exeqa_` column of a portfolio's `density_df`, the same conditional expectation `aggregate` uses for capital allocation, and the paper's algorithm ships as `Portfolio.make_comonotonic_allocations`. Figure 1.5 is the target.

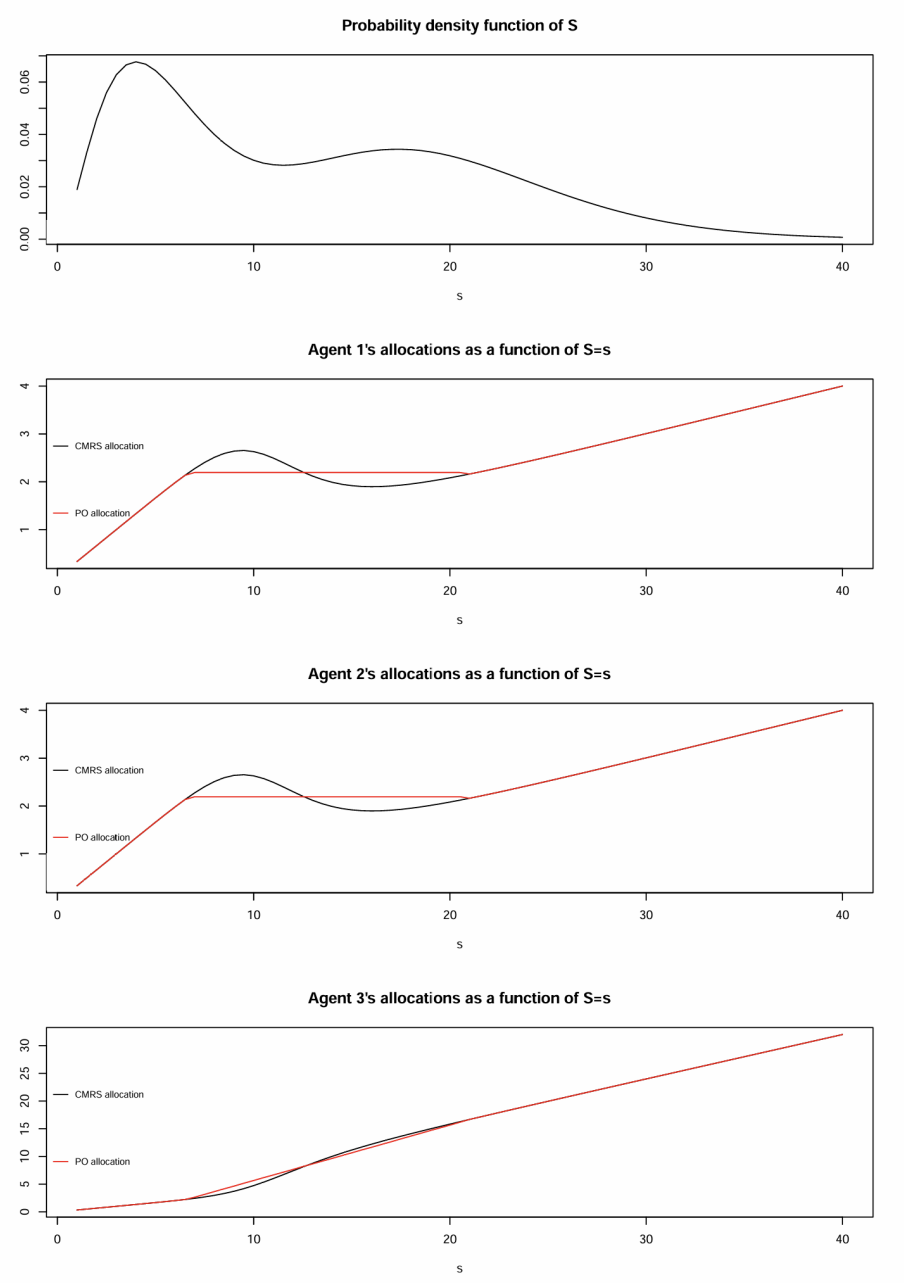


Figure 1: Plots of the probability density function of S and of the agents' allocations.

Figure 1.5.: Denuit, Dhaene, Ghossoub and Robert (2025), Figure 1.

```
port Denuit.Pool
  agg Agent1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent2
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent3
    1 claim
    sev 2 * gamma [1 8] wts [0.504541349633 0.495458650367] splice [0 0] [30 30]
```

```
    fixed
```

A truncated distribution is a `splice`, and the mixture is a two component severity, so the whole three agent pool is one portfolio declaration. Before reading any allocation, check the model against the paper's own specification, which Table 1.8 does.

Table 1.8.: The pool's means against direct quadrature on the paper's own specification.

quantity	**quadrature**	**aggregate**
E[X_1], truncated exponential on [0, 10]	1.93216345	1.93216337
E[X_3], truncated mixture on [0, 30]	8.77944917	8.77944913
E[S]	12.64377607	12.64377587

Two structural facts follow from the specification and hold to numerical precision in Table 1.9, which is what says the pool is built right. Agents 1 and 2 are identically distributed, so $\kappa_1 \equiv \kappa_2$; and an allocation adds up, so $\sum_i \kappa_i(s) = s$ at every s.

Table 1.9.: The two structural invariants, symmetry of agents 1 and 2 and adding up, at numerical precision.

$\max \lvert\kappa_1 - \kappa_2\rvert$ **over** $s \in [0.5, 45]$	$\max \lvert \sum \kappa_i - s\rvert$
0.0e+00	1.1e-10

Now the figure, Figure 1.6. The black curves are the conditional mean allocations, the red the comonotonic Pareto optimal improvement from the paper's algorithm.

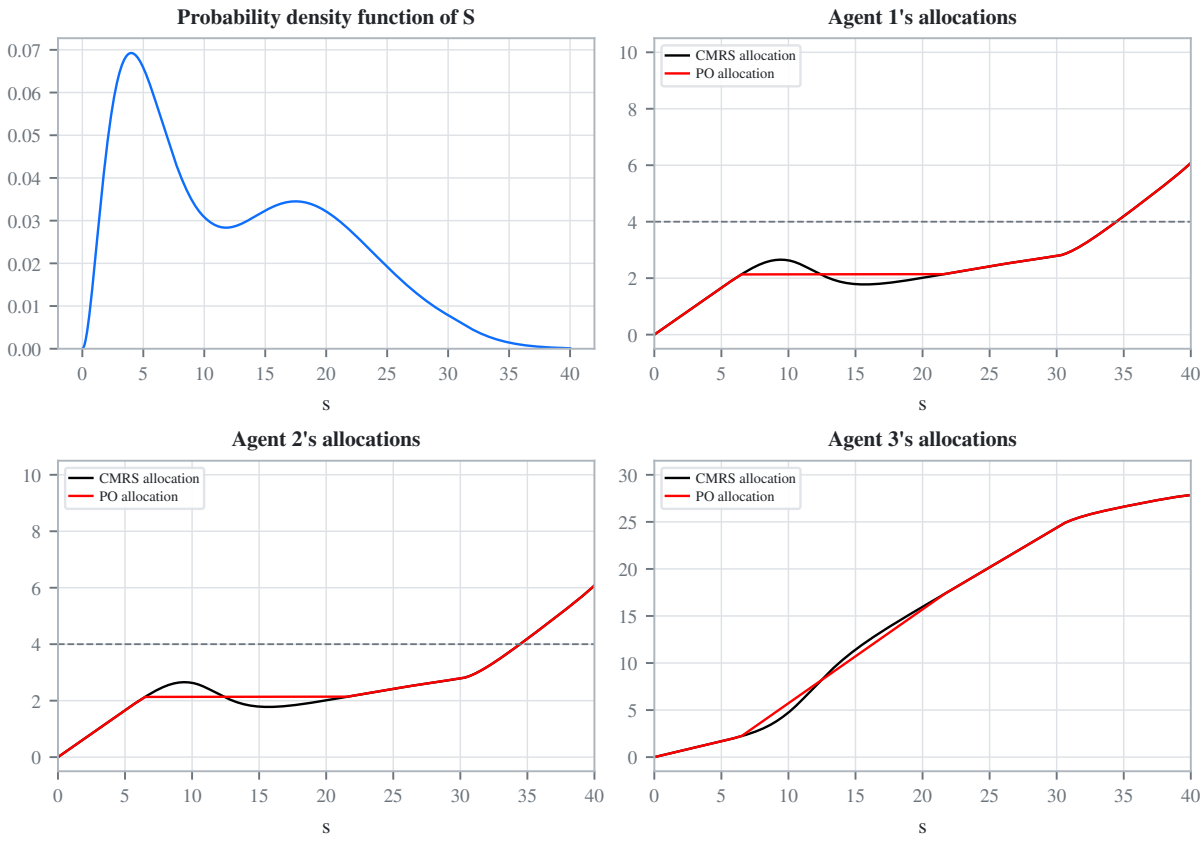


Figure 1.6.: Reproducing Figure 1. The dashed line on the agent 1 and 2 panels is the top of the published axis.

Table 1.10.: The conditional mean allocations at seven values of the pool total, showing the non-monotone stretch.

s	$\kappa_1 = \kappa_2$	κ_3	**sum**
10	2.63104	4.73793	10.00000
15	1.78877	11.42245	15.00000
20	2.01153	15.97694	20.00000
25	2.42093	20.15814	25.00000
30	2.78147	24.43705	30.00000
35	4.19309	26.61381	35.00000
40	6.07707	27.84586	40.00000

The paper's qualitative claim reproduces exactly in Table 1.10: κ_1 and κ_2 rise to 2.63 at $s = 10$, **fall** to 1.79 by $s = 15$, and rise thereafter, so the conditional mean allocation is not comonotonic and not Pareto optimal, and the algorithm flattens precisely that stretch.

The published Figure 1.5 cannot be right at its own right hand edge, and it takes no computation to see why. Agent 3's loss is capped at $N = 30$, so at $s = 40$ adding up forces $\kappa_1 + \kappa_2 \geq 10$, and by symmetry $\kappa_1 = \kappa_2 \geq 5$. The published panels for agents 1 and 2 are drawn on an axis whose top tick is 4. The true values are $\kappa_1(35) = 4.19$ and $\kappa_1(40) = 6.08$, which leave the box. Dropping the $N = 30$ truncation of agent 3, and only that, brings $\kappa_1(40)$ down to 3.31 and inside it, which is the one reading of the model consistent with what is printed.

Nothing in the paper's theorems depends on this. The algorithm is right, the code that ships it here is the paper's, and the interesting part of Figure 1.5, the non-monotone stretch the algorithm exists to repair, is exactly where the paper says it is.

1.7. Where the detail is

Each of these has a chapter, and the chapters carry what a page cannot: the transcription decisions, the readings that had to be inferred and then confirmed by fit, the grid caveats, and the arithmetic behind every acceptance test. Venter (1983), Bear and Nemlick (1990) and Homer and Clark (2003) are in the actuarial models part, Bruno et al. (2006) in the part on exact computation, and Denuit, Dhaene, et al. (2025) sits with Denuit, Ortega-Jiménez, et al. (2025) in the part on conditional expectations given the sum.

The book also carries what did not work, which is the more informative half. Jin et al. (2016) has a part to itself and is not a reproduction: the feature it needs, common shock dependence on frequencies rather than on a joint severity, does not exist. Ben Rached et al. (2024) and Wilson and Keich (2016) locate a floor under FFT accuracy, below roughly 10^{-13}, and Wilson and Keich (2017) shows the repair. Three of the fifteen papers contain errors the reproduction found, which is not a criticism of the authors so much as evidence that the check runs in both directions.

Part II.

Actuarial models

2. Venter and the transformed gamma distribution

Reproducing Venter (1983), Exhibit 3

2.1. Context and problem

Venter (1983) introduces the transformed gamma and transformed beta families to actuarial work, and proposes the transformed gamma as a three-moment approximation to the *aggregate* loss distribution. The paper closes with a test of that proposal. Exhibit 3 sets up one compound distribution and computes its cumulative probability and excess ratio three ways over the range $25,000 to $850,000 in $25,000 steps:

1. the **characteristic function** method, working from the continuous severity,
2. the **recursive** method of Adelson and Panjer, working from a severity discretized at $500 intervals,
3. the **transformed gamma** fitted to the first three aggregate moments.

Venter's conclusion is that "the two exact methods are extremely similar, indicating that not much is lost by the discrete approximation to severity", and that the transformed gamma "is reasonably close over a wide range of loss sizes".

This is an unusually good reproduction target for `aggregate`, because the two exact columns are not two different models. They are one model computed at two resolutions, and `aggregate` computes both by changing a single argument. The transformed gamma is reproducible too, since the transformed gamma *is* `scipy.stats.gengamma`.

2.2. Paper solution and gold standard

Exhibit 3 Part 2, Figure 2.1, gives the assumptions in full. Frequency is Poisson with $\lambda = 13.7376$. Severity is a piecewise linear CDF over $0 to $250,000, tabulated at 22 knots, with the remaining mass a point mass at the $250,000 limit. The stated moments are severity mean 18,198 with CV 2.6600 and skewness 3.6746, and aggregate mean 250,000 with CV 0.7667 and skewness 1.0744. The fitted transformed gamma has $r = 0.5613125$, $\alpha = 1.8300318$, $\lambda = 1/417{,}896.414$.

Transcribing a 1983 table from a scanned page needs an acceptance test, and the paper supplies three: if the severity knots are read correctly, the mean, CV and skewness must come out.

EXHIBIT 3
PART 2

AGGREGATE LOSS DISTRIBUTIONS
COMPARATIVE ASSUMPTIONS

Frequency: Poisson λ = 13.7376
Piecewise Linear *CDF*

Limit (000)	Cumulative Probability	Limit (000)	Cumulative Probability
1	.38935	25	.85690
5	.77870	35	.87927
6	.78438	50	.90280
7	.78981	75	.92739
8	.79498	100	.94256
9	.79993	125	.95277
10	.80466	150	.96009
12.5	.81564	175	.96556
15	.82553	200	.96979
17.5	.83449	225	.97316
20	.84264	250	.97590

Discrete *PDF*

Amount	Probability
500	.38326640625
1000	.03041796875
1500 to 4000	.04866875 each 500
4500	.054731628
5000	.019691497
5500 to 249,000 at each N = 500k	Piecewise linear probability from N − 250 to N + 250
249,500	.0000685
250,000	.0241137

Moments

	Mean	Coefficient of Variation	Coefficient of Skewness
Severity	18,198	2.6600	3.6746
Aggregate	250,000	.7667	1.0744

Transformed Gamma Parameters

r : .5613125
α : 1.8300318
λ : 1 ÷ 417896.414

Figure 2.1.: Venter (1983), Exhibit 3, Part 2.

```
%xmode minimal
import numpy as np
import pandas as pd
import scipy.stats as ss
from aggregate.distributions import Aggregate
from bookstyle import GT

KX = np.array([0, 1e3, 5e3, 6e3, 7e3, 8e3, 9e3, 10e3, 12.5e3, 15e3, 17.5e3,
               20e3, 25e3, 35e3, 50e3, 75e3, 100e3, 125e3, 150e3, 175e3,
               200e3, 225e3, 250e3])
```

```
KF = np.array([0, .38935, .77870, .78438, .78981, .79498, .79993, .80466,
               .81564, .82553, .83449, .84264, .85690, .87927, .90280,
               .92739, .94256, .95277, .96009, .96556, .96979, .97316,
               .97590])
LIMIT, LAM = 250e3, 13.7376
ATOM = 1 - KF[-1]                 # the mass Venter puts at the $250,000 limit
seg, a_, b_ = np.diff(KF), KX[:-1], KX[1:]

def raw_moment(k):
    """E[X^k] for a piecewise uniform density plus the atom at the limit."""
    return ((seg * (b_ ** (k + 1) - a_ ** (k + 1))
             / ((k + 1) * (b_ - a_))).sum() + ATOM * LIMIT ** k)

s1, s2, s3 = raw_moment(1), raw_moment(2), raw_moment(3)
sv = s2 - s1 ** 2
GT(pd.DataFrame([
    {'quantity': 'severity mean', 'recomputed': s1, 'Exhibit 3': 18198},
    {'quantity': 'severity CV', 'recomputed': np.sqrt(sv) / s1, 'Exhibit 3': 2.6600},
    {'quantity': 'severity skewness',
     'recomputed': (s3 - 3 * s1 * s2 + 2 * s1 ** 3) / sv ** 1.5, 'Exhibit 3': 3.6746},
    {'quantity': 'aggregate mean', 'recomputed': LAM * s1, 'Exhibit 3': 250000},
    {'quantity': 'aggregate CV', 'recomputed': np.sqrt(LAM * s2) / (LAM * s1),
     'Exhibit 3': 0.7667},
    {'quantity': 'aggregate skewness', 'recomputed': LAM * s3 / (LAM * s2) ** 1.5,
     'Exhibit 3': 1.0744},
]).set_index('quantity'), formatters={'recomputed': 4, 'Exhibit 3': 4})
```

```
Exception reporting mode: Minimal
```

Table 2.1.: The three severity and three aggregate moments recomputed from the transcribed knots, against the printed values.

quantity	recomputed	Exhibit 3
severity mean	18,198.1950	18,198.0000
severity CV	2.6600	2.6600
severity skewness	3.6746	3.6746
aggregate mean	249,999.5236	250,000.0000
aggregate CV	0.7667	0.7667
aggregate skewness	1.0744	1.0744

All six agree in Table 2.1, so the transcription is correct. The severity mean is 18,198.195 against a printed 18,198, and the aggregate mean is 249,999.5 against a printed 250,000. The scanned copy shows the aggregate mean as 250,600, which the arithmetic identifies as a scanning artifact of 250,000.

2.3. The `aggregate` reproduction

The severity is a piecewise uniform density with an atom at the policy limit. `aggregate` builds the density part as a `chistogram`, whose bins are exactly the tabulated knots. The atom is easier to obtain than to declare: carry the limit mass in one wide bin above $250,000 and then apply a `250000 xs 0` layer, which folds everything above the limit onto the limit itself. Table 2.2 reads back the mean, CV and total mass at Venter's own $500 bucket.

```
edges = np.concatenate((KX, [LIMIT * 1.2]))
probs = np.concatenate((seg, [ATOM]))

def venter(bs, log2):
    obj = Aggregate('V', exp_en=LAM, exp_limit=LIMIT, exp_attachment=0,
                    sev_name='chistogram', sev_xs=edges, sev_ps=probs,
                    freq_name='poisson')
    obj.update(log2=log2, bs=bs, normalize=False, padding=1)
    return obj

a500 = venter(500, 16)
GT(pd.DataFrame([{'bucket size': 500, 'mean': a500.est_m, 'CV': a500.est_cv,
                  'total mass': a500.agg_density.sum()}]).set_index('bucket size'),
   formatters={'mean': 2, 'CV': 4, 'total mass': 12})
```

Table 2.2.: The built aggregate at Venter's $500 bucket, with the total mass as a grid check.

bucket size	mean	CV	total mass
500	249,999.52	0.7667	1.000000000000

Venter's third column needs no new machinery either. His transformed gamma, with distribution function $G(x; r, \alpha, \lambda) = \mathrm{IG}((\lambda x)^{\alpha}; r)$, is the generalized gamma: `gengamma(a=r, c=alpha, scale=1/lam)`. Table 2.3 checks that identification against the moments Venter prints.

```
R, ALPHA, LAM_TG = 0.5613125, 1.8300318, 1 / 417896.414
tg = ss.gengamma(R, ALPHA, scale=1 / LAM_TG)
GT(pd.DataFrame([{'quantity': q, 'from Venter parameters': v, 'Exhibit 3': p}
                 for q, v, p in [('mean', tg.mean(), 250000),
                                 ('CV', tg.std() / tg.mean(), 0.7667),
                                 ('skewness', float(tg.stats('s')), 1.0744)]]
                ).set_index('quantity'),
   formatters={'from Venter parameters': 4, 'Exhibit 3': 4})
```

Table 2.3.: The `scipy` generalized gamma at Venter's fitted parameters, against the Exhibit 3 moments.

quantity	from Venter parameters	Exhibit 3
mean	250,000.0010	250,000.0000
CV	0.7667	0.7667
skewness	1.0744	1.0744

Now the exhibit itself, Exhibit 3 Part 1, Figure 2.2. The bucket size is the only thing that changes between the two exact columns, so run it at Venter's $500 and at three finer settings; Table 2.4 puts the first twelve rows beside the published ones.

EXHIBIT 3
PART 1

AGGREGATE LOSS DISTRIBUTIONS
COMPARATIVE SUMMARY

Aggregate Loss ($000)	Characteristic Function Method		Recursive Method		Transformed Gamma	
	Cumulative Probability	Excess Ratio	Cumulative Probability	Excess Ratio	Cumulative Probability	Excess Ratio
25	.0508	.9016	.0516	.9016	.0621	.9031
50	.1291	.8107	.1298	.8107	.1260	.8125
75	.2009	.7273	.2015	.7272	.1895	.7283
100	.2676	.6507	.2683	.6507	.2520	.6503
125	.3289	.5806	.3295	.5806	.3129	.5786
150	.3843	.5163	.3848	.5163	.3717	.5129
175	.4341	.4573	.4346	.4573	.4280	.4529
200	.4788	.4030	.4793	.4029	.4817	.3984
225	.5189	.3529	.5193	.3529	.5324	.3491
250	.5548	.3066	.5552	.3066	.5801	.3047
275	.6034	.2642	.6040	.2642	.6245	.2650
300	.6556	.2273	.6561	.2273	.6658	.2295
325	.7008	.1951	.7013	.1951	.7039	.1981
350	.7405	.1672	.7408	.1672	.7388	.1702
375	.7749	.1431	.7752	.1431	.7707	.1457
400	.8047	.1221	.8049	.1221	.7995	.1243
425	.8303	.1039	.8305	.1039	.8255	.1055
450	.8524	.0880	.8526	.0880	.8488	.0893
475	.8714	.0742	.8716	.0742	.8696	.0752
500	.8878	.0622	.8879	.0622	.8881	.0631
525	.9045	.0518	.9047	.0518	.9043	.0528
550	.9201	.0430	.9203	.0430	.9186	.0439
575	.9332	.0357	.9333	.0357	.9310	.0364
600	.9442	.0296	.9443	.0296	.9418	.0301
625	.9534	.0245	.9535	.0245	.9511	.0247
650	.9611	.0202	.9611	.0202	.9592	.0203
675	.9675	.0167	.9675	.0167	.9660	.0165
700	.9728	.0137	.9729	.0137	.9718	.0134
725	.9773	.0112	.9773	.0112	.9768	.0109
750	.9810	.0091	.9810	.0091	.9809	.0088
775	.9844	.0074	.9844	.0074	.9844	.0070
800	.9873	.0060	.9873	.0060	.9873	.0056
825	.9897	.0048	.9897	.0048	.9897	.0045
850	.9916	.0039	.9916	.0039	.9917	.0035

Figure 2.2.: Venter (1983), Exhibit 3, Part 1.

```
XS = np.arange(25_000, 875_000, 25_000)
CHAR = [.0508, .1291, .2009, .2676, .3289, .3843, .4341, .4788, .5189, .5548,
        .6034, .6556, .7008, .7405, .7749, .8047, .8303, .8524, .8714, .8878,
        .9045, .9201, .9332, .9442, .9534, .9611, .9675, .9728, .9773, .9810,
        .9844, .9873, .9897, .9916]
RECU = [.0516, .1298, .2015, .2683, .3295, .3848, .4346, .4793, .5193, .5552,
        .6040, .6561, .7013, .7408, .7752, .8049, .8305, .8526, .8716, .8879,
        .9047, .9203, .9333, .9443, .9535, .9611, .9675, .9729, .9773, .9810,
        .9844, .9873, .9897, .9916]
XR = [.9016, .8107, .7273, .6507, .5806, .5163, .4573, .4030, .3529, .3066,
      .2642, .2273, .1951, .1672, .1431, .1221, .1039, .0880, .0742, .0622,
      .0518, .0430, .0357, .0296, .0245, .0202, .0167, .0137, .0112, .0091,
      .0074, .0060, .0048, .0039]
TG = [.0621, .1260, .1895, .2520, .3129, .3717, .4280, .4817, .5324, .5801,
```

```
      .6245, .6658, .7039, .7388, .7707, .7995, .8255, .8488, .8696, .8881,
      .9043, .9186, .9310, .9418, .9511, .9592, .9660, .9718, .9768, .9809,
      .9844, .9873, .9897, .9917]

out = pd.DataFrame({'aggregate loss': XS, 'Venter char fn': CHAR,
                    'Venter recursive': RECU})
for bs, log2 in [(500, 16), (100, 18), (20, 20), (5, 22)]:
    o = venter(bs, log2)
    out[f'agg bs={bs}'] = [o.density_df.loc[x, 'F'] for x in XS]
    if bs == 500:
        out['agg excess ratio'] = [1 - o.density_df.loc[x, 'lev'] / o.est_m
                                   for x in XS]
out['Venter excess ratio'] = XR
out['Venter transf gamma'] = TG
out['gengamma cdf'] = [tg.cdf(x) for x in XS]
GT(out.set_index('aggregate loss').head(12), float_format=4)
```

Table 2.4.: The first twelve rows of Exhibit 3, Part 1, at four bucket sizes, with the excess ratio and transformed gamma columns.

aggregate loss	Venter char fn	Venter recursive	agg bs=500	agg excess ratio	agg bs=100	agg bs=20	agg bs=5	Venter excess ratio	Venter transf gamma	gengamma cdf
25,000	0.0508	0.0516	0.0516	0.9016	0.0509	0.0508	0.0508	0.9016	0.0621	0.0621
50,000	0.1291	0.1298	0.1298	0.8107	0.1293	0.1291	0.1291	0.8107	0.1260	0.1260
75,000	0.2009	0.2015	0.2015	0.7273	0.2010	0.2009	0.2009	0.7273	0.1895	0.1895
100,000	0.2676	0.2683	0.2683	0.6507	0.2678	0.2677	0.2677	0.6507	0.2520	0.2520
125,000	0.3289	0.3295	0.3295	0.5806	0.3290	0.3289	0.3289	0.5806	0.3129	0.3129
150,000	0.3843	0.3848	0.3848	0.5163	0.3844	0.3843	0.3843	0.5163	0.3717	0.3717
175,000	0.4341	0.4346	0.4346	0.4573	0.4342	0.4341	0.4341	0.4573	0.4280	0.4280
200,000	0.4788	0.4793	0.4792	0.4030	0.4789	0.4788	0.4788	0.4030	0.4817	0.4817
225,000	0.5189	0.5193	0.5193	0.3529	0.5190	0.5189	0.5189	0.3529	0.5324	0.5324
250,000	0.5548	0.5552	0.5552	0.3066	0.5549	0.5549	0.5548	0.3066	0.5801	0.5801
275,000	0.6034	0.6040	0.6040	0.2642	0.6035	0.6034	0.6034	0.2642	0.6245	0.6245
300,000	0.6556	0.6561	0.6561	0.2273	0.6557	0.6556	0.6556	0.2273	0.6658	0.6658

2.4. Comments

2.4.1. The bucket size selects which of Venter's exact columns you get

```
GT(pd.DataFrame([
    {'bucket size': bs,
     'max abs diff vs characteristic function':
         float(np.abs(out[f'agg bs={bs}'] - out['Venter char fn']).max()),
     'max abs diff vs recursive':
         float(np.abs(out[f'agg bs={bs}'] - out['Venter recursive']).max())}
    for bs in [500, 100, 20, 5]]).set_index('bucket size'),
   float_format=5)
```

Table 2.5.: Largest absolute difference from each of Venter's two exact columns, as the bucket is refined.

bucket size	**max abs diff vs characteristic function**	**max abs diff vs recursive**
500	0.00082	0.00005
100	0.00019	0.00067
20	0.00009	0.00081
5	0.00007	0.00084

Table 2.5 is the whole argument of this chapter. At Venter's \$500 discretization, `aggregate` agrees with his **recursive** column to 5×10^{-5} over all 34 rows, which is exact at the four decimal places printed, and differs from his **characteristic function** column by up to 8.2×10^{-4}. Refine the bucket and the two allegiances swap: by `bs = 5` the agreement with the characteristic function column is 7×10^{-5} and the disagreement with the recursive column has grown to 8.4×10^{-4}.

Venter presents two *methods* but we can reproduce both using FFTs at two resolutions. `aggregate`'s FFT with a \$500 bucket agrees with the Adelson and Panjer recursion, and FFT with a \$5 bucket agrees with the characteristic function inversion. The 0.0008 that Venter reports as the gap between his two exact methods is not a difference between characteristic functions and recursion at all. It is the discretization error of a \$500 bucket, and it is the same error `aggregate` incurs when asked for the same bucket.

2.4.2. The excess ratio and the transformed gamma

```
GT(pd.DataFrame([
    {'comparison': 'agg excess ratio at bs=500 vs Venter',
     'max abs diff': float(np.abs(out['agg excess ratio']
                                  - out['Venter excess ratio']).max())},
    {'comparison': 'scipy gengamma vs Venter transformed gamma column',
     'max abs diff': float(np.abs(out['gengamma cdf']
                                  - out['Venter transf gamma']).max())},
]).set_index('comparison'), float_format=6)
```

Table 2.6.: The excess ratio and transformed gamma columns, each against its published counterpart.

comparison	**max abs diff**
agg excess ratio at bs=500 vs Venter	0.000052
scipy gengamma vs Venter transformed gamma column	0.000049

Table 2.6 puts both at 5×10^{-5}, so both columns reproduce exactly at the printed precision. The excess ratio comes straight off `density_df['lev']`, since $R(a) = 1 - \mathsf{E}[L \wedge a]/\mathsf{E}[L]$.

The transformed gamma check is worth doing for its own sake. Venter's Appendix A solves two nonlinear equations in Γ functions by Newton iteration in APL, then Exhibit 1 evaluates the incomplete gamma by a 30 to 200 term series expansion. `scipy.stats.gengamma(0.5613125, 1.8300318,`

`scale=417896.414)` returns the same 34 values. The identification is exact rather than approximate: the transformed gamma with parameters (r, α, λ) is `gengamma` with shape $a = r$, shape $c = \alpha$ and scale $1/\lambda$.

2.4.3. On the approximation Venter was testing

Venter's own verdict, that the transformed gamma is "reasonably close over a wide range of loss sizes", reads generously with the exact answer beside it.

```
err = 100 * (np.array(TG) - np.array(CHAR))
inside = [x for i, x in enumerate(XS) if np.all(np.abs(err[i:]) < 0.1)]
GT(pd.DataFrame([
    {'statistic': 'largest overstatement, points of probability',
     'value': err.max(), 'at': XS[err.argmax()]},
    {'statistic': 'largest understatement, points of probability',
     'value': err.min(), 'at': XS[err.argmin()]},
    {'statistic': 'first loss size beyond which the error stays under 0.1 points',
     'value': np.nan, 'at': inside[0]},
]).set_index('statistic'), formatters={'value': 2, 'at': '{x:,.0f}'})
```

Table 2.7.: How far the three-moment transformed gamma strays from the exact distribution, in points of probability.

statistic	value	at
largest overstatement, points of probability	2.53	250,000
largest understatement, points of probability	-1.60	125,000
first loss size beyond which the error stays under 0.1 points	—	725,000

By Table 2.7 the three-moment fit crosses the exact distribution four times over the range, running 1.1 points of probability high at $25,000, 1.6 points low at $125,000, and 2.5 points high at $250,000, its worst single row. It does not settle inside a tenth of a point until $725,000. The excess ratio column holds up better where it matters commercially, within 2% of the ratio through $650,000, before degrading to 10% at $850,000 where the ratio itself is down to 0.0039.

That is a fair result for a three-parameter fit and a poor substitute for the exact calculation, which is precisely the trade Venter names in his summary: the exact methods "require more input information ... and they also require substantially more computation", and he predicts they will become "increasingly viable" as computing gets cheaper. The four builds in this document take a few seconds each.

2.4.4. What did not transcribe

Figure 2.1 also prints Venter's $500 discrete PDF, and I could not decode its first five rows. The rule stated for the body of that column, that the probability at N is the piecewise linear probability from $N - 250$ to $N + 250$, reproduces the tabulated 0.04866875 for the $1,500 to $4,000 block exactly, but does not give the printed values at $500, $1,000, $4,500 or $5,000, and the printed $500 entry is more than twice what any nearest-bucket rule allows. Nothing here depends on it: `aggregate` does its own discretization from the piecewise linear CDF, and the CDF is verified independently by the three severity moments.

3. Mack and Fackler's test of Riebesell's rule

Reproducing Mack and Fackler (2003)

3.1. Context and problem

German liability insurers price increased limits by a rule Paul Riebesell published in 1936: doubling the sum insured multiplies the net premium by a constant factor $1 + z$, typically $z = 20\%$. Written out,

$$b(v) = b_0(1+z)^{\mathrm{ld}(v/v_0)} = b_0\left(\frac{v}{v_0}\right)^{\mathrm{ld}(1+z)},$$

a pure power law in the sum insured, and so invariant to inflation and to currency. That invariance is why it survives: unlike a property exposure curve it never needs re-basing.

Mack and Fackler (2003) asks whether the rule is consistent with the collective risk model, where $b(v) = \mathsf{E}[N]\,\mathsf{E}[X \wedge v]$. Differentiating gives $S(v) = b'(v)/\mathsf{E}[N]$, and a power law premium forces a power law survival function, that is a Pareto with

$$\alpha = 1 - \mathrm{ld}(1+z) \in (0,1).$$

An α below one means an infinite mean, so the implied claim size law cannot be Pareto everywhere. The paper's contribution is to show it does not have to be. There is a threshold u such that any realistic claim size law, fitted with a Pareto tail above u, satisfies Riebesell's rule exactly for all sums insured above u, and the threshold turns out to sit below the range of sums insured actually written.

3.2. Paper solution and gold standard

The condition on the threshold is that

$$\alpha_F(u) := 1 - \frac{u\,S(u)}{\int_0^u S(x)\,\mathrm{d}x} = 1 - \frac{u\,S(u)}{\mathsf{E}[X \wedge u]}$$

equals α. The quantity $\alpha_F(u)$ runs from 0 at $u = 0$ to 1 as $u \to \infty$ for any continuous F with a finite mean, so a solution always exists. Table 1, Figure 3.1, tabulates it for four severities at seven multiples of the mean, and it is the paper's one block of numbers.

The formulation is convenient for `aggregate`: α_F is built from a limited expected value and a survival function, both of which come straight out of a build.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build

US = [0.01, 0.1, 1, 3, 10, 30, 100]
CV18 = float(np.sqrt(np.exp(1.8 ** 2) - 1))
SEVS = {                                  # all normalized to E(X) = 1
    'Exponential': 'sev 1 * expon',
    'Gamma a=0.1': 'sev 10 * gamma 0.1',
    'Lognormal s=1.8': f'sev lognorm 1 cv {CV18}',
    'American Pareto a=1.5': 'sev 0.5 * lomax 1.5',
}
PAPER = {
    'Exponential': [0.005, 0.049, 0.418, 0.842, 0.9996, None, None],
    'Gamma a=0.1': [0.092, 0.151, 0.294, 0.442, 0.709, 0.952, 0.9999],
    'Lognormal s=1.8': [0.028, 0.185, 0.5, 0.663, 0.815, 0.914, 0.972],
    'American Pareto a=1.5': [0.0147, 0.127, 0.545, 0.740, 0.867, 0.928, 0.962],
}
```

```
Exception reporting mode: Minimal
```

Two readings had to be settled first. **"American Pareto"** is the two parameter Pareto $S(x) = (\theta/(\theta + x))^{\alpha}$, `lomax` in DecL, as against the European single parameter form; with $\alpha = 1.5$ the scale $\theta = 0.5$ gives mean 1. **"Gamma $\alpha = 0.1$"** is the shape, so scale 10 for a unit mean. Both are confirmed by Table 3.1 reproducing.

3.3. The `aggregate` reproduction

Table 1. Values of $\alpha_F(u)$ for some distributions F.

u/E(X)	Exponential	Gamma α = 0.1	Lognormal σ = 1.8	American Pareto α = 1.5
0.01	0.005	0.092	0.028	0.0147
0.1	0.049	0.151	0.185	0.127
1	0.418	0.294	0.5	0.545
3	0.842	0.442	0.663	0.740
10	0.9996	0.709	0.815	0.867
30		0.952	0.914	0.928
100		0.9999	0.972	0.962

Figure 3.1.: Mack and Fackler (2003), Table 1.

```
def lev(prog, u, log2=20):
    """E[X and u]: one fixed claim in the layer u xs 0."""
    return build(f'agg S 1 claim {u} xs 0 {prog} fixed',
                 log2=log2, bs=u / 2 ** (log2 - 2), normalize=False).est_m

def sf(prog, u, log2=20):
    """S(u) read off an unlimited build on a grid reaching well past u."""
    obj = build(f'agg S 1 claim {prog} fixed', log2=log2,
                bs=4 * u / 2 ** log2, normalize=False)
```

```
    return float(obj.density_df.loc[u, 'S'])

rows = []
for name, prog in SEVS.items():
    for i, u in enumerate(US):
        L, S = lev(prog, u), sf(prog, u)
        rows.append({'F': name, 'u / E(X)': u, 'agg E[X and u]': L,
                     'agg S(u)': S, 'alpha_F(u)': 1 - u * S / L,
                     'Table 1': PAPER[name][i]})
t1 = pd.DataFrame(rows)
t1['abs diff'] = (t1['alpha_F(u)'] - t1['Table 1']).abs()
GT(t1.set_index(['F', 'u / E(X)']),
   formatters={'agg E[X and u]': 6, 'agg S(u)': 6, 'alpha_F(u)': 6,
               'Table 1': 4, 'abs diff': '{x:.2e}'})
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
S: aggregate PMF deficit 9.608e-01 (Σp = 1 − 9.608e-01 < 1); forwards and backwards S diverge by
the deficit (forwards > backwards). Further occurrences this session are suppressed; each object
records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

Table 3.1.: $\alpha_F(u)$ for the four severities at seven multiples of the mean, against Table 1.

F	*u / E(X)*	agg E[X and u]	agg S(u)	alpha_F(u)	Table 1	abs diff
Exponential	0.01	0.009950	0.990050	0.004992	0.0050	8.31e-06
	0.10	0.095163	0.904837	0.049167	0.0490	1.67e-04
	1.00	0.632121	0.367879	0.418024	0.4180	2.44e-05
	3.00	0.950213	0.049787	0.842814	0.8420	8.14e-04
	10.00	0.999955	0.000045	0.999546	0.9996	5.40e-05
	30.00	1.000000	0.000000	1.000000	—	—
	100.00	1.000000	0.000000	1.000000	—	—
Gamma a=0.1	0.01	0.005211	0.473231	0.091859	0.0920	1.41e-04
	0.10	0.039736	0.337379	0.150943	0.1510	5.68e-05
	1.00	0.244508	0.172448	0.294714	0.2940	7.14e-04
	3.00	0.492912	0.091642	0.442242	0.4420	2.42e-04
	10.00	0.830454	0.024127	0.709469	0.7090	4.69e-04
	30.00	0.986982	0.001565	0.952423	0.9520	4.23e-04
	100.00	0.999991	0.000001	0.999945	0.9999	4.45e-05
Lognormal s=1.8	0.01	0.009786	0.951384	0.027762	0.0280	2.38e-04
	0.10	0.079431	0.647735	0.184536	0.1850	4.64e-04
	1.00	0.368120	0.184060	0.500001	0.5000	7.66e-07
	3.00	0.582473	0.065478	0.662758	0.6630	2.42e-04
	10.00	0.794314	0.014658	0.815465	0.8150	4.65e-04
	30.00	0.917975	0.002639	0.913755	0.9140	2.45e-04
	100.00	0.978551	0.000272	0.972238	0.9720	2.38e-04
American Pareto a=1.5	0.01	0.009852	0.970733	0.014730	0.0147	3.02e-05
	0.10	0.087129	0.760725	0.126898	0.1270	1.02e-04
	1.00	0.422650	0.192450	0.544659	0.5450	3.41e-04
	3.00	0.622036	0.053995	0.739590	0.7400	4.10e-04
	10.00	0.781782	0.010391	0.867082	0.8670	8.19e-05
	30.00	0.871963	0.002099	0.927785	0.9280	2.15e-04
	100.00	0.929465	0.000351	0.962245	0.9620	2.45e-04

```
GT(pd.DataFrame([{'cells compared': int(t1['abs diff'].notna().sum()),
                  'largest absolute difference': float(t1['abs diff'].max())}]),
   formatters={'largest absolute difference': '{x:.2e}'})
```

Table 3.2.: Worst cell error over the whole of Table 1.

cells compared	largest absolute difference
26	8.14e-04

By Table 3.2 all 26 cells land within 8×10^{-4}, inside the paper's three decimal rounding. The two blank cells are the exponential at $u = 30$ and $u = 100$, where α_F is 1 to more places than the table prints and Mack and Fackler leave the entries empty.

Three exact statements in the text follow directly, in Table 3.3.

```
Z = 0.20
ALPHA = 1 - np.log(1 + Z) / np.log(2)
GT(pd.DataFrame([
    {'quantity': 'alpha = 1 - ld(1+z), z = 20%', 'recomputed': ALPHA,
     'paper': 0.737},
    {'quantity': 'exposure exponent 1 - alpha', 'recomputed': 1 - ALPHA,
     'paper': 0.263},
    {'quantity': 'mixed tail index alpha + beta, beta = 1.3',
     'recomputed': ALPHA + 1.3, 'paper': 2.037},
]).set_index('quantity'), float_format=6)
```

Table 3.3.: The three constants the text quotes for $z = 20$, recomputed.

quantity	recomputed	paper
alpha = 1 - ld(1+z), z = 20%	0.736966	0.737000
exposure exponent 1 - alpha	0.263034	0.263000
mixed tail index alpha + beta, beta = 1.3	2.036966	2.037000

The last row of Table 3.3 is the paper's resolution of an apparent contradiction. Riebesell forces $\alpha < 1$ on the claim size law of a *single* risk, whereas observed portfolio tails in liability run near 2. Section 8 shows the two are compatible: mixing the individual laws over a Pareto distribution of sums insured with index β gives a portfolio tail of index $\alpha + \beta$, and with $\beta = 1.3$ that is 2.037.

3.4. Theorem 2, end to end

The theorem says: take any realistic F, find the u solving $\alpha_F(u) = \alpha$, replace the tail above u with a Pareto, and Riebesell's rule holds exactly from u upward. The worked instance in the text is the lognormal with $\sigma = 1.8$, for which the paper reads off Table 1 that "we replace the lognormal above $u \approx 5 \cdot \mathsf{E}(X)$".

Scale to the paper's own liability context, $\mathsf{E}(X)$ below €50,000 and sums insured from about €500,000; Table 3.4 solves for the threshold.

```
import scipy.stats as ss
from scipy.integrate import quad
from scipy.optimize import brentq

SIGMA, EX = 1.8, 50_000.0
LN_SCALE = EX * np.exp(-SIGMA ** 2 / 2)
FZ = ss.lognorm(SIGMA, scale=LN_SCALE)
U = brentq(lambda u: 1 - u * FZ.sf(u) / quad(FZ.sf, 0, u, limit=300)[0] - ALPHA,
           EX, 20 * EX, xtol=1e-6)
SU = float(FZ.sf(U))

GT(pd.DataFrame([{'E(X)': EX, 'threshold u*': U, 'u* / E(X)': U / EX,
                  'paper': 'u ~ 5 E(X)', 'S(u*)': SU}]),
   formatters={'E(X)': '{x:,.0f}', 'threshold u*': '{x:,.1f}',
               'u* / E(X)': 4, 'S(u*)': 8})
```

Table 3.4.: The threshold u^* solving $\alpha_F(u) = \alpha$ for the lognormal with $\sigma = 1.8$, against the paper's reading of Table 1.

E(X)	threshold u*	u* / E(X)	paper	S(u*)
50,000	259,286.7	5.1857	u ~ 5 E(X)	0.03480845

The splice is a DecL clause, so Theorem 2's construction can be written rather than tabulated. Two components, each conditioned to its own interval: the lognormal below u^* and a single parameter Pareto with the Riebesell index above it, weighted $1 - S(u^*)$ and $S(u^*)$ so the survival function is continuous at the join.

```
SPLICE = (f'sev [{LN_SCALE:.10g} {U:.10g}] * [lognorm pareto] '
          f'[{SIGMA:g} {ALPHA:.10g}] '
          f'wts [{1 - SU:.12g} {SU:.12g}] '
          f'splice [0 {U:.10g}] [{U:.10g} inf]')
print(SPLICE)
```

```
sev [9894.934954 259286.6924] * [lognorm pareto] [1.8 0.7369655942] wts [0.965191548087
0.034808451913] splice [0 259286.6924] [259286.6924 inf]
```

Every limited expected value in Table 3.5 is read off **one** grid, which matters: a limited expected value computed on a grid tuned to its own limit is not comparable with one computed on a different grid when the severity has structure far below the limit, and this severity has all its density three orders of magnitude below the largest sum insured.

```
LOG2, BS = 22, 5.0                      # x_max = 21.0M, covers every a below

def lev_spliced(a):
    return build(f'agg Riebesell 1 claim {a:.10g} xs 0 {SPLICE} fixed',
                 log2=LOG2, bs=BS, normalize=False).est_m

v = 5_000_000.0
lev_v = lev_spliced(v)
rows = []
for a in [50_000., 100_000., 250_000., U, 500_000., 1e6, 2e6, 5e6]:
    ra = lev_spliced(a) / lev_v
    rows.append({'a': a, 'a >= u*': a >= U - 1, 'agg E[X and a]': lev_spliced(a),
                 'agg r(a)': ra, 'Riebesell (a/v)^0.263': (a / v) ** (1 - ALPHA),
                 'ratio': ra / (a / v) ** (1 - ALPHA)})
```

```
GT(pd.DataFrame(rows).set_index('a'),
   formatters={'agg E[X and a]': '{x:,.2f}', 'agg r(a)': 9,
               'Riebesell (a/v)^0.263': 9, 'ratio': 9})
```

Table 3.5.: The loss elimination ratio off the spliced severity against Riebesell's $(a/v)^{1-\alpha}$, above and below the threshold.

a	a >= u*	agg E[X and a]	agg r(a)	Riebesell (a/v)^0.263	ratio
50.000k	False	18,406.01	0.246296481	0.297804454	0.827040960
100.000k	False	25,103.44	0.335916800	0.357365344	0.939981466
250.000k	False	33,981.96	0.454723048	0.454762189	0.999913931
259.287k	True	34,312.44	0.459145281	0.459146069	0.999998283
500.000k	True	40,781.87	0.545714627	0.545714627	1.000000000
1.000M	True	48,938.24	0.654857552	0.654857552	1.000000000
2.000M	True	58,725.89	0.785829062	0.785829062	1.000000000
5.000M	True	74,731.12	1.000000000	1.000000000	1.000000000

In Table 3.5, above the threshold the loss elimination ratio matches Riebesell's $(a/v)^{1-\alpha}$ to nine significant figures. Below it the agreement fails, and fails hard: at $a = \mathsf{E}(X)$ the true ratio is 17% under the Riebesell value. That is not a numerical artifact, it is Theorem 2 being tight. The rule is not asymptotic or approximate above u, and it is simply wrong below it.

The premium form of the same statement, Table 3.6, is sharper still, because the rule was stated as a premium rule in the first place.

```
GT(pd.DataFrame([{'sum insured v': v0, 'v >= u*': v0 >= U,
                  'b(2v)/b(v) = E[X and 2v] / E[X and v]':
                      lev_spliced(2 * v0) / lev_spliced(v0),
                  "Riebesell's 1 + z": 1.20}
                 for v0 in [300_000., 600_000., 1_200_000., 100_000.]]
                ).set_index('sum insured v'),
   formatters={'b(2v)/b(v) = E[X and 2v] / E[X and v]': 6,
               "Riebesell's 1 + z": 2})
```

Table 3.6.: The premium ratio on doubling the sum insured, at three sums insured above the threshold and one below.

sum insured v	v >= u*	b(2v)/b(v) = E[X and 2v] / E[X and v]	Riebesell's 1 + z
300,000	True	1.200000	1.20
600,000	True	1.200000	1.20
1,200,000	True	1.200000	1.20
100,000	False	1.270985	1.20

In Table 3.6, doubling the sum insured multiplies the collective risk premium by exactly 1.200000 at every sum insured above the threshold, and by 1.271 below it. A 1936 rating rule turns out to be an exact consequence of the collective risk model on the range where it is used, and only there.

3.5. Comments

3.5.1. What `aggregate` is doing here, and what it is not

This is a severity paper. There is no aggregate distribution anywhere in it: $\mathsf{E}[N]$ enters every formula and cancels out of every one. `aggregate` is being used as a limited expected value engine on a spliced severity, which it does well, but the compound machinery is idle.

The reproduction is still worth having for two reasons. Mack and Fackler's threshold condition is stated entirely in terms of $\mathsf{E}[X \wedge u]$ and $S(u)$, which are the two things exposure rating is built from, so the whole paper is one short expression away from quantities `aggregate` already reports. And Theorem 2's construction is a `splice` clause: two components, two conditioning windows and two weights, which is the same shape as the theorem's own statement. `Mack2003.RiebesellPareto` in the library carries the Riebesell index on its own, ground up rather than spliced, and has no finite mean, so it can only be used the way the paper uses it, inside a limit.

3.5.2. A grid caveat worth carrying forward

The first attempt at the exposure curve gave a clean power law with the wrong constant, off by 2.6%. The cause was giving each limit its own bucket size, `bs = a / 2**17`. That is normally sensible, but here the severity has essentially all its density below €300,000 while the largest limit is €5,000,000, so a grid tuned to the top limit puts the entire lognormal body in its first few buckets. Limited expected values that are going to be **compared with each other** have to come off a common grid. The version above uses one grid for all eight limits and the artifact disappears.

3.5.3. The far tail does not matter, and the paper says so

The Pareto tail with $\alpha = 0.737$ has an infinite mean, so it has no unconditional moments to report. The `splice` clause carries it anyway, with no cutoff and no residual swept onto a last bin, because every limited expected value on this page is taken inside a limit, which is the only way the paper ever uses the curve. The paper says as much: "we can replace F_u above some very high amount $w \gg u$ with something else, e.g. in order to ascertain a finite expectation", because the behaviour of the law above the range where Riebesell is applied does not enter equation (2). Useful licence, and not needed here.

The construction is exact rather than approximate, which is why nine significant figures are available to agree to. The weights $1 - S(u^*)$ and $S(u^*)$ against the two conditioning windows reproduce S continuously at the join by definition, so the spliced survival function *is* the theorem's F_u, not a discretization of it.

One diagnostic follows from the missing moments and is worth naming so it is not read as an error. At some limits the Pareto component is capped to a point mass, its variance is zero to rounding and so comes out very slightly negative, and `aggregate` reports that along with a `nan` skewness. It reaches the moment diagnostics only. Limited expected values, which are all this page reads, are unaffected.

4. Mata and Verheyen on trend by layer

Reproducing Mata and Verheyen (2005)

4.1. Context and problem

Two practical problems spoil excess of loss experience rating. Reinsurers often receive individual claim detail only above a reporting threshold, so lower layers cannot be trended claim by claim; and when the ceding company's limits profile shifts during the experience period, averaging loss costs across years compares things that are not comparable.

Mata and Verheyen (2005) fixes both with the mathematics of exposure rating. A trend factor **for a layer** can be derived from the ground up trend, the severity curve and the limits profile, without touching individual claims, by asking what the expected severity in the layer costs at two claim cost levels:

$$(\text{Excess Trend})_p = \frac{\mathsf{E}_Y[Y \wedge T] - \mathsf{E}_Y[Y \wedge B]}{\mathsf{E}_X[X \wedge T] - \mathsf{E}_X[X \wedge B]},$$

with T and B the top and bottom of the layer capped at the policy limit. The same apparatus, run against two limits profiles instead of two cost levels, gives an exposure adjustment by layer. Both are then split into frequency and severity components, since the count of claims reaching a layer moves with ground up severity trend even when ground up frequency is flat.

Everything reduces to limited expected values, which makes it a clean `aggregate` exercise.

4.2. Paper solution and gold standard

The worked example runs through Tables 1 to 14, beginning with Figure 4.1. A ceding company writes limits from \$250,000 to \$5,000,000, severity is lognormal with $\mu = 9.31$ and $\sigma = 2.29$ in 2005 values, ground up severity trend is 8% a year, the expected gross loss ratio is 60%, and rates rose a cumulative 50% between 2000 and 2005. Five layers are priced. The 2000 severity is the same lognormal with $\mu' = \mu - \ln(1.08^5) = 8.93$.

The headline results are that the \$4MM xs \$1MM layer trends at 9.59% a year against an unlimited 8%, that the same layer's exposure has doubled while total limits exposure rose only 9%, and that most of the excess trend is additional claims reaching the layer rather than larger claims within it.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build

MU, SIGMA, TREND, YEARS, ELR, OL = 9.31, 2.29, 1.08, 5, 0.60, 1.50
CV = float(np.sqrt(np.exp(SIGMA ** 2) - 1))
M05 = float(np.exp(MU + SIGMA ** 2 / 2))
M00 = M05 / TREND ** YEARS
PL = [250e3, 500e3, 750e3, 1e6, 5e6]
SP05 = [2_250_000, 5_400_000, 2_925_000, 6_300_000, 9_000_000]
SP00 = [2_250_000, 4_500_000, 2_925_000, 3_150_000, 3_000_000]
LAYERS = [(250e3, 0.), (250e3, 250e3), (500e3, 500e3), (4e6, 1e6), (5e6, 0.)]
LN = ['$250k xs 0', '$250k xs $250k', '$500k xs $500k', '$4MM xs $1MM',
      '$5MM xs 0']
LOG2, BS = 20, 25.0                 # one grid, x_max = 26.2M
```

```
Exception reporting mode: Minimal
```

The 2000 mean uses the unrounded $\mu' = 8.925195$. Using the printed 8.93 instead moves the limited expected values by about half a percent, more than the effects the paper is measuring, and the paper's own Table 1 is consistent with the unrounded value.

4.3. The `aggregate` reproduction

Table 1: Limits Profile and Limited Expected Values

Underwriting Year		2005	2000
Policy Limit (PL)	Premium (SP)	$E_Y[Y \wedge PL]$	$E_X[X \wedge PL]$
250,000	2,250,000	48,539	38,900
500,000	5,400,000	64,416	50,191
750,000	2,925,000	74,252	56,947
1,000,000	6,300,000	81,301	61,681
5,000,000	9,000,000	117,221	84,401
Total	25,875,000		

Figure 4.1.: Mata and Verheyen (2005), Table 1.

```
def lev(a, mean):
    if a == 0:
        return 0.0
    return build(f'agg M 1 claim {a} xs 0 sev lognorm {mean} cv {CV} fixed',
                 log2=LOG2, bs=BS, normalize=False).est_m

PTS = sorted({0.} | set(PL) | {min(p, t + l) for p in PL for l, t in LAYERS}
             | {min(p, t) for p in PL for l, t in LAYERS})
AGG = {(a, y): lev(a, M05 if y == 2005 else M00)
       for y in (2005, 2000) for a in PTS}
PAP = {(p, 2005): v for p, v in zip(PL, [48539, 64416, 74252, 81301, 117221])}
PAP.update({(p, 2000): v for p, v in zip(PL, [38900, 50191, 56947, 61681, 84401])})
PAP.update({(0., 2005): 0.0, (0., 2000): 0.0})
```

```
t1 = pd.DataFrame({'policy limit': PL, 'premium 2005': SP05,
                   'agg EY[Y and PL]': [AGG[(p, 2005)] for p in PL],
                   'Table 1 2005': [PAP[(p, 2005)] for p in PL],
                   'agg EX[X and PL]': [AGG[(p, 2000)] for p in PL],
                   'Table 1 2000': [PAP[(p, 2000)] for p in PL]})
t1['2005 paper / agg'] = t1['Table 1 2005'] / t1['agg EY[Y and PL]']
t1['2000 paper / agg'] = t1['Table 1 2000'] / t1['agg EX[X and PL]']
GT(t1.set_index('policy limit'),
   formatters={'premium 2005': '{x:,.0f}', 'agg EY[Y and PL]': '{x:,.2f}',
               'Table 1 2005': '{x:,.0f}', 'agg EX[X and PL]': '{x:,.2f}',
               'Table 1 2000': '{x:,.0f}', '2005 paper / agg': 6,
               '2000 paper / agg': 6})
```

Table 4.1.: Limited expected values at the five policy limits in both years, against Table 1.

policy limit	premium 2005	agg EY[Y and PL]	Table 1 2005	agg EX[X and PL]	Table 1 2000	2005 paper / agg	2000 paper / agg
250,000	2,250,000	48,519.84	48,539	38,895.61	38,900	1.000395	1.000113
500,000	5,400,000	64,424.73	64,416	50,214.86	50,191	0.999864	0.999525
750,000	2,925,000	74,289.42	74,252	56,996.34	56,947	0.999496	0.999134
1,000,000	6,300,000	81,363.94	81,301	61,752.27	61,681	0.999226	0.998846
5,000,000	9,000,000	117,503.76	117,221	84,637.21	84,401	0.997594	0.997209

In Table 4.1 the limited expected values do not agree. The paper is 0.04% high at $250,000 and drifts steadily to 0.24% low at $5,000,000, in both years. The discrepancy is small but it is systematic, and it is worth settling before reading anything downstream.

`aggregate` is not the source. The lognormal limited expected value has a closed form, $\mathsf{E}[X \wedge a] = e^{\mu+\sigma^2/2}\,\Phi\left(\frac{\ln a-\mu-\sigma^2}{\sigma}\right) + a\,S(a)$, and Table 4.2 arbitrates:

```
import scipy.stats as ss
FZ05 = ss.lognorm(SIGMA, scale=np.exp(MU))

def closed(a):
    return (M05 * ss.norm.cdf((np.log(a) - MU - SIGMA ** 2) / SIGMA)
            + a * FZ05.sf(a))

chk = pd.DataFrame({'a': PL, 'aggregate': [AGG[(p, 2005)] for p in PL],
                    'closed form': [closed(p) for p in PL],
                    'Table 1': [PAP[(p, 2005)] for p in PL]})
chk['agg / closed form - 1'] = chk['aggregate'] / chk['closed form'] - 1
chk['Table 1 / closed form - 1'] = chk['Table 1'] / chk['closed form'] - 1
GT(chk.set_index('a'),
   formatters={'aggregate': '{x:,.4f}', 'closed form': '{x:,.4f}',
               'Table 1': '{x:,.0f}', 'agg / closed form - 1': '{x:.2e}',
               'Table 1 / closed form - 1': '{x:.2e}'})
```

Table 4.2.: The build and Table 1 each measured against the closed form lognormal limited expected value.

a	aggregate	closed form	Table 1	agg / closed form - 1	Table 1 / closed form - 1
250,000	48,519.8355	48,519.8377	48,539	-4.54e-08	3.95e-04
500,000	64,424.7341	64,424.7363	64,416	-3.42e-08	-1.36e-04
750,000	74,289.4170	74,289.4192	74,252	-2.97e-08	-5.04e-04
1,000,000	81,363.9371	81,363.9393	81,301	-2.71e-08	-7.74e-04
5,000,000	117,503.7600	117,503.7622	117,221	-1.88e-08	-2.41e-03

In Table 4.2, `aggregate` reproduces the closed form to better than five parts in a hundred million at every limit. Table 1 is out by up to twenty-four parts in ten thousand.

4.4. The decisive test

The question that matters is whether the *method* reproduces, so run the entire pipeline twice: once on `aggregate`'s limited expected values, once on the paper's own Table 1 figures. The five policy limits happen to include every layer boundary the example needs, so the second run is possible with nothing but what the paper prints. The six published targets are Figure 4.2, Figure 4.3 and Figure 4.4 for the trend split, and Figure 4.5, Figure 4.7 and Figure 4.8 for the exposure split; Figure 4.6 carries the limits profiles both runs use.

Table 3: Trend Factor per Policy and Total Layer Trend

	Layer of Loss				
Policy Limit	\$250k xs 0	\$250k xs \$250k	\$500k xs \$500k	\$4MM xs \$1MM	\$5MM xs 0
250,000	1.248				1.248
500,000	1.248	1.406			1.283
750,000	1.248	1.406	1.456		1.304
1,000,000	1.248	1.406	1.470		1.318
5,000,000	1.248	1.406	1.470	1.581	1.389
Total	1.248	1.406	1.468	1.581	1.328
Annualized	4.53%	7.06%	7.98%	9.59%	5.83%

Figure 4.2.: Mata and Verheyen (2005), Table 3.

Table 4: Expected number of claims by policy and frequency trend by layer

		\$250k *xs* 0		\$250k *xs* \$250k		\$500k *xs* \$500k		\$4MM *xs* \$1MM		\$5MM *xs* 0	
Limit ('000)	$E[N_p]$	2000	2005	2000	2005	2000	2005	2000	2005	2000	2005
250	27.81	27.81	27.81							27.81	27.81
500	50.30	50.30	50.30	3.17	4.35					50.30	50.30
750	23.64	23.64	23.64	1.49	2.05	0.79	1.13			23.64	23.64
1,000	46.49	46.49	46.49	2.93	4.03	1.55	2.23			46.49	46.49
5,000	46.07	46.07	46.07	2.90	3.99	1.54	2.21	0.75	1.13	46.07	46.07
Total	194.31	194.31	194.31	10.49	14.42	3.88	5.57	0.75	1.13	194.31	194.31
Excess Frequency Trend		1.00		1.37		1.44		1.50		1.00	
Annualized Trend		0%		6.57%		7.50%		8.47%		0%	

Figure 4.3.: Mata and Verheyen (2005), Table 4.

Table 5: Summary of trend factors by layer of loss

	\$250k *xs* 0	\$250k *xs* \$250k	\$500k *xs* \$500k	\$4MM *xs* \$1MM	\$5MM *xs* 0
Layer Trend Factor	1.248	1.406	1.468	1.581	1.328
Annual Trend Factor	4.53%	7.06%	7.98%	9.59%	5.83%
Frequency Trend Factor	1.00	1.37	1.44	1.50	1.00
Annual Frequency Trend	0.00%	6.57%	7.50%	8.47%	0.00%
Severity Trend Factor	1.248	1.023	1.022	1.053	1.328
Annual Severity Trend	4.53%	0.46%	0.44%	1.04%	5.83%

Figure 4.4.: Mata and Verheyen (2005), Table 5.

Table 11: Exposure Adjustment by Layer of Loss

	Layer of Loss				
Limit	\$250k *xs* 0	\$250k *xs* \$250k	\$500k *xs* \$500k	\$4MM *xs* \$1MM	\$5MM *xs* 0
250,000	0.667				0.667
500,000	0.800	0.800			0.800
750,000	0.667	0.667	0.667		0.667
1,000,000	1.333	1.333	1.333		1.333
5,000,000	2.000	2.000	2.000	2.000	2.000
Total	0.982	1.066	1.354	2.000	1.090

Figure 4.5.: Mata and Verheyen (2005), Table 11.

Table 12: Expected Frequency by Policy Limit

Limit	$E[M_p]$	$E[N_p]$
250,000	27.81	41.72
500,000	50.30	62.87
750,000	23.64	35.45
1,000,000	46.49	34.87
5,000,000	46.07	23.03
Total	194.31	197.95

Figure 4.6.: Mata and Verheyen (2005), Table 12.

Table 13: Frequency Exposure Adjustment by Layer of Loss

	$250k xs 0		$250k xs $250k		$500k xs $500k		$4MM xs $1MM		$5MM xs 0	
$(1 - F_Y(A))$	1.000		0.087		0.048		0.025		1.000	
Limit ('000)	2000	2005	2000	2005	2000	2005	2000	2005	2000	2005
250	41.72	27.81							41.72	27.81
500	62.87	50.30	5.44	4.35					62.87	50.30
750	35.45	23.64	3.07	2.05	1.70	1.13			35.45	23.64
1,000	34.87	46.49	3.02	4.03	1.67	2.23			34.87	46.49
5,000	23.03	46.07	1.99	3.99	1.11	2.21	0.57	1.13	23.03	46.07
Total	197.95	194.31	13.53	14.42	4.48	5.57	0.57	1.13	197.95	194.31
Freq. Adj.	0.98		1.07		1.24		2.00		0.98	

Figure 4.7.: Mata and Verheyen (2005), Table 13.

Table 14: Summary of Exposure Adjustment by Layer of Loss

	$250k xs 0	$250k xs $250k	$500k xs $500k	$4MM xs $1MM	$5MM xs 0
Layer	0.982	1.066	1.354	2.000	1.090
Frequency	0.982	1.066	1.245	2.000	0.982
Severity	1.000	1.000	1.087	1.000	1.110

Figure 4.8.: Mata and Verheyen (2005), Table 14.

```python
def sf(a, mean):
    if a == 0:
        return 1.0          # continuous severity, no atom at the origin
    return float(build(f'agg M 1 claim sev lognorm {mean} cv {CV} fixed',
                       log2=LOG2, bs=BS, normalize=False).density_df.loc[a, 'S'])

def pipeline(L):
    def lsev(p, limit, att, y):
        return L[(min(p, att + limit), y)] - L[(min(p, att), y)]

    pct = pd.DataFrame({n: [lsev(p, l, a, 2005) / L[(p, 2005)] for p in PL]
                        for n, (l, a) in zip(LN, LAYERS)},
                       index=[f'{p:,.0f}' for p in PL])
    lc05 = pct.mul(np.array(SP05) * ELR, axis=0)
    lc00 = pct.mul(np.array(SP00) * OL * ELR, axis=0)
    tr = pd.DataFrame({n: [(lambda d: lsev(p, l, a, 2005) / d if d > 0 else np.nan)
                           (lsev(p, l, a, 2000)) for p in PL]
                       for n, (l, a) in zip(LN, LAYERS)},
                      index=[f'{p:,.0f}' for p in PL])
    layer_trend = pd.Series({c: (tr[c] * lc05[c]).sum() / lc05[c].sum()
                             for c in tr.columns})
    EN = {p: s * ELR / L[(p, 2005)] for p, s in zip(PL, SP05)}
    EM = {p: s * OL * ELR / L[(p, 2005)] for p, s in zip(PL, SP00)}
    ft = {n: (sum(EN[p] * sf(a, M05) for p in PL if p > a)
              / sum(EN[p] * sf(a, M00) for p in PL if p > a))
          for n, (l, a) in zip(LN, LAYERS)}
    fa = {n: (sum(EN[p] * sf(a, M05) for p in PL if p > a)
              / sum(EM[p] * sf(a, M05) for p in PL if p > a))
          for n, (l, a) in zip(LN, LAYERS)}
    ea = lc05.sum() / lc00.sum()
    return pd.DataFrame({'T3 layer trend': layer_trend,
                         'T4 freq trend': pd.Series(ft),
                         'T5 sev trend': layer_trend / pd.Series(ft),
                         'T11 exposure adj': ea,
                         'T13 freq adj': pd.Series(fa),
```

```
                        'T14 sev adj': ea / pd.Series(fa)}), lc05

from_agg, lc_agg = pipeline(AGG)
from_paper, lc_paper = pipeline(PAP)
PRINTED = pd.DataFrame(
    {'T3 layer trend': [1.248, 1.406, 1.468, 1.581, 1.328],
     'T4 freq trend': [1.00, 1.37, 1.44, 1.50, 1.00],
     'T5 sev trend': [1.248, 1.023, 1.022, 1.053, 1.328],
     'T11 exposure adj': [0.982, 1.066, 1.354, 2.000, 1.090],
     'T13 freq adj': [0.982, 1.066, 1.245, 2.000, 0.982],
     'T14 sev adj': [1.000, 1.000, 1.087, 1.000, 1.110]}, index=LN)
GT(from_paper.rename_axis('layer'), float_format=4)
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning: M:
severity 'lognorm' cannot be reproduced on this grid. The body needs bs <= 565.546 (7.5e-06% of
the mean is supplied below the first half bucket, and is placed at 0); the mean is not complete
until the grid reaches 15,491,470,518, so matching it to 0.0001 needs log2 = 25, and log2 = 20 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)
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
M: aggregate PMF deficit 3.446e-04 (Σp = 1 − 3.446e-04 < 1); forwards and backwards S diverge by
the deficit (forwards > backwards). Further occurrences this session are suppressed; each object
records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning: M:
severity 'lognorm' cannot be reproduced on this grid. The body needs bs <= 384.901 (1.843e-05% of
the mean is supplied below the first half bucket, and is placed at 0); the mean is not complete
until the grid reaches 10,543,234,532, so matching it to 0.0001 needs log2 = 25, and log2 = 20 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)
```

Table 4.3.: The whole pipeline, Tables 3, 4, 5, 11, 13 and 14, driven by the paper's own Table 1 figures.

layer	T3 layer trend	T4 freq trend	T5 sev trend	T11 exposure adj	T13 freq adj	T14 sev adj
$250k xs 0	1.2478	1.0000	1.2478	0.9816	0.9816	1.0000
$250k xs $250k	1.4062	1.3745	1.0230	1.0657	1.0657	1.0000
$500k xs $500k	1.4678	1.4358	1.0222	1.3535	1.2446	1.0875
$4MM xs $1MM	1.5810	1.5015	1.0529	2.0000	2.0000	1.0000
$5MM xs 0	1.3277	1.0000	1.3277	1.0900	0.9816	1.1105

```
GT(pd.DataFrame({"largest deviation, driven by the paper's Table 1":
                     (from_paper - PRINTED).abs().max(),
                 'largest deviation, driven by aggregate':
                     (from_agg - PRINTED).abs().max(),
                 'decimals the paper prints': [3, 2, 3, 3, 3, 3]}).rename_axis('quantity'),
   formatters={"largest deviation, driven by the paper's Table 1": 6,
               'largest deviation, driven by aggregate': 6,
               'decimals the paper prints': 0})
```

Table 4.4.: Largest deviation from the printed values under each of the two runs, beside the precision the paper prints.

quantity	largest deviation, driven by the paper's Table 1	largest deviation, driven by aggregate	decimals the paper prints
T3 layer trend	0.000251	0.001804	3
T4 freq trend	0.004529	0.004529	2
T5 sev trend	0.000251	0.001268	3
T11 exposure adj	0.000463	0.000769	3
T13 freq adj	0.000398	0.000723	3
T14 sev adj	0.000492	0.000844	3

Table 4.3 is the answer and Table 4.4 scores it: driven by the paper's own Table 1, every column reproduces to the last digit printed. The `T4 freq trend` column looks worse only because the paper prints it to two decimals, and it is identical in both runs since the frequency trend depends on survival probabilities alone and never touches a limited expected value.

Table 2's layer loss costs, Figure 4.9, come back within $60 on layers of up to $9.4 million in Table 4.5.

Table 2: Exposure Rating using 2005 severity distribution

	Layer of Loss				
Policy Limit	\$250k xs 0	\$250k xs \$250k	\$500k xs \$500k	\$4MM xs \$1MM	\$5MM xs 0
250,000	1,350,000	0	0	0	1,350,000
500,000	2,441,429	798,571	0	0	3,240,000
750,000	1,147,250	375,256	232,495	0	1,755,000
1,000,000	2,256,760	738,167	785,072	0	3,780,000
5,000,000	2,236,034	731,388	777,862	1,654,717	5,400,000
Total	9,431,473	2,643,382	1,795,428	1,654,717	15,525,000

Figure 4.9.: Mata and Verheyen (2005), Table 2.

```
GT(pd.DataFrame({'from paper Table 1': lc_paper.sum(),
                 'paper Table 2': [9_431_473, 2_643_382, 1_795_428, 1_654_717,
                                   15_525_000],
                 'from aggregate': lc_agg.sum()}).rename_axis('layer'), float_format=0)
```

Table 4.5.: Layer loss costs from each run, against the printed Table 2.

layer	from paper Table 1	paper Table 2	from aggregate
\$250k xs 0	9,431,477	9,431,473	9,420,254
\$250k xs \$250k	2,643,433	2,643,382	2,645,445
\$500k xs \$500k	1,795,369	1,795,428	1,798,459
\$4MM xs \$1MM	1,654,721	1,654,717	1,660,841
\$5MM xs 0	15,525,000	15,525,000	15,525,000

So the implementation of the method is verified exactly, and the only discrepancy in the whole reproduction is Table 1. Driven by `aggregate`, the derived quantities move in the third decimal, as Table 4.6 sets out: the $4MM xs $1MM layer trend is 1.5792 rather than 1.5810, an annualized 9.57% rather than 9.59%.

```
GT(pd.DataFrame({'agg layer trend': from_agg['T3 layer trend'],
                 'agg annualized': from_agg['T3 layer trend'] ** (1 / YEARS) - 1,
                 'paper layer trend': PRINTED['T3 layer trend'],
                 'paper annualized': PRINTED['T3 layer trend'] ** (1 / YEARS) - 1,
                 'agg freq trend': from_agg['T4 freq trend'],
                 'agg freq annualized':
                     from_agg['T4 freq trend'] ** (1 / YEARS) - 1}).rename_axis('layer'),
   formatters={'agg layer trend': 4, 'paper layer trend': 4,
               'agg freq trend': 4, 'agg annualized': '{x:.2%}',
               'paper annualized': '{x:.2%}', 'agg freq annualized': '{x:.2%}'})
```

Table 4.6.: Layer and frequency trend by layer, as multiples over five years and annualized, under both runs.

layer	agg layer trend	agg annualized	paper layer trend	paper annualized	agg freq trend	agg freq annualized
$250k xs 0	1.2474	4.52%	1.2480	4.53%	1.0000	0.00%
$250k xs $250k	1.4051	7.04%	1.4060	7.05%	1.3745	6.57%
$500k xs $500k	1.4664	7.96%	1.4680	7.98%	1.4358	7.50%
$4MM xs $1MM	1.5792	9.57%	1.5810	9.59%	1.5015	8.47%
$5MM xs 0	1.3273	5.83%	1.3280	5.84%	1.0000	0.00%

4.5. Comments

4.5.1. What is wrong with Table 1

The paper's limited expected values are 0.04% high at the lowest limit and 0.24% low at the highest, in a smooth progression, in both years. That shape is the signature of a numerical integration that loses tail as the upper limit moves out, which is what a spreadsheet doing $\int_0^a x f(x)\,\mathrm{d}x$ on a fixed number of uniform steps would produce. A uniform-step trapezoid rule with roughly 900 to 1000 steps gets close to the pattern but does not match it at every limit, so the scheme is not identified here, only its symptom.

Nothing about the paper's conclusions depends on it. The published deltas are large, a 9.59% layer trend against 8% unlimited, and a doubled exposure against a 9% total limits increase, and the third-decimal correction leaves all of them intact. The finding is worth recording because it is exactly the kind of thing a reproduction is for: two independent computations of the same quantity, one of which turns out to be a numerical approximation nobody flagged as one.

4.5.2. Reading the survival function of a very heavy lognormal

The frequency trend needs $S(A)$ at each attachment, and $\sigma = 2.29$ gives a coefficient of variation of 13.7. On a grid reaching $26M the build reports a mass deficit of 3.4×10^{-4}, which is severity mass above the grid, and the survival values carry a relative error of about 4×10^{-5}.

Pushing the grid out does not help, it hurts. Reaching far enough to make the deficit negligible would need $x_{\max} \approx 4 \times 10^9$, and at any affordable bucket count the discretization error then dominates: at `bs = 3906` the survival error is 6×10^{-3}, two orders of magnitude worse than at `bs = 25`. For reading a survival function in the body of a distribution, resolution beats reach.

One related detail: the paper prints $1 - F_Y(A) = 1.000$ for the layers attaching at zero. A continuous severity has no atom at the origin, so the exact value is 1, but `aggregate` returns $F(0) = P(X \leq \text{bs}/2) = 0.0015$ at `bs = 25`. The code above returns 1 explicitly at $A = 0$ rather than reading it off the grid; without that the frequency trend for the two ground up layers comes out at 1.0011 instead of the correct 1.

4.5.3. Where the compound machinery would come in

As with Mack and Fackler (2003), this is a severity and layer paper. `aggregate` supplies limited expected values and survival probabilities; the frequency in equation (3.3) is a derived count, $\mathsf{E}[N_p] = (\text{SP})(\text{ELR})\beta_p/\mathsf{E}_Y[S_{Y,p}]$, not a distribution. The natural extension, which the paper does not take, is that once the layer expected loss and the claim count are both in hand, the layer's *aggregate* distribution is one more DecL clause away, and with it the aggregate deductibles and reinstatements that a real treaty carries.

5. Bodoff and excess of policy limits losses

Reproducing Bodoff (2017), Exhibit 4

5.1. Context and problem

An excess of policy limits (XPL) loss is one where the insurer pays past the policy limit, usually because a court finds it mishandled a claim it could have settled inside the limit. Exposure rating looks unworkable for XPL: the method allocates loss to a layer using the limited expected value of a severity curve capped at the policy limit, and XPL losses are by definition the ones the cap failed to stop.

Bodoff (2017) makes the obstacle disappear with one change of view. Treat the *effectiveness* of the policy limit as a Bernoulli random variable Z: with probability p the limit caps the loss, and with probability $1 - p$ it does not. Then the policy limited expected value is

$$\mathsf{PLEV}(X, k) = p\,\mathsf{E}[X \wedge k] + (1 - p)\,\mathsf{E}[X],$$

which is a mixture of two ordinary limited expected values, one at the policy limit and one unlimited. Nothing new is needed: XPL is not a separate severity curve grafted on, it is the same curve read without the cap. This is the paper's stated departure from Braithwaite and Ware (1997), whose approach required building a freestanding XPL size of loss curve.

5.2. Paper solution and gold standard

The reproduction target is Exhibit 4, Figure 5.2, the six-layer exposure rating. Its inputs are the three short exhibits in Figure 5.1: a nine-point policy limits profile with premium weights (Figure 5.1a), a Pareto severity with $\theta = 50{,}000$ and $\alpha = 1.50$ (Figure 5.1b), and $p = 99\%$ for every policy limit below \$25M (Figure 5.1c).

The layer exposure factor, the paper's equation (3.12), is

$$\frac{p\,[\mathsf{E}[X \wedge \min(\mathrm{PL}, T)] - \mathsf{E}[X \wedge \min(\mathrm{PL}, B)]] + (1 - p)\,[\mathsf{E}[X \wedge T] - \mathsf{E}[X \wedge B]]}{\mathsf{E}[X \wedge \mathrm{PL}]}$$

with T and B the top and bottom of the reinsurance layer. Traditional exposure rating is the same expression at $p = 1$.

Two model choices in those three exhibits are not spelled out and have to be inferred.

"Pareto, theta 50,000, alpha 1.50" means the two parameter Pareto, $S(x) = (\theta/(\theta + x))^{\alpha}$, the form in Clark's reinsurance study note that the paper cites, not the single parameter Pareto supported on $[\theta, \infty)$. In DecL this is `lomax`.

Exhibit 3's row "Policy Limit = $25M, p = 100%" is a global cap, not a row of the limits profile. No policy in Exhibit 1 has a $25M limit. The row is saying that the XPL branch is itself capped at $25M, which is what makes the six layers a complete tower: they run 0 to $500k, $500k to $1M, $1M to $2M, $2M to $5M, $5M to $10M, $10M to $25M, and the total of Figure 5.2 is the whole tower.

Exhibit 1. Policy limits distribution

1 Policy Limit	2 % of premium	3 ELR%
50,000	1.0%	65.0%
100,000	1.0%	65.0%
500,000	2.0%	65.0%
1,000,000	80.0%	65.0%
2,000,000	10.0%	65.0%
3,000,000	1.0%	65.0%
4,000,000	1.0%	65.0%
5,000,000	3.0%	65.0%
10,000,000	1.0%	65.0%

(a) Exhibit 1, the policy limits distribution.

Exhibit 2. Severity curve

Item #	Description	Value
1	Curve	Pareto
2	Theta	50,000
3	Alpha	1.50

(b) Exhibit 2, the severity curve.

Exhibit 3. Probabilities that policy limits successfully cap losses

	p	1 – p
All Policy Limits < $25M	99%	1.00%
Policy Limit = $25M	100%	0.00%

(c) Exhibit 3, the probabilities that policy limits cap losses.

Figure 5.1.: Bodoff (2017), Exhibits 1 to 3, the three inputs to the exposure rating.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build

LIMITS = [50e3, 100e3, 500e3, 1e6, 2e6, 3e6, 4e6, 5e6, 10e6]
WEIGHT = [.01, .01, .02, .80, .10, .01, .01, .03, .01]
LAYERS = [(500e3, 0), (500e3, 500e3), (1e6, 1e6), (3e6, 2e6),
          (5e6, 5e6), (15e6, 10e6)]
P, CAP, THETA, ALPHA = 0.99, 25e6, 50e3, 1.5
```

```
Exception reporting mode: Minimal
```

5.3. The `aggregate` reproduction

Every quantity in Exhibit 4 is a limited expected value of the severity, and a limited expected value is what a single fixed claim in the layer `a xs 0` costs. That is one `build` call.

```
def lev(a, log2=20):
    """E[X and a] as the expected loss of one fixed claim in the layer a xs 0."""
    return build(f'agg B 1 claim {a} xs 0 sev {THETA} * lomax {ALPHA} fixed',
                 log2=log2, bs=a / 2 ** (log2 - 2), normalize=False).est_m

def closed_form(a):
    """theta/(alpha-1) [1 - (theta/(theta+a))^(alpha-1)]."""
    return THETA / (ALPHA - 1) * (1 - (THETA / (THETA + a)) ** (ALPHA - 1))
```

```
pts = sorted(set(LIMITS) | {CAP} | {t + y for y, t in LAYERS}
             | {t for y, t in LAYERS} - {0})
LEV = {a: lev(a) for a in pts}
chk = pd.DataFrame({'a': pts, 'aggregate': [LEV[a] for a in pts],
                    'closed form': [closed_form(a) for a in pts]})
chk['relative error'] = (chk['aggregate'] / chk['closed form'] - 1).abs()
GT(chk)
```

Table 5.1.: Limited expected values of the Pareto severity at every breakpoint the exhibit needs, against the closed form.

a	aggregate	closed form	relative error
50,000	29,289	29,289	1.28e-12
100,000	42,265	42,265	4.03e-12
500,000	69,849	69,849	6.49e-11
1,000,000	78,178	78,178	2.33e-10
2,000,000	84,383	84,383	8.62e-10
3,000,000	87,196	87,196	1.88e-09
4,000,000	88,889	88,889	3.27e-09
5,000,000	90,050	90,050	5.05e-09
10,000,000	92,947	92,947	1.96e-08
25,000,000	95,532	95,532	1.19e-07

In Table 5.1, `aggregate` agrees with the closed form to within 1.2 parts in ten million at every limit, so the exposure rating rests on the severity and the arithmetic, not on the grid.

Exhibit 4. Exposure rating output

1	2	3	4	5	6
			Layer Losses as % of total ground up losses	Layer Losses as % of total ground up losses	Implied Loading for XPL
Layer	Limit	Attachment	Traditional Exposure Rating	Proposed Method Including XPL	Proposed / Traditional – 1
1	500,000	—	88.420%	88.440%	0.023%
2	500,000	500,000	10.067%	10.074%	0.072%
3	1,000,000	1,000,000	1.150%	1.219%	5.989%
4	3,000,000	2,000,000	0.333%	0.403%	21.057%
5	5,000,000	5,000,000	0.031%	0.068%	119.369%
6	15,000,000	10,000,000	0.000%	0.033%	#N/A
Total			100.000%	100.237%	0.237%

Figure 5.2.: Bodoff (2017), Exhibit 4.

```
LEV[0.0] = 0.0

def exposure_factor(pl, limit, att, p):
    top, bot = att + limit, att
    return (p * (LEV[min(pl, top)] - LEV[min(pl, bot)])
            + (1 - p) * (LEV[min(CAP, top)] - LEV[min(CAP, bot)])) / LEV[pl]

rows = []
for limit, att in LAYERS:
    trad = sum(w * exposure_factor(pl, limit, att, 1.0)
               for pl, w in zip(LIMITS, WEIGHT))
    prop = sum(w * exposure_factor(pl, limit, att, P)
```

```
                for pl, w in zip(LIMITS, WEIGHT))
    rows.append({'limit': limit, 'attachment': att,
                 'traditional %': 100 * trad, 'proposed %': 100 * prop,
                 'XPL loading %': 100 * (prop / trad - 1) if trad else np.nan})
ex4 = pd.DataFrame(rows)
ex4.loc['Total', ['traditional %', 'proposed %']] = ex4[
    ['traditional %', 'proposed %']].sum()
ex4.loc['Total', 'XPL loading %'] = 100 * (ex4.loc['Total', 'proposed %']
                                           / ex4.loc['Total', 'traditional %'] - 1)
ex4.index = [f'${l/1000:,.0f}k xs {a/1000:,.0f}k' for l, a in  ex4[[
    'limit', 'attachment']].values[:-1]] + ['Total']
ex4.index.name = 'Layer'
GT(ex4.iloc[:, 2:], float_format=3)
```

Table 5.2.: Exhibit 4 rebuilt, the six layer tower at $p = 1$ and at $p = 99$ with the implied XPL loading.

Layer	**traditional %**	**proposed %**	**XPL loading %**
$500k xs 0k	88.420	88.440	0.023
$500k xs 500k	10.067	10.074	0.072
$1,000k xs 1,000k	1.150	1.219	5.989
$3,000k xs 2,000k	0.333	0.403	21.057
$5,000k xs 5,000k	0.031	0.068	119.369
$15,000k xs 10,000k	0.000	0.033	—
Total	100.000	100.237	0.237

Table 5.3 transcribes what Exhibit 4 prints, for comparison with Table 5.2.

Table 5.3.: Bodoff (2017), Exhibit 4 as printed.

Layer	**Traditional**	**Proposed**	**Loading**
$500k xs 0	88.420%	88.440%	0.023%
$500k xs $500k	10.067%	10.074%	0.072%
$1M xs $1M	1.150%	1.219%	5.989%
$3M xs $2M	0.333%	0.403%	21.057%
$5M xs $5M	0.031%	0.068%	119.369%
$15M xs $10M	0.000%	0.033%	n/a
Total	**100.000%**	**100.237%**	**0.237%**

Every cell matches to the precision printed, including the 119.369% loading on $5M xs $5M and the 0.237% total. The two inferred model choices are therefore confirmed: had either the Pareto form or the $25M cap been read wrongly, the top two layers would have missed by far more than a rounding.

5.4. Comments

5.4.1. The result the paper is selling

Bodoff's point is not the total, which is a modest 0.237% load on the whole tower. It is the shape of column 6, isolated in Table 5.4.

```
sh = ex4.drop('Total')[['attachment', 'traditional %', 'proposed %',
                        'XPL loading %']].copy()
sh['share of gross loss'] = sh['proposed %'] / ex4.loc['Total', 'proposed %']
GT(sh.set_index('attachment'))
```

Table 5.4.: The XPL loading by attachment, beside each layer's share of gross loss.

attachment	traditional %	proposed %	XPL loading %	share of gross loss
0	88.42	88.44	0.02	0.88231
500,000	10.07	10.07	0.07	0.10050
1,000,000	1.15	1.22	5.99	0.01216
2,000,000	0.33	0.40	21.06	0.00402
5,000,000	0.03	0.07	119.37	0.00068
10,000,000	0.00	0.03	—	0.00033

In Table 5.4 one parameter, a single flat $p = 99\%$ applied to every policy, produces XPL loadings that run from 2 basis points on the working layer to 119% on \$5M xs \$5M and to an infinite relative load on \$15M xs \$10M, where traditional exposure rating returns exactly zero because no policy in the profile has a limit above \$10M. That last column is the practical payoff: a layer that traditional exposure rating prices at zero is not in fact free, and the method gives a number for it rather than a shrug.

The loadings also depend on the limits profile and not only on p, which is easy to miss. The 80% concentration at the \$1M limit is what drives the loading pattern; a book written mostly at \$5M limits would get a quite different column 6 from the same p.

5.4.2. Where this sits for `aggregate`

This is a severity and layer exercise, not a compound distribution exercise. There is no frequency in the paper at all: Exhibit 4 is expressed in percentages of ground up loss, and the expected loss ratio of 65% in Exhibit 1 cancels out of every ratio. `aggregate` is doing one job here, computing $\mathsf{E}[X \wedge a]$ for a heavy Pareto, and doing it to eight figures.

That said, the paper's section 3.1.1 sketches exactly the collective risk model `aggregate` is for: simulate a loss, simulate the policy limit attached to it, then simulate Z and cap or do not cap. In DecL that whole construction is a mixed severity, since a Bernoulli mixture of a limited and an unlimited curve is just a two point severity mixture, and the surrounding frequency comes free. Bodoff's parsimony argument, that XPL "is not a completely new entity" but the same size of loss distribution read without a cap, is exactly what makes it expressible.

5.4.3. A caveat on the severity

`Bodoff2017.Pareto` is one of the curves already transcribed into `actuarial-severity-curves.agg`. With $\alpha = 1.5$ it has infinite variance, so it raises `InfiniteVarianceError` if built unlimited. Every use here is inside a layer, which bounds it. The mean is finite, $\theta/(\alpha - 1) = 100{,}000$, and the \$25M cap in Exhibit 3 brings even the XPL branch down to 95,532.

6. Homer and Clark's bivariate FFT

Reproducing Homer and Clark (2003)

6.1. Context and problem

The collective risk model gives the distribution of a sum of claims. It does not, on its own, give the *joint* distribution of two quantities that both depend on the same claims. Homer and Clark (2003) gives three examples where the joint distribution is needed.

1. An insured retains the first $600,000 of each occurrence and buys a stop loss on the total retention. The insurer thus writes two covers: a per occurrence excess of $600,000 and an aggregate stop loss on the retention, a "specific and agg". It can price either cover alone, but not their sum, because the two are dependent: an excess loss can only happen if the retention was breached.
2. A general liability policy caps indemnity but pays defense costs in addition, so the aggregate is loss capped at a policy aggregate plus uncapped expense, with the two correlated at the claim level.
3. A dynamic financial analysis model simulates small losses in aggregate and large losses individually, and treating them as independent understates the tail, because a high claim count raises both.

Their answer is the 2-d analog of our standard FFT-convolution algorithm: build a bivariate per-claim severity matrix M_x whose (j, k) entry is the joint probability of the two components, then

$$M_z = \text{IFFT}\left(\text{PGF}\left(\text{FFT}(M_x)\right)\right)$$

with a two-dimensional transform and the claim count probability generating function applied elementwise. The frequency couples the two axes; the severity matrix carries the per-claim dependence; and one transform delivers the joint aggregate.

Mildenhall (2005) describes a second approach with a bivariate frequency distribution, but that is not considered here.

6.2. Paper solution and gold standard

The paper provides three fully specified examples, each with the input and output matrices, as Table 6.1 sets out.

Table 6.1.: The three worked examples, their axes, their frequencies and what each publishes.

Section	Axes	Frequency	Published
2	retained loss, excess loss	NB mean 5, variance 6	Tables 2.3 to 2.6 plus five statistics
3	indemnity loss, ALAE	NB mean 4, variance 8	Tables 3.3 to 3.6
4	small loss dollars, large loss **counts**	NB mean 10, variance 20	Tables 4.2 to 4.4

Appendix A provides an R code doing a 3×3 two-dimensional FFT by hand, and Appendix B derives a closed-form covariance between small and large aggregate losses. Both are checkable.

The negative binomial enters through its probability generating function, which the paper writes in terms of mean M and variance V as

$$\phi(t) = (V/M - (V/M - 1)t)^{-M^2/(V-M)}.$$

Note that for over-dispersed distributions $V > M$, with the Poisson lower limit $V = M$. `aggregate` describes the same law as a gamma-mixed Poisson: matching $n + n^2c^2 = V$ at $n = M$ gives mixing coefficient of variation

$$c = \sqrt{(V - M)/M^2} = \sqrt{(\nu - 1)/n}$$

where $\nu = V/M$ is the variance-to-mean ratio. Table 6.2 translates the paper's three probability generating functions into the mixing CV each build needs.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build

K = 200_000.0

def nb_cv(mean, var):
    """Gamma mixing CV reproducing a negative binomial with this mean and variance."""
    return float(np.sqrt((var - mean) / mean ** 2))

GT(pd.DataFrame([
    {'section': 2, 'NB mean': 5, 'NB variance': 6, 'paper PGF': '$(1.2 - 0.2t)^{-25}$',
     'mixed gamma cv': nb_cv(5, 6)},
    {'section': 3, 'NB mean': 4, 'NB variance': 8, 'paper PGF': '$(2 - t)^{-4}$',
     'mixed gamma cv': nb_cv(4, 8)},
    {'section': 4, 'NB mean': 10, 'NB variance': 20, 'paper PGF': '$(2 - t)^{-10}$',
     'mixed gamma cv': nb_cv(10, 20)},
]))
```

```
Exception reporting mode: Minimal
```

Table 6.2.: The paper's three negative binomial frequencies, and the gamma mixing CV that reproduces each.

section	NB mean	NB variance	paper PGF	mixed gamma cv
2	5	6	$(1.2 - 0.2t)^{-25}$	0.20000
3	4	8	$(2 - t)^{-4}$	0.50000
4	10	20	$(2 - t)^{-10}$	0.31623

Before introducing the insurance examples, we provide an arithmetic check. Appendix A convolves

$$M_X = \begin{pmatrix} .4 & 0 & 0 \\ .3 & .3 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

with itself and prints the answer. In DecL that is a fixed count of two against a discrete bivariate severity, computed in Table 6.3.

```
toy = build('bivariate A dfreq [2] dbvsev [0 1 2] [0 1 2] '
            '[[.4 0 0] [.3 .3 0] [0 0 0]]', bs=(1., 1.), log2=(3, 3))
GT(pd.DataFrame(toy.density[:3, :3], index=[0, 1, 2], columns=[0, 1, 2]),
   float_format=6)
```

Table 6.3.: The Appendix A 3×3 matrix convolved with itself.

level_0	0	1	2
0	0.160000	0.000000	0.000000
1	0.240000	0.240000	0.000000
2	0.090000	0.180000	0.090000

The paper's R session returns `0.16, 0.24, 0.09` down the first column and `0.24, 0.18, 0.09` for the rest, which is what Table 6.3 came back with.

6.3. Section 2: Per-Occurrence And Stop Loss

Dietrichson Drilling retains $600,000 per occurrence and buys $400,000 xs $600,000 per occurrence plus $5,000,000 xs $3,000,000 on the aggregate retention. Severity is the five point discrete law of Figure 6.1; frequency is negative binomial with mean 5 and variance 6.

TABLE 2.2

SEVERITY DISTRIBUTION FOR DIETRICHSON DRILLING

Probability	Loss Amount	Excess Loss
0.00%	0	0
37.80%	200,000	0
23.50%	400,000	0
14.60%	600,000	0
9.10%	800,000	200,000
15.00%	1,000,000	400,000
Average	480,000	78,200

Figure 6.1.: Homer and Clark (2003), Table 2.2.

The paper builds Table 2.3, Figure 6.2, the joint per-claim law of retained and ceded, by hand. `aggregate` does not need it built: retained and ceded are the **net and ceded views of an occurrence layer**, so the bivariate severity is the cession map applied to the gross severity. That is

`occ_bivariate`, and the DecL keyword is `netceded`. The remaining targets for this section are Figure 6.3, Figure 6.4 and Figure 6.5.

TABLE 2.3

SINGLE CLAIM PRIMARY & EXCESS LOSS BIVARIATE DISTRIBUTION

Loss Capped at 600,000	Loss Excess of 600,000					
	0	200,000	400,000	600,000	800,000	1,000,000
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
200,000	**37.80%**	0.00%	0.00%	0.00%	0.00%	0.00%
400,000	**23.50%**	0.00%	0.00%	0.00%	0.00%	0.00%
600,000	**14.60%**	**9.10%**	**15.00%**	0.00%	0.00%	0.00%
800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.2.: Homer and Clark (2003), Table 2.3.

TABLE 2.4

AGGREGATE PRIMARY & AGGREGATE EXCESS LOSS
BIVARIATE DISTRIBUTION

Loss Capped at 600,000	Loss Excess of 600,000						
	0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	1.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
200,000	1.65%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
400,000	2.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
600,000	3.09%	0.40%	0.66%	0.00%	0.00%	0.00%	0.00%
800,000	3.34%	0.65%	1.07%	0.00%	0.00%	0.00%	0.00%
1,000,000	3.39%	0.96%	1.58%	0.00%	0.00%	0.00%	0.00%
1,200,000	3.22%	1.27%	2.16%	0.26%	0.21%	0.00%	0.00%
1,400,000	2.86%	1.40%	2.44%	0.44%	0.36%	0.00%	0.00%
1,600,000	2.43%	1.45%	2.59%	0.66%	0.54%	0.00%	0.00%
1,800,000	1.97%	1.40%	2.57%	0.90%	0.78%	0.09%	0.05%
2,000,000	1.54%	1.26%	2.38%	1.02%	0.92%	0.15%	0.08%
2,200,000	1.17%	1.09%	2.12%	1.08%	1.01%	0.24%	0.13%
2,400,000	0.86%	0.90%	1.80%	1.08%	1.05%	0.33%	0.20%
2,600,000	0.62%	0.72%	1.47%	1.00%	1.01%	0.38%	0.24%
2,800,000	0.43%	0.55%	1.16%	0.88%	0.93%	0.42%	0.27%
3,000,000	0.29%	0.41%	0.89%	0.75%	0.82%	0.43%	0.29%
3,200,000	0.20%	0.30%	0.66%	0.61%	0.70%	0.41%	0.29%
3,400,000	0.13%	0.21%	0.48%	0.48%	0.57%	0.37%	0.28%
3,600,000	0.08%	0.15%	0.34%	0.37%	0.45%	0.32%	0.25%
3,800,000	0.05%	0.10%	0.24%	0.27%	0.35%	0.27%	0.22%
4,000,000	0.03%	0.07%	0.16%	0.20%	0.26%	0.22%	0.19%
4,200,000	0.02%	0.04%	0.11%	0.14%	0.19%	0.17%	0.15%
4,400,000	0.01%	0.03%	0.07%	0.10%	0.14%	0.13%	0.12%
4,600,000	0.01%	0.02%	0.05%	0.07%	0.10%	0.10%	0.09%
4,800,000	0.00%	0.01%	0.03%	0.04%	0.07%	0.07%	0.07%
5,000,000	0.00%	0.01%	0.02%	0.03%	0.04%	0.05%	0.05%
5,200,000	0.00%	0.00%	0.01%	0.02%	0.03%	0.03%	0.04%
5,400,000	0.00%	0.00%	0.01%	0.01%	0.02%	0.02%	0.03%
5,600,000	0.00%	0.00%	0.00%	0.01%	0.01%	0.02%	0.02%
5,800,000	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%
6,000,000	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%
6,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%
6,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
8,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.3.: Homer and Clark (2003), Table 2.4.

TABLE 2.5

AGGREGATE PRIMARY EXCESS & AGGREGATE EXCESS LOSS BIVARIATE DISTRIBUTION

Stop Loss	Loss Excess of 600,000						
	0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	30.28%	12.45%	22.89%	8.07%	7.65%	2.04%	1.26%
200,000	0.20%	0.30%	0.66%	0.61%	0.70%	0.41%	0.29%
400,000	0.13%	0.21%	0.48%	0.48%	0.57%	0.37%	0.28%
600,000	0.08%	0.15%	0.34%	0.37%	0.45%	0.32%	0.25%
800,000	0.05%	0.10%	0.24%	0.27%	0.35%	0.27%	0.22%
1,000,000	0.03%	0.07%	0.16%	0.20%	0.26%	0.22%	0.19%
1,200,000	0.02%	0.04%	0.11%	0.14%	0.19%	0.17%	0.15%
1,400,000	0.01%	0.03%	0.07%	0.10%	0.14%	0.13%	0.12%
1,600,000	0.01%	0.02%	0.05%	0.07%	0.10%	0.10%	0.09%
1,800,000	0.00%	0.01%	0.03%	0.04%	0.07%	0.07%	0.07%
2,000,000	0.00%	0.01%	0.02%	0.03%	0.04%	0.05%	0.05%
2,200,000	0.00%	0.00%	0.01%	0.02%	0.03%	0.03%	0.04%
2,400,000	0.00%	0.00%	0.01%	0.01%	0.02%	0.02%	0.03%
2,600,000	0.00%	0.00%	0.00%	0.01%	0.01%	0.02%	0.02%
2,800,000	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%
3,000,000	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%
3,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%
3,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
5,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.4.: Homer and Clark (2003), Table 2.5.

TABLE 2.6

PROBABILITIES FOR AGGREGATE PRIMARY EXCESS PLUS AGGREGATE EXCESS LOSS

Loss & ALAE	Probability	Calculation
0	30.28%	= 30.28%
200,000	12.64%	= 0.20% + 12.45%
400,000	23.31%	= 0.13% + 0.30% + 22.89%
600,000	9.02%	= 0.08% + 0.21% + 0.66% + 8.07%
800,000	8.94%	= 0.05% + 0.15% + 0.48% + 0.61% + 7.65%
⋮	⋮	⋮

Figure 6.5.: Homer and Clark (2003), Table 2.6.

Table 6.4 records the grid the build lands on and checks the severity mean against the paper's.

```
SEV = 'dsev [200000 400000 600000 800000 1000000] [.378 .235 .146 .091 .15]'
a = build(f'agg HC 5 claims {SEV} occurrence net of 400000 xs 600000 '
          f'mixed gamma {nb_cv(5, 6)}', bs=K, log2=8)
bd = a.occ_bivariate(views=('net', 'ceded'), bs=K, log2_x=7, log2_y=5)
gx, gy = bd.axis_xs
d = bd.density
GT(pd.DataFrame([{'grid': str(d.shape), 'bs': str(bd.bs),
                  'deficit': bd.deficit, 'total mass': d.sum(),
                  'severity mean': (np.arange(5) + 1) @ np.array(
                      [.378, .235, .146, .091, .15]) * K,
                  'paper severity mean': 480_000}]))
```

Table 6.4.: The section 2 bivariate grid, its mass deficit, and the severity mean against the paper's $480,000.

grid	bs	deficit	total mass	severity mean	paper severity mean
(128, 32)	[200000.0, 200000.0]	93.703f	1.00000	480,000	480,000

The single claim joint law is a by-product, and it is Table 2.3: compare Table 6.5 with Figure 6.2.

```
S = np.asarray(bd._S.todense()) if hasattr(bd._S, 'todense') else bd._S
GT(pd.DataFrame(100 * S[:6, :4], index=[f'{v:,.0f}' for v in gx[:6]],
                columns=[f'{v:,.0f}' for v in gy[:4]]), float_format=2)
```

Table 6.5.: The joint per-claim law of retained and ceded, as percentages, read off the cession map.

level_0	0	200,000	400,000	600,000
0	0.00	0.00	0	0
200,000	37.80	0.00	0	0
400,000	23.50	0.00	0	0
600,000	14.60	9.10	15	0
800,000	0.00	0.00	0	0
1,000,000	0.00	0.00	0	0

Table 2.4 is the aggregate joint. The paper prints a 26×7 block of it to two decimals, reproduced in Table 6.6 and scored in Table 6.7.

```
T24 = {0: [1.05, 0, 0, 0, 0, 0, 0], 200_000: [1.65, 0, 0, 0, 0, 0, 0],
       400_000: [2.38, 0, 0, 0, 0, 0, 0],
       600_000: [3.09, .40, .66, 0, 0, 0, 0],
       800_000: [3.34, .65, 1.07, 0, 0, 0, 0],
       1_000_000: [3.39, .96, 1.58, 0, 0, 0, 0],
       1_200_000: [3.22, 1.27, 2.16, .26, .21, 0, 0],
       1_400_000: [2.86, 1.40, 2.44, .44, .36, 0, 0],
       1_600_000: [2.43, 1.45, 2.59, .66, .54, 0, 0],
       1_800_000: [1.97, 1.40, 2.57, .90, .78, .09, .05],
       2_000_000: [1.54, 1.26, 2.38, 1.02, .92, .15, .08],
       2_200_000: [1.17, 1.09, 2.12, 1.08, 1.01, .24, .13],
       2_400_000: [.86, .90, 1.80, 1.08, 1.05, .33, .20],
       2_600_000: [.62, .72, 1.47, 1.00, 1.01, .38, .24],
       2_800_000: [.43, .55, 1.16, .88, .93, .42, .27],
```

```
       3_000_000: [.29, .41, .89, .75, .82, .43, .29],
       3_200_000: [.20, .30, .66, .61, .70, .41, .29],
       3_400_000: [.13, .21, .48, .48, .57, .37, .28],
       3_600_000: [.08, .15, .34, .37, .45, .32, .25],
       3_800_000: [.05, .10, .24, .27, .35, .27, .22],
       4_000_000: [.03, .07, .16, .20, .26, .22, .19],
       4_200_000: [.02, .04, .11, .14, .19, .17, .15],
       4_400_000: [.01, .03, .07, .10, .14, .13, .12],
       4_600_000: [.01, .02, .05, .07, .10, .10, .09],
       4_800_000: [0, .01, .03, .04, .07, .07, .07],
       5_000_000: [0, .01, .02, .03, .04, .05, .05]}
got24 = np.array([100 * d[int(round(r / K)), :7] for r in T24])
pap24 = np.array(list(T24.values()))
GT(pd.DataFrame(got24, index=[f'{r:,.0f}' for r in T24],
                columns=[f'{int(j * K):,.0f}' for j in range(7)]),
   float_format=2)
```

Table 6.6.: The aggregate joint distribution of net and ceded, the 26 × 7 block Table 2.4 prints, as percentages.

level_0	0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	1.05	0.00	0.00	0.00	0.00	0.00	0.00
200,000	1.65	0.00	0.00	0.00	0.00	0.00	0.00
400,000	2.38	0.00	0.00	0.00	0.00	0.00	0.00
600,000	3.09	0.40	0.66	0.00	0.00	0.00	0.00
800,000	3.34	0.65	1.07	0.00	0.00	0.00	0.00
1,000,000	3.39	0.96	1.58	0.00	0.00	0.00	0.00
1,200,000	3.22	1.27	2.16	0.26	0.21	0.00	0.00
1,400,000	2.86	1.40	2.44	0.44	0.36	0.00	0.00
1,600,000	2.43	1.45	2.59	0.66	0.54	0.00	0.00
1,800,000	1.97	1.40	2.57	0.90	0.78	0.09	0.05
2,000,000	1.54	1.26	2.38	1.02	0.92	0.15	0.08
2,200,000	1.17	1.09	2.12	1.08	1.01	0.24	0.13
2,400,000	0.86	0.90	1.80	1.08	1.05	0.33	0.20
2,600,000	0.62	0.72	1.47	1.00	1.01	0.38	0.24
2,800,000	0.43	0.55	1.16	0.88	0.93	0.42	0.27
3,000,000	0.29	0.41	0.89	0.75	0.82	0.43	0.29
3,200,000	0.20	0.30	0.66	0.61	0.70	0.41	0.29
3,400,000	0.13	0.21	0.48	0.48	0.57	0.37	0.28
3,600,000	0.08	0.15	0.34	0.37	0.45	0.32	0.25
3,800,000	0.05	0.10	0.24	0.27	0.35	0.27	0.22
4,000,000	0.03	0.07	0.16	0.20	0.26	0.22	0.19
4,200,000	0.02	0.04	0.11	0.14	0.19	0.17	0.15
4,400,000	0.01	0.03	0.07	0.10	0.14	0.13	0.12
4,600,000	0.01	0.02	0.05	0.07	0.10	0.10	0.09
4,800,000	0.00	0.01	0.03	0.04	0.07	0.07	0.07
5,000,000	0.00	0.01	0.02	0.03	0.04	0.05	0.05

```
GT(pd.DataFrame([{'cells compared': got24.size,
                  'worst absolute difference, points of probability':
                      float(np.abs(got24 - pap24).max())}]))
```

Table 6.7.: Worst cell error over the whole published block of Table 2.4.

cells compared	worst absolute difference, points of probability
182	0.00497

Table 2.5, Figure 6.4, applies the stop loss: collapse every net outcome at or below $3,000,000 into one row and relabel the rest. Then the five statistics the paper quotes fall straight out of the joint, in Table 6.8.

```
k = int(round(3_000_000 / K))
d25 = np.vstack([d[:k + 1].sum(axis=0)[None, :], d[k + 1:]])
g25 = np.arange(d25.shape[0]) * K
hit = 1 - d25.sum(axis=1)[0]
GT(pd.DataFrame([
    {'statistic': 'expected loss to the stop loss layer',
     'aggregate': (d25.sum(axis=1) * g25).sum(), 'paper': 123_529},
    {'statistic': 'probability the stop loss is hit',
     'aggregate': hit, 'paper': 0.1508},
    {'statistic': 'mean stop loss given it is hit',
     'aggregate': (d25[1:].sum(axis=1) * g25[1:]).sum() / hit, 'paper': 819_210},
    {'statistic': 'expected loss to the per-occurrence layer',
     'aggregate': (d25.sum(axis=0) * gy).sum(), 'paper': 391_000},
    {'statistic': 'expected per-occurrence loss given the stop loss is hit',
     'aggregate': (d25[1:].sum(axis=0) * gy).sum() / hit, 'paper': 830_334},
]), float_format=4)
```

Table 6.8.: The five statistics the paper quotes off Table 2.5, taken from the joint distribution.

statistic	aggregate	paper
expected loss to the stop loss layer	123,529.2639	123,529.0000
probability the stop loss is hit	0.1508	0.1508
mean stop loss given it is hit	819,209.9260	819,210.0000
expected loss to the per-occurrence layer	391,000.0000	391,000.0000
expected per-occurrence loss given the stop loss is hit	830,334.0247	830,334.0000

Table 6.8 has all five to the digit. The last two are the paper's punchline: the per-occurrence layer costs $391,000 on average, but $830,334 in the scenarios where the stop loss also pays. Writing both covers is more than twice as bad as the two expected values suggest, and that is invisible to anyone pricing the covers separately.

Table 2.6 sums the antidiagonals to get the distribution of the two covers combined, Table 6.9.

```
n = 5
tot = np.zeros(n)
for i in range(d25.shape[0]):
    for j in range(min(d25.shape[1], n)):
        if i + j < n:
            tot[i + j] += d25[i, j]
GT(pd.DataFrame({'combined loss': [f'{i * K:,.0f}' for i in range(n)],
                 'aggregate %': 100 * tot,
                 'Table 2.6 %': [30.28, 12.64, 23.31, 9.02, 8.94]}),
   float_format=2)
```

Table 6.9.: The distribution of the two covers combined, against the five cells Table 2.6 prints.

combined loss	aggregate %	Table 2.6 %
0	30.28	30.28
200,000	12.64	12.64
400,000	23.31	23.31
600,000	9.02	9.02
800,000	8.94	8.94

6.4. Section 3: loss subject to an aggregate, plus unlimited ALAE

Here the bivariate severity is given rather than derived, so DecL takes the matrix directly with `dbv-sev`.

TABLE 3.2

SEVERITY DISTRIBUTION FOR DIETRICHSON DRILLING

Probability	Loss Amount
10.00%	0
45.00%	200,000
9.00%	400,000
9.00%	600,000
9.00%	800,000
18.00%	1,000,000
Average	432,000
Average ALAE %	37.29%

Figure 6.6.: Homer and Clark (2003), Table 3.2.

One transcription note. Table 3.3, Figure 6.7, is printed to two decimals and its row sums are 10.00%, 45.00%, 9.00%, 9.00%, 9.00% and 17.97%, against the loss marginal of Figure 6.6 which requires 18.00% on the last. The printed matrix therefore carries 99.97% of the mass. Scaling each row to its Table 3.2 marginal is the correction the paper's own inputs dictate. The outputs to match are Figure 6.8, Figure 6.9 and Figure 6.10.

TABLE 3.3

SINGLE CLAIM LOSS & ALAE BIVARIATE DISTRIBUTION

Loss Amount	ALAE 0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	8.39%	1.47%	0.13%	0.01%	0.00%	0.00%	0.00%
200,000	27.98%	13.29%	3.16%	0.50%	0.06%	0.01%	0.00%
400,000	4.15%	3.21%	1.25%	0.32%	0.06%	0.01%	0.00%
600,000	3.07%	3.30%	1.77%	0.64%	0.17%	0.04%	0.01%
800,000	2.28%	3.13%	2.15%	0.99%	0.34%	0.09%	0.02%
1,000,000	3.37%	5.65%	4.73%	2.64%	1.11%	0.37%	0.10%
1,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.7.: Homer and Clark (2003), Table 3.3.

TABLE 3.4

AGGREGATE LOSS & AGGREGATE ALLOCATED LOSS ADJUSTMENT EXPENSE JOINT DISTRIBUTION

Loss Amount	ALAE 0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	7.42%	0.23%	0.02%	0.00%	0.00%	0.00%	0.00%
200,000	4.33%	2.23%	0.59%	0.11%	0.02%	0.00%	0.00%
400,000	2.22%	2.10%	1.01%	0.33%	0.08%	0.02%	0.00%
600,000	1.41%	1.82%	1.20%	0.54%	0.18%	0.05%	0.01%
800,000	1.06%	1.71%	1.40%	0.77%	0.33%	0.11%	0.03%
1,000,000	1.11%	2.09%	1.98%	1.27%	0.62%	0.25%	0.08%
1,200,000	0.69%	1.59%	1.84%	1.42%	0.83%	0.39%	0.16%
1,400,000	0.40%	1.07%	1.45%	1.32%	0.90%	0.50%	0.23%
1,600,000	0.25%	0.75%	1.15%	1.19%	0.92%	0.57%	0.30%
1,800,000	0.16%	0.56%	0.95%	1.08%	0.93%	0.64%	0.37%
2,000,000	0.12%	0.44%	0.82%	1.02%	0.96%	0.72%	0.46%
2,200,000	0.07%	0.30%	0.61%	0.84%	0.87%	0.72%	0.50%
2,400,000	0.04%	0.19%	0.43%	0.65%	0.73%	0.66%	0.50%
2,600,000	0.03%	0.13%	0.31%	0.50%	0.60%	0.59%	0.48%
2,800,000	0.02%	0.09%	0.22%	0.38%	0.50%	0.52%	0.45%
3,000,000	0.01%	0.06%	0.16%	0.30%	0.41%	0.46%	0.42%
3,200,000	0.01%	0.04%	0.11%	0.22%	0.32%	0.38%	0.37%
3,400,000	0.00%	0.02%	0.07%	0.15%	0.24%	0.30%	0.32%
3,600,000	0.00%	0.01%	0.05%	0.11%	0.18%	0.24%	0.27%
3,800,000	0.00%	0.01%	0.03%	0.08%	0.13%	0.19%	0.22%
4,000,000	0.00%	0.01%	0.02%	0.05%	0.10%	0.14%	0.18%

Figure 6.8.: Homer and Clark (2003), Table 3.4.

TABLE 3.5

AGGREGATE LOSS CAPPED AT 2,000,000 & AGGREGATE ALLOCATED LOSS ADJUSTMENT EXPENSE JOINT DISTRIBUTION

Loss Amount	ALAE 0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	7.42%	0.23%	0.02%	0.00%	0.00%	0.00%	0.00%
200,000	4.33%	2.23%	0.59%	0.11%	0.02%	0.00%	0.00%
400,000	2.22%	2.10%	1.01%	0.33%	0.08%	0.02%	0.00%
600,000	1.41%	1.82%	1.20%	0.54%	0.18%	0.05%	0.01%
800,000	1.06%	1.71%	1.40%	0.77%	0.33%	0.11%	0.03%
1,000,000	1.11%	2.09%	1.98%	1.27%	0.62%	0.25%	0.08%
1,200,000	0.69%	1.59%	1.84%	1.42%	0.83%	0.39%	0.16%
1,400,000	0.40%	1.07%	1.45%	1.32%	0.90%	0.50%	0.23%
1,600,000	0.25%	0.75%	1.15%	1.19%	0.92%	0.57%	0.30%
1,800,000	0.16%	0.56%	0.95%	1.08%	0.93%	0.64%	0.37%
2,000,000	0.31%	1.31%	2.88%	4.41%	5.28%	5.31%	4.73%
2,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.9.: Homer and Clark (2003), Table 3.5.

TABLE 3.6

PROBABILITIES FOR LIMITED LOSS PLUS ALAE

Combined Loss + ALAE	Probability	Calculation
0	7.42%	= 7.42%
200,000	4.56%	= 4.33% + 0.23%
400,000	4.47%	= 2.22% + 2.23% + 0.02%
600,000	4.09%	= 1.41% + 2.10% + 0.59% + 0.00%
800,000	3.99%	= 1.06% + 1.82% + 1.01% + 0.11% + 0.00%
⋮	⋮	⋮

Figure 6.10.: Homer and Clark (2003), Table 3.6.

Table 6.10 records the grid and checks both marginal means, and Table 6.11 is the published block itself.

```
XS3 = [0, 200e3, 400e3, 600e3, 800e3, 1000e3]
YS3 = [0, 200e3, 400e3, 600e3, 800e3, 1000e3, 1200e3]
M3 = np.array([
    [.0839, .0147, .0013, .0001, .0000, .0000, .0000],
    [.2798, .1329, .0316, .0050, .0006, .0001, .0000],
    [.0415, .0321, .0125, .0032, .0006, .0001, .0000],
    [.0307, .0330, .0177, .0064, .0017, .0004, .0001],
    [.0228, .0313, .0215, .0099, .0034, .0009, .0002],
    [.0337, .0565, .0473, .0264, .0111, .0037, .0010]])
TARGET = np.array([.10, .45, .09, .09, .09, .18])       # Table 3.2
M3 = M3 * (TARGET / M3.sum(axis=1))[:, None]
```

```
def dbvsev(name, xs, ys, M, count, cv):
    rows = ' '.join('[' + ' '.join(f'{float(v):.12g}' for v in r) + ']'
                    for r in M)
    return (f'bivariate {name} {count} claims '
            f'dbvsev [{" ".join(f"{float(v):.0f}" for v in xs)}] '
            f'[{" ".join(f"{float(v):.0f}" for v in ys)}] [{rows}] '
            f'mixed gamma {cv}')

b3 = build(dbvsev('HC3', XS3, YS3, M3, 4, nb_cv(4, 8)), bs=(K, K), log2=(7, 6))
d3 = b3.density
GT(pd.DataFrame([{'grid': str(d3.shape), 'deficit': b3.deficit,
                  'loss mean': (d3.sum(axis=1) * b3.axis_xs[0]).sum(),
                  'paper 4 x 432,000': 4 * 432_000,
                  'ALAE mean': (d3.sum(axis=0) * b3.axis_xs[1]).sum()}]))
```

Table 6.10.: The section 3 bivariate grid, its mass deficit, and the loss and ALAE means.

grid	deficit	loss mean	paper 4 x 432,000	ALAE mean
(128, 64)	1.906p	1.728M	1,728,000	656,960

```
T34 = {0: [7.42, .23, .02, 0, 0, 0, 0],
       200_000: [4.33, 2.23, .59, .11, .02, 0, 0],
       400_000: [2.22, 2.10, 1.01, .33, .08, .02, 0],
       600_000: [1.41, 1.82, 1.20, .54, .18, .05, .01],
       800_000: [1.06, 1.71, 1.40, .77, .33, .11, .03],
       1_000_000: [1.11, 2.09, 1.98, 1.27, .62, .25, .08],
       1_200_000: [.69, 1.59, 1.84, 1.42, .83, .39, .16],
       1_400_000: [.40, 1.07, 1.45, 1.32, .90, .50, .23],
       1_600_000: [.25, .75, 1.15, 1.19, .92, .57, .30],
       1_800_000: [.16, .56, .95, 1.08, .93, .64, .37],
       2_000_000: [.12, .44, .82, 1.02, .96, .72, .46],
       2_200_000: [.07, .30, .61, .84, .87, .72, .50],
       2_400_000: [.04, .19, .43, .65, .73, .66, .50],
       2_600_000: [.03, .13, .31, .50, .60, .59, .48],
       2_800_000: [.02, .09, .22, .38, .50, .52, .45],
       3_000_000: [.01, .06, .16, .30, .41, .46, .42],
       3_200_000: [.01, .04, .11, .22, .32, .38, .37],
       3_400_000: [0, .02, .07, .15, .24, .30, .32],
       3_600_000: [0, .01, .05, .11, .18, .24, .27],
       3_800_000: [0, .01, .03, .08, .13, .19, .22],
       4_000_000: [0, .01, .02, .05, .10, .14, .18]}
got34 = np.array([100 * d3[int(round(r / K)), :7] for r in T34])
GT(pd.DataFrame(got34, index=[f'{r:,.0f}' for r in T34],
                columns=[f'{int(j * K):,.0f}' for j in range(7)]),
   float_format=2)
```

Table 6.11.: The aggregate joint distribution of capped loss and ALAE, the 21 × 7 block Table 3.4 prints, as percentages.

level_0	0	200,000	400,000	600,000	800,000	1,000,000	1,200,000
0	7.42	0.23	0.02	0.00	0.00	0.00	0.00
200,000	4.33	2.22	0.59	0.11	0.02	0.00	0.00
400,000	2.22	2.10	1.01	0.33	0.08	0.02	0.00
600,000	1.41	1.82	1.20	0.54	0.18	0.05	0.01
800,000	1.06	1.70	1.39	0.77	0.33	0.11	0.03
1,000,000	1.11	2.09	1.98	1.27	0.62	0.25	0.08
1,200,000	0.69	1.59	1.84	1.42	0.84	0.40	0.16
1,400,000	0.40	1.07	1.45	1.32	0.90	0.50	0.23
1,600,000	0.25	0.75	1.15	1.19	0.92	0.57	0.30
1,800,000	0.17	0.56	0.95	1.08	0.93	0.64	0.37
2,000,000	0.12	0.44	0.82	1.02	0.96	0.72	0.46
2,200,000	0.07	0.30	0.62	0.84	0.87	0.72	0.50
2,400,000	0.04	0.19	0.43	0.65	0.73	0.66	0.50
2,600,000	0.03	0.13	0.31	0.50	0.61	0.59	0.48
2,800,000	0.02	0.09	0.22	0.38	0.50	0.52	0.46
3,000,000	0.01	0.06	0.16	0.30	0.41	0.46	0.42
3,200,000	0.01	0.04	0.11	0.22	0.32	0.38	0.38
3,400,000	0.00	0.02	0.07	0.15	0.24	0.30	0.32
3,600,000	0.00	0.01	0.05	0.11	0.18	0.24	0.27
3,800,000	0.00	0.01	0.03	0.08	0.13	0.19	0.22
4,000,000	0.00	0.01	0.02	0.05	0.10	0.15	0.18

```
CAP = 2_000_000
kc = int(round(CAP / K))
d35 = np.vstack([d3[:kc], d3[kc:].sum(axis=0)[None, :]])
n = 5
tot3 = np.zeros(n)
for i in range(d35.shape[0]):
    for j in range(min(d35.shape[1], n)):
        if i + j < n:
            tot3[i + j] += d35[i, j]
GT(pd.DataFrame([
    {'check': 'Table 3.4, worst of the 21 x 7 published cells, points',
     'value': float(np.abs(got34 - np.array(list(T34.values()))).max())},
    {'check': 'Table 3.6, worst of the 5 published cells, points',
     'value': float(np.abs(100 * tot3
                           - np.array([7.42, 4.56, 4.47, 4.09, 3.99])).max())},
]), float_format=4)
```

Table 6.12.: Worst published cell error for Tables 3.4 and 3.6.

check	value
Table 3.4, worst of the 21 x 7 published cells, points	0.0061
Table 3.6, worst of the 5 published cells, points	0.0036

```
GT(pd.DataFrame({'loss (capped) + ALAE': [f'{i * K:,.0f}' for i in range(n)],
                 'aggregate %': 100 * tot3,
                 'Table 3.6 %': [7.42, 4.56, 4.47, 4.09, 3.99]}),
   float_format=2)
```

Table 6.13.: Capped loss plus ALAE combined, against the five cells Table 3.6 prints.

loss (capped) + ALAE	aggregate %	Table 3.6 %
0	7.42	7.42
200,000	4.56	4.56
400,000	4.47	4.47
600,000	4.09	4.09
800,000	3.99	3.99

Table 6.12 scores both, and Table 6.13 prints the combined distribution itself. The $2,000,000 general aggregate applies to indemnity only, so the cap is a row collapse on the loss axis with the ALAE axis untouched. That asymmetry is the whole reason the example needs a joint distribution: cap the sum and you have the wrong contract.

6.5. Section 4: dollars against counts

The third example puts aggregate small loss **dollars** on one axis and large loss **counts** on the other. The bivariate severity is degenerate by construction: a claim is either small, contributing dollars and no count, or large, contributing one count and no dollars. The inputs are Figure 6.11 and Figure 6.12, and the targets Figure 6.13 and Figure 6.14.

TABLE 4.1

SEVERITY DISTRIBUTION

Probability	Loss Amount
0.00%	0
43.80%	200,000
24.60%	400,000
13.80%	600,000
7.80%	800,000
10.00%	1,000,000
Average	431,200

Figure 6.11.: Homer and Clark (2003), Table 4.1.

TABLE 4.2

SINGLE CLAIM SMALL LOSS & LARGE COUNTS JOINT DISTRIBUTION

Small Loss	Large Loss Counts 0	1	2	3	4	5
0	0.00%	**10.00%**	0.00%	0.00%	0.00%	0.00%
200,000	**43.80%**	0.00%	0.00%	0.00%	0.00%	0.00%
400,000	**24.60%**	0.00%	0.00%	0.00%	0.00%	0.00%
600,000	**13.80%**	0.00%	0.00%	0.00%	0.00%	0.00%
800,000	**7.80%**	0.00%	0.00%	0.00%	0.00%	0.00%
1,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,200,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,400,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,600,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1,800,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2,000,000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 6.12.: Homer and Clark (2003), Table 4.2.

TABLE 4.3

AGGREGATE CLAIM SMALL LOSS & LARGE COUNTS
BIVARIATE DISTRIBUTION

Small Loss	Large Loss Counts 0	1	2	3	4	5	6
0	0.10%	0.05%	0.01%	0.00%	0.00%	0.00%	0.00%
200,000	0.21%	0.12%	0.04%	0.01%	0.00%	0.00%	0.00%
400,000	0.38%	0.22%	0.07%	0.02%	0.00%	0.00%	0.00%
600,000	0.58%	0.36%	0.12%	0.03%	0.01%	0.00%	0.00%
800,000	0.82%	0.53%	0.18%	0.05%	0.01%	0.00%	0.00%
1,000,000	1.07%	0.71%	0.26%	0.07%	0.01%	0.00%	0.00%
1,200,000	1.31%	0.91%	0.34%	0.09%	0.02%	0.00%	0.00%
1,400,000	1.54%	1.11%	0.43%	0.12%	0.03%	0.00%	0.00%
1,600,000	1.74%	1.30%	0.52%	0.15%	0.03%	0.01%	0.00%
1,800,000	1.90%	1.46%	0.60%	0.18%	0.04%	0.01%	0.00%
2,000,000	2.02%	1.60%	0.68%	0.20%	0.05%	0.01%	0.00%
2,200,000	2.09%	1.71%	0.75%	0.23%	0.06%	0.01%	0.00%
2,400,000	2.12%	1.78%	0.80%	0.25%	0.06%	0.01%	0.00%
2,600,000	2.11%	1.82%	0.84%	0.27%	0.07%	0.02%	0.00%
2,800,000	2.06%	1.83%	0.86%	0.29%	0.08%	0.02%	0.00%
3,000,000	1.98%	1.81%	0.87%	0.30%	0.08%	0.02%	0.00%
3,200,000	1.88%	1.75%	0.87%	0.30%	0.08%	0.02%	0.00%
3,400,000	1.76%	1.68%	0.85%	0.30%	0.09%	0.02%	0.00%
3,600,000	1.62%	1.59%	0.82%	0.30%	0.09%	0.02%	0.00%
3,800,000	1.48%	1.49%	0.79%	0.29%	0.09%	0.02%	0.00%
4,000,000	1.34%	1.38%	0.74%	0.28%	0.08%	0.02%	0.00%
4,200,000	1.20%	1.26%	0.70%	0.27%	0.08%	0.02%	0.00%
4,400,000	1.06%	1.14%	0.64%	0.25%	0.08%	0.02%	0.00%
4,600,000	0.94%	1.03%	0.59%	0.24%	0.08%	0.02%	0.00%
4,800,000	0.82%	0.91%	0.54%	0.22%	0.07%	0.02%	0.00%
5,000,000	0.71%	0.81%	0.48%	0.20%	0.07%	0.02%	0.00%

Figure 6.13.: Homer and Clark (2003), Table 4.3.

TABLE 4.4

CONDITIONAL DISTRIBUTIONS OF LARGE COUNTS GIVEN AGGREGATE SMALL LOSSES

Small Loss	Large Loss Counts 0	1	2	3	4	5	6
0	59.87%	29.94%	8.23%	1.65%	0.27%	0.04%	0.00%
200,000	56.88%	31.28%	9.39%	2.03%	0.36%	0.05%	0.01%
400,000	54.91%	32.07%	10.18%	2.33%	0.43%	0.07%	0.01%
600,000	53.26%	32.68%	10.87%	2.60%	0.50%	0.08%	0.01%
800,000	51.80%	33.17%	11.49%	2.85%	0.57%	0.10%	0.01%
1,000,000	50.37%	33.62%	12.11%	3.12%	0.64%	0.11%	0.02%
1,200,000	49.03%	34.01%	12.70%	3.39%	0.72%	0.13%	0.02%
1,400,000	47.77%	34.34%	13.26%	3.65%	0.80%	0.15%	0.02%
1,600,000	46.55%	34.63%	13.81%	3.92%	0.89%	0.17%	0.03%
1,800,000	45.38%	34.87%	14.34%	4.19%	0.97%	0.19%	0.03%
2,000,000	44.26%	35.09%	14.86%	4.47%	1.07%	0.22%	0.04%
2,200,000	43.17%	35.26%	15.37%	4.74%	1.16%	0.24%	0.04%
2,400,000	42.12%	35.41%	15.86%	5.02%	1.26%	0.27%	0.05%
2,600,000	41.10%	35.53%	16.34%	5.31%	1.37%	0.30%	0.06%
2,800,000	40.11%	35.62%	16.80%	5.59%	1.47%	0.33%	0.06%
3,000,000	39.16%	35.69%	17.25%	5.88%	1.58%	0.36%	0.07%
3,200,000	38.22%	35.73%	17.69%	6.17%	1.70%	0.39%	0.08%
3,400,000	37.32%	35.75%	18.12%	6.46%	1.82%	0.43%	0.09%
3,600,000	36.44%	35.75%	18.54%	6.75%	1.94%	0.47%	0.10%
3,800,000	35.58%	35.73%	18.94%	7.05%	2.07%	0.51%	0.11%
4,000,000	34.74%	35.69%	19.33%	7.34%	2.19%	0.55%	0.12%
4,200,000	33.93%	35.63%	19.71%	7.64%	2.33%	0.59%	0.13%
4,400,000	33.14%	35.56%	20.08%	7.94%	2.46%	0.64%	0.14%
4,600,000	32.37%	35.47%	20.44%	8.24%	2.61%	0.69%	0.16%
4,800,000	31.61%	35.36%	20.78%	8.54%	2.75%	0.74%	0.17%
5,000,000	30.88%	35.25%	21.12%	8.83%	2.90%	0.79%	0.19%

Figure 6.14.: Homer and Clark (2003), Table 4.4.

```
XS4 = [0, 200e3, 400e3, 600e3, 800e3]
YS4 = [0, 1, 2, 3, 4, 5]
M4 = np.array([[.000, .100, 0, 0, 0, 0],
               [.438, .000, 0, 0, 0, 0],
               [.246, .000, 0, 0, 0, 0],
               [.138, .000, 0, 0, 0, 0],
               [.078, .000, 0, 0, 0, 0]])
b4 = build(dbvsev('HC4', XS4, YS4, M4, 10, nb_cv(10, 20)),
           bs=(K, 1.0), log2=(7, 5))
d4 = b4.density
gx4, gy4 = b4.axis_xs
T43 = {0: [.10, .05, .01, 0, 0, 0, 0], 200_000: [.21, .12, .04, .01, 0, 0, 0],
       400_000: [.38, .22, .07, .02, 0, 0, 0],
       600_000: [.58, .36, .12, .03, .01, 0, 0],
       800_000: [.82, .53, .18, .05, .01, 0, 0],
       1_000_000: [1.07, .71, .26, .07, .01, 0, 0],
       2_000_000: [2.02, 1.60, .68, .20, .05, .01, 0],
       3_000_000: [1.98, 1.81, .87, .30, .08, .02, 0],
       4_000_000: [1.34, 1.38, .74, .28, .08, .02, 0],
       5_000_000: [.71, .81, .48, .20, .07, .02, 0]}
got43 = np.array([100 * d4[int(round(r / K)), :7] for r in T43])
GT(pd.DataFrame(got43, index=[f'{r:,.0f}' for r in T43],
```

```
                columns=[f'{j} large' for j in range(7)]), float_format=2)
```

Table 6.14.: The joint distribution of aggregate small loss dollars against large loss counts, the block Table 4.3 prints, as percentages.

level_0	0 large	1 large	2 large	3 large	4 large	5 large	6 large
0	0.10	0.05	0.01	0.00	0.00	0.00	0.00
200,000	0.21	0.12	0.04	0.01	0.00	0.00	0.00
400,000	0.38	0.22	0.07	0.02	0.00	0.00	0.00
600,000	0.58	0.36	0.12	0.03	0.01	0.00	0.00
800,000	0.82	0.53	0.18	0.05	0.01	0.00	0.00
1,000,000	1.07	0.71	0.26	0.07	0.01	0.00	0.00
2,000,000	2.02	1.60	0.68	0.20	0.05	0.01	0.00
3,000,000	1.98	1.81	0.87	0.30	0.08	0.02	0.00
4,000,000	1.34	1.38	0.74	0.28	0.08	0.02	0.00
5,000,000	0.71	0.81	0.48	0.20	0.07	0.02	0.00

Table 4.4 rescales each row of Table 6.14 to 100%, giving in Table 6.15 the conditional law of large claim counts given the aggregate small loss, which is what the simulation actually needs.

```
cond = d4 / d4.sum(axis=1, keepdims=True)
T44 = {0: [59.87, 29.94, 8.23, 1.65, .27, .04, 0],
       1_000_000: [50.37, 33.62, 12.11, 3.12, .64, .11, .02],
       3_000_000: [39.16, 35.69, 17.25, 5.88, 1.58, .36, .07],
       5_000_000: [30.88, 35.25, 21.12, 8.83, 2.90, .79, .19]}
got44 = np.array([100 * cond[int(round(r / K)), :7] for r in T44])
GT(pd.DataFrame(got44, index=[f'{r:,.0f}' for r in T44],
                columns=[f'{j} large' for j in range(7)]), float_format=2)
```

Table 6.15.: Large loss counts conditional on the aggregate small loss, against the four rows Table 4.4 prints.

level_0	0 large	1 large	2 large	3 large	4 large	5 large	6 large
0	59.87	29.94	8.23	1.65	0.27	0.04	0.00
1,000,000	50.37	33.62	12.11	3.12	0.64	0.11	0.02
3,000,000	39.16	35.69	17.25	5.88	1.58	0.36	0.07
5,000,000	30.88	35.25	21.12	8.83	2.90	0.79	0.19

```
GT(pd.DataFrame([
    {'check': 'Table 4.3, worst published cell, points',
     'value': float(np.abs(got43 - np.array(list(T43.values()))).max())},
    {'check': 'Table 4.4, worst published cell, points',
     'value': float(np.abs(got44 - np.array(list(T44.values()))).max())},
]), float_format=4)
```

Table 6.16.: Worst published cell error for Tables 4.3 and 4.4.

check	value
Table 4.3, worst published cell, points	0.0050
Table 4.4, worst published cell, points	0.0050

The paper's Appendix B derives the covariance of small and large aggregate losses in closed form,

$$\mathrm{Cov}(S, L) = p\mu_S(1-p)\mu_L(\sigma_N^2 - \mu_N).$$

Table 6.16 scores both tables. Since every large loss in this example is exactly \$1,000,000, the count axis converts to dollars by a scale factor and the identity can be checked against the joint directly, in Table 6.17.

```
p = 0.90
muS = (43.8 * 200 + 24.6 * 400 + 13.8 * 600 + 7.8 * 800) / 90 * 1e3
muL, sig2N, muN = 1e6, 20.0, 10.0
ES = (d4.sum(axis=1) * gx4).sum()
EL = (d4.sum(axis=0) * gy4).sum() * muL
ESL = (d4 * gx4[:, None] * (gy4[None, :] * muL)).sum()
GT(pd.DataFrame([
    {'source': 'Appendix B closed form',
     'Cov(S, L)': p * muS * (1 - p) * muL * (sig2N - muN)},
    {'source': 'from the joint density', 'Cov(S, L)': ESL - ES * EL},
]).set_index('source'), formatters={'Cov(S, L)': '{x:,.2f}'})
```

Table 6.17.: The Appendix B closed form covariance of small and large aggregate loss, against the same quantity taken from the joint density.

source	Cov(S, L)
Appendix B closed form	331,200,000,000.00
from the joint density	331,199,999,961.86

Table 6.17 agrees to better than two parts in ten billion, and the sign is positive because $\sigma_N^2 > \mu_N$ for the negative binomial. The paper's observation that the covariance vanishes under a Poisson count is immediate from the same formula, and is the reason its early DFA model got away with the independence assumption for as long as it did.

6.6. Comments

6.6.1. This is the paper `aggregate` was waiting for

Every construction in it maps onto a shipped DecL form, as Table 6.18 shows.

Table 6.18.: Each construction in the paper and the DecL form that expresses it.

Paper	DecL
Section 2, retained against ceded	`netceded agg ... occurrence net of 400000 xs 600000`, or `Aggregate.occ_bivariate`
Sections 3 and 4, a given joint severity	`bivariate ... dbvsev [xs] [ys] [[rows]]`
Different units on the two axes	`bs=(200000, 1)`
Appendix A, a bare two-dimensional convolution	`bivariate ... dfreq [2] dbvsev ...`

Section 2 is the sharpest of the three, because `aggregate` does not need Table 2.3 at all. The retained and ceded amounts are the net and ceded views of one occurrence layer, so the joint per-claim severity is the cession map pushed forward along the gross severity, and the object computes it. The paper spends a page constructing a matrix that is now a consequence of writing `occurrence net of 400000 xs 600000`.

The contrast with Jin et al. (2016), assessed on its own page, is worth drawing. That paper is dependence through a **common shock on the claim counts**, which `aggregate` cannot express. This one is dependence through a **joint severity under a shared count**, which is exactly what `aggregate`'s bivariate is. Jin and coauthors cite Clark and Homer for their FFT benchmark, so the two sit adjacent in the literature; they are on opposite sides of the feature boundary.

6.6.2. What the reproduction cost

Table 6.19 scores every published block at once.

```
def score(got, paper):
    p = np.array(paper if isinstance(paper, np.ndarray) else list(paper.values()))
    return (got.size, int((np.round(got, 2) != np.round(p, 2)).sum()),
            float(np.abs(got - p).max()))

rows = []
for tag, got, pap in [('Appendix A, 3x3 convolution', None, None),
                      ('Section 2, Table 2.4', got24, pap24),
                      ('Section 3, Table 3.4', got34, T34),
                      ('Section 4, Table 4.3', got43, T43),
                      ('Section 4, Table 4.4', got44, T44)]:
    if got is None:
        rows.append({'target': tag, 'published cells': 9,
                     'cells differing at 2 dp': 0, 'worst raw difference': 0.0})
    else:
        n, diff, worst = score(got, pap)
        rows.append({'target': tag, 'published cells': n,
                     'cells differing at 2 dp': diff,
                     'worst raw difference': worst})
GT(pd.DataFrame(rows), float_format=4)
```

Table 6.19.: Every published block, its cell count, how many differ at the two decimals printed, and the worst raw difference.

target	published cells	cells differing at 2 dp	worst raw difference
Appendix A, 3x3 convolution	9	0	0.0000
Section 2, Table 2.4	182	0	0.0050
Section 3, Table 3.4	147	11	0.0061
Section 4, Table 4.3	70	0	0.0050
Section 4, Table 4.4	28	0	0.0050

In Table 6.19, section 2 and section 4 reproduce **every** published cell at the two decimals printed, and the five section 2 statistics reproduce to the digit. Section 3 reproduces 136 of its 147 cells exactly and misses the other eleven by one unit in the last printed place.

Those eleven are the price of the input matrix. Table 3.3 is itself printed to two decimals, so each of its 42 entries carries up to 0.005% of unstated error, and a four-fold convolution compounds that. Sections 2 and 4 do not have the problem because their inputs are exact: section 2's severity matrix is derived from a five point severity by the cession map, and section 4's is a five point severity with a single 10% atom. Where the paper gives an exact input, `aggregate` returns its exact output; where the paper gives a rounded input, the rounding survives.

6.6.3. Grid sizing is the one thing to get right

The first attempt at sections 3 and 4 used a 2^6 loss axis and returned garbage: two thirds of the mass missing in section 4 and nine tenths in section 3, and rows that looked like the paper's but displaced several million dollars up the grid. The cause is that the pinned bucket size and log2 did not cover the measured window, so the window was clipped, and the object says so in Table 6.20.

```
small = build(dbvsev('HC4b', XS4, YS4, M4, 10, nb_cv(10, 20)),
              bs=(K, 1.0), log2=(6, 5))
GT(pd.DataFrame([
    {'log2 per axis': '(6, 5)', 'x reach': small.axis_xs[0][-1],
     'deficit': small.deficit},
    {'log2 per axis': '(7, 5)', 'x reach': gx4[-1], 'deficit': b4.deficit},
]))
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\bivariate.py:1907: DefectiveDistributionWarning:
bivariate HC4b: pinned (bs, log2) does not cover the measured window on an axis, so the joint
loses mass off the grid -- raise log2, coarsen bs, or unpin one of them. Further occurrences this
session are suppressed; each object records its own verdict in validation.
  bss, log2s, x_mins, his, clipped = self._size_axes(
```

Table 6.20.: The section 4 build at two loss axis sizes, showing the reach and the mass deficit each gives.

log2 per axis	x reach	deficit
(6, 5)	16,600,000	678.958m
(7, 5)	25,400,000	529.132f

A deficit near 1, as in the first row of Table 6.20, is not a subtle warning, and it is the first thing to read off a bivariate build. Doubling one axis fixed it. This is the two-dimensional version of the padding discussion in the paper's own Appendix C, where the error bound on the wrapped convolution is $\|R\| \le \sum_{j>M} \mathsf{P}(N = j)$: run out of room and the mass does not disappear, it wraps.

6.6.4. Two blemishes in the paper

Table 3.3 does not sum to one. Printed to two decimals its rows give 10, 45, 9, 9, 9 and 17.97 percent against the 18 percent that Table 3.2 requires, so the matrix as published carries 99.97% of the mass. Scaling each row onto its Table 3.2 marginal is the correction, and with it Table 3.6 reproduces exactly; normalizing the whole matrix instead leaves Table 3.6 off by 0.01 in three of five cells.

Table 2.6 is captioned "Loss & ALAE Probability Calculation" although it belongs to section 2 and concerns aggregate primary excess plus aggregate excess loss. The heading is a leftover from section 3.

7. Bear and Nemlick on adjustable treaty features

Reproducing Bear and Nemlick (1990), Table I

7.1. Context and problem

An excess of loss treaty is rarely a flat exchange of premium for cover. It carries provisions that make the money depend on the loss: an annual aggregate deductible, a limited number of reinstatements, a loss corridor the cedant takes back, a retrospective rating plan on the premium, a profit commission, a sliding scale commission. Bear and Nemlick (1990) is the standard treatment of the family, and its organizing observation is that all six are the same object. Each is a deterministic function of a realized loss quantity, so each is priced by pushing that function forward over the aggregate loss distribution. Nothing about the treaty changes except which line of the statement the function writes into.

That reduces pricing to one question: where does the aggregate loss distribution come from? The paper answers it twice. The **lognormal model** fits a lognormal to the aggregate mean and coefficient of variation and reads the answer out of a printed table of excess pure premium ratios. The **collective risk model** builds the distribution properly, by characteristic function inversion, using the Heckman-Meyers algorithm. The paper's purpose is to find out how much the cheap method costs, and its conclusion is that for these six treaties it costs very little.

Both models are what `aggregate` computes directly, and the six features are now six DecL clauses. This page reproduces the paper's Table I: the lognormal column by rebuilding the paper's own arithmetic, and the collective risk column by FFT. Table 7.1 is the whole mapping.

Table 7.1.: The six adjustable features and the DecL clause that carries each.

Treaty	Feature	The clause
I	aggregate deductible	`aggregate net of 360 xs 0`
II	three free reinstatements	`occurrence net of 700 xs 0 deposit 1172 reinstatements 3 free`
III	loss corridor	`corridor 100% po 100% xs 100%`
IV	retrospective rating	`retro basic 0 lcm 1.3333 min 360 max 1200 premium`
V	profit commission	`pc 25% after 20%`
VI	sliding scale commission	`slide 40% at 35% and 25% at 55% and 20% at 65%`

7.2. Paper solution and gold standard

Table I on page 75, Figure 7.6, is the target. It gives, for each treaty, the item the feature is priced for, under the lognormal model and under four runs of the collective risk model indexed by a contagion parameter c and a mixing parameter b. Table 7.2 transcribes it.

Table 7.2.: Bear and Nemlick (1990), Table I as printed.

Example	Unadjusted rate	Item compared	Lognormal	$c = 0$, $b = 0$	$c = .05$, $b = .05$	$c = .05$, $b = .10$	$c = .10$, $b = .10$
I Aggregate deductible	5.00%	adjusted rate	1.47%	1.58	1.68	1.73	1.77
II Limited reinstatement	25.00	adjusted rate	19.53	19.89	19.72	19.55	19.52
III Loss corridor	5.00	adjusted rate	4.02	3.67	3.71	3.74	3.73
IV Retro rating plan	5.00	expected rate after retro	5.02	5.20	5.18	5.14	5.14
V Profit commission		expected profit commission	8.37	8.24	8.50	8.69	8.75
VI Sliding scale commission		expected sliding scale commission	31.04	30.31	30.90	31.22	31.33

The two parameters are Heckman-Meyers uncertainty loadings. Appendix C defines c as the variance of a gamma variable with mean one multiplying the Poisson claim count mean, which is exactly `aggregate`'s gamma mixed Poisson: DecL `mixed gamma {sqrt(c)}`, since the clause takes the coefficient of variation of a mixing distribution whose mean is one. It defines b as the variance of an inverse gamma variable with mean one dividing the severity. There is no analog of b in DecL, so only the $c = 0$, $b = 0$ column is a like for like comparison. The other three are shown throughout, and the contagion only counterpart beside them.

Two more exhibits are needed, and both are reproducible in their own right. Appendix F Exhibit 1 and Appendix J Exhibit 1, Figure 7.1 and Figure 7.2, derive the aggregate coefficient of variation for the two subject books from a single parameter Pareto severity and an excess claim count variance to mean ratio. Appendix B prints the lognormal excess pure premium ratio tables the whole lognormal method reads from.

7.2.1. The two subject books

All amounts on this page are in $000.

Treaties I and IV sit on the same book: $160 excess of $40 per occurrence, subject premium $12,000 split $9,000 and $3,000 between two classes with layer loss costs of 4.00% and 3.00% of subject premium, and single parameter Pareto severity with shape q of 0.90 and 0.95.

Treaties II and V sit on the other: $700 excess of $300 per occurrence, subject premium $6,000 split equally between three classes with layer loss costs of 10%, 14% and 21%, and Pareto shapes 1.5, 1.3 and 1.1. Treaty V runs the same book for three years.

Treaty III is $400 excess of $100 on $10,000 of subject premium in three classes at 3.20%, 3.80% and 3.50%, all with $q = 1.1$. Treaty VI is $900 excess of $100 on $25,000 of subject premium, and the paper gives nothing about it beyond a 10% layer loss cost and a coefficient of variation of 0.485.

The single parameter Pareto with truncation k and shape q has $S(w) = (k/w)^q$ on $w \geq k$, so a loss excess of k is a two parameter Pareto with scale k and shape q: DecL `{k} * pareto {q} - {k}`, supported on $[0, \infty)$. Layered at the treaty limit it gives the excess claim size distribution Figure 7.1 and Figure 7.2 tabulate.

```
%xmode minimal
import warnings

import numpy as np
import pandas as pd
from aggregate import lognorm_fit, lognorm_lev
from bookstyle import GT, build

# Treaty II cedes its whole severity, so the net severity is a point mass at
# zero whose empirical variance lands a rounding below it and trips numpy on
# the square root. Cosmetic, and it reaches nothing on this page.
warnings.filterwarnings('ignore', 'invalid value encountered in sqrt')


def mixing_cv(vmr, claims):
    """Gamma mixing CV giving an excess claim count variance to mean ratio.

    A gamma mixed Poisson with mixing CV c and mean n has variance
    n + n^2 c^2, so the variance to mean ratio is 1 + n c^2.
    """
    return float(np.sqrt((vmr - 1) / claims))


def exhibit_1(port, printed):
    """Compare a book's per class parameters with the paper's Exhibit 1."""
    rows = [{'class': a.name, 'claims': a.n, 'severity mean': a.sev_m,
             'severity sd': a.sev_m * a.sev_cv, 'aggregate mean': a.actual_m,
             'aggregate sd': a.actual_m * a.actual_cv,
             'aggregate cv': a.actual_cv} for a in port]
    rows.append({'class': 'all classes', 'claims': sum(a.n for a in port),
                 'severity mean': np.nan, 'severity sd': np.nan,
                 'aggregate mean': port.est_m,
                 'aggregate sd': port.est_m * port.est_cv,
                 'aggregate cv': port.est_cv})
    got = pd.DataFrame(rows).set_index('class')
    out = pd.concat({'aggregate': got, 'paper': printed.reindex(got.index)},
                    axis=1).swaplevel(axis=1).sort_index(axis=1)
    return out[got.columns]
```

```
Exception reporting mode: Minimal
```

Appendix F Exhibit 1, Figure 7.1, in full. The excess claim count variance to mean ratios in line 8 come from ISO increased limits parameters and are the paper's own inputs, so they are given to the build rather than derived. Table 7.3 is the comparison.

APPENDIX F
EXHIBIT 1
DETERMINATION OF AGGREGATE LOSS DISTRIBUTION SPECIFICATION PARAMETERS FOR NONPROPORTIONAL LOSS SHARING PLANS

	Class of Business		
	Class 1	Class 2	All Classes Combined
(1) Actual or estimated subject premium for treaty period	9,000,000	3,000,000	12,000,000
(2) Expected layer loss cost for entire layer prior to the application of all forms of coinsurance (layer burning cost) [expressed as a percentage of subject premium]	4.0000%	3.0000%	3.7500%
(3) Expected losses for the entire reinsured layer for the treaty period [(1) × (2)]	360,000	90,000	450,000
(4) Single Parameter Pareto q values for severity distributions	0.900	0.950	
(5) Mean excess claim size in layer (μ_s)	69,848	67,039	69,267
(6) Standard deviation of excess claim sizes in layer (σ_s)	60,908	60,084	60,749
(7) Expected number of claims in layer prior to the application of nonproportional loss sharing provisions (μ_c) [(3)/(5)]	5.154	1.343	6.497
(8) Excess claim count variance-to-mean ratio (VMR_c)	1.032	1.067	1.039
(9) Standard deviation of distribution of aggregate losses in layer $[\sigma_s^2 \cdot \mu_c + (\mu_c \cdot VMR_c) \cdot \mu_s^2]^{1/2}$	212,298	106,228	237,391
(10) Coefficient of variation of distribution of aggregate losses in layer [(9)/(3)]	0.590	1.180	0.528
(11) Selected coefficient of variation of aggregate loss distribution for all classes combined			0.528

NOTES

Lines (5) and (6): The mean excess claim size and the standard deviation of the excess claim sizes are based on a Single Parameter Pareto distribution assumption with the parameter (q value) given in item (4). (See Appendix E for formulas.) The all classes combined mean excess claim size is an average of the individual class mean claim sizes, weighted on the expected excess claim counts on line (7). The all classes combined claim size standard deviation is computed as follows:

(A) For each class of business, calculate the sum of the squares of items (5) and (6), respectively.
(B) Take a weighted average of the sums in (A), using the expected excess claim counts on line (7) as weights.
(C) Subtract the square of the all classes combined mean excess claim size from the result in (B).
(D) Take the square root of the result in (C) to obtain the all classes combined excess claim size standard deviation.

Line (8): The individual class excess claim count variance-to-mean ratios are calculated using the ISO increased limits parameters and the formulas in Appendix D. The all classes combined excess claim count variance-to-mean ratio is an average of the individual class variance-to-mean ratios, weighted on the expected claim counts on line (7).

Line (9): The standard deviation of the aggregate loss distribution for all classes combined is obtained by summing the squares of the aggregate loss distribution standard deviations for the individual classes and then taking the square root of the result.

Figure 7.1.: Bear and Nemlick (1990), Appendix F, Exhibit 1.

```
book_1 = build(f'''
port BearNemlick.One
    agg Class1 9000 exposure at 0.04 rate 160 xs 0
        sev 40 * pareto 0.90 - 40 mixed gamma {mixing_cv(1.032, 5.154)}
    agg Class2 3000 exposure at 0.03 rate 160 xs 0
        sev 40 * pareto 0.95 - 40 mixed gamma {mixing_cv(1.067, 1.343)}
''')

printed_1 = pd.DataFrame(
    [[5.154, 69_848, 60_908, 360_000, 212_298, 0.590],
     [1.343, 67_039, 60_084,  90_000, 106_228, 1.180],
     [6.497, np.nan, np.nan, 450_000, 237_391, 0.528]],
    index=['Class1', 'Class2', 'all classes'],
    columns=['claims', 'severity mean', 'severity sd', 'aggregate mean',
             'aggregate sd', 'aggregate cv']) / [1, 1e3, 1e3, 1e3, 1e3, 1]
GT(exhibit_1(book_1, printed_1), float_format=3)
```

Table 7.3.: The first subject book, class by class, against Appendix F Exhibit 1.

	claims		severity mean		severity sd		aggregate mean		aggregate sd		aggregate cv	
class	**aggregate**	**paper**	**aggregate**	**paper**	**aggregate**	**paper**	**aggregate**	**paper**	**aggregate**	**paper**	**aggregate**	**paper**
Class1	5.154	5.154	69.848	69.848	60.908	60.908	360.000	360	212.298	212.298	0.590	0.590
Class2	1.343	1.343	67.039	67.039	60.084	60.084	90.000	90	106.227	106.228	1.180	1.180
all classes	6.497	6.497	—	—	—	—	450.000	450	237.391	237.391	0.528	0.528

Appendix J Exhibit 1, Figure 7.2, the same for the second book, compared in Table 7.4.

APPENDIX J

EXHIBIT 1

DETERMINATION OF AGGREGATE LOSS DISTRIBUTION
SPECIFICATION PARAMETERS FOR A SINGLE TREATY YEAR

	Class of Business			
	Class 1	Class 2	Class 3	All Classes Combined
(1) Actual or estimated subject premium for treaty period	2,000,000	2,000,000	2,000,000	6,000,000
(2) Expected layer loss cost for entire layer prior to the application of all forms of coinsurance (layer burning cost) [expressed as a percentage of subject premium]	10.0000%	14.0000%	21.0000%	15.0000%
(3) Expected losses for the entire reinsured layer for the treaty period [(1) × (2)]	200,000	280,000	420,000	900,000
(4) Single Parameter Pareto q values for severity distributions	1.500	1.300	1.100	
(5) Mean excess claim size in layer (μ_k)	271,366	303,155	340,296	310,897
(6) Standard deviation of excess claim sizes in layer (σ_k)	246,592	257,600	266,584	260,265
(7) Expected number of claims in layer prior to the application of nonproportional loss sharing provisions (μ_c) [(3)÷(5)]	0.737	0.924	1.234	2.895
(8) Excess claim count variance-to-mean ratio (VMR_i)	1.006	1.009	1.019	1.012
(9) Standard deviation of distribution of aggregate losses in layer $[\sigma_k^2 \cdot \mu_c + (\mu_k \cdot VMR_i) \cdot \mu_k^2]^{1/2}$	315,301	383,323	483,065	692,606
(10) Coefficient of variation of distribution of aggregate losses in layer [(9)÷(3)]	1.577	1.369	1.150	0.770

NOTES:

Lines (5) and (6): The mean excess claim size and the standard deviation of the excess claim sizes are based on a Single Parameter Pareto distribution assumption with the parameter (q value) given in item (4). (See Appendix E for formulas.) The all classes combined mean excess claim size is an average of the individual class mean claim sizes, weighted on the expected excess claim counts on line (7). The all classes combined claim size standard deviation is computed as follows:

(A) For each class of business, calculate the sum of the squares of items (5) and (6), respectively.
(B) Take a weighted average of the sums in (A), using the expected excess claim counts on line (7) as weights.
(C) Subtract the square of the all classes combined mean excess claim size from the result in (B).
(D) Take the square root of the result in (C) to obtain the all classes combined excess claim size standard deviation.

Line (8): The individual class excess claim count variance-to-mean ratios are calculated using the ISO increased limits parameters and the formulas in Appendix D. The all classes combined excess claim count variance-to-mean ratio is an average of the individual class variance-to-mean ratios, weighted on the expected claim counts on line (7).

Line (9): The standard deviation of the aggregate loss distribution for all classes combined is obtained by summing the squares of the aggregate loss distribution standard deviations for the individual classes and then taking the square root of the result.

Figure 7.2.: Bear and Nemlick (1990), Appendix J, Exhibit 1.

```
book_2 = build(f'''
port BearNemlick.Two
    agg Class1 2000 exposure at 0.10 rate 700 xs 0
        sev 300 * pareto 1.5 - 300 mixed gamma {mixing_cv(1.006, 0.737)}
    agg Class2 2000 exposure at 0.14 rate 700 xs 0
        sev 300 * pareto 1.3 - 300 mixed gamma {mixing_cv(1.009, 0.924)}
    agg Class3 2000 exposure at 0.21 rate 700 xs 0
        sev 300 * pareto 1.1 - 300 mixed gamma {mixing_cv(1.019, 1.234)}
''')

printed_2 = pd.DataFrame(
    [[0.737, 271_395, 246_596, 200_000, 315_301, 1.577],
     [0.924, 303_133, 257_600, 280_000, 383_323, 1.369],
     [1.234, 340_296, 266_584, 420_000, 483_065, 1.150],
     [2.895, np.nan, np.nan, 900_000, 692_606, 0.770]],
    index=['Class1', 'Class2', 'Class3', 'all classes'],
    columns=['claims', 'severity mean', 'severity sd', 'aggregate mean',
             'aggregate sd', 'aggregate cv']) / [1, 1e3, 1e3, 1e3, 1e3, 1]
GT(exhibit_1(book_2, printed_2), float_format=3)
```

Table 7.4.: The second subject book, class by class, against Appendix J Exhibit 1.

	claims		severity mean		severity sd		aggregate mean		aggregate sd		aggregate cv	
class	aggregate	paper	aggregate	paper	aggregate	paper	aggregate	paper	aggregate	paper	aggregate	paper
Class1	0.737	0.737	271.366	271.395	246.592	246.596	200.000	200	315.301	315.301	1.577	1.577
Class2	0.924	0.924	303.155	303.133	257.600	257.600	280.000	280	383.323	383.323	1.369	1.369
Class3	1.234	1.234	340.296	340.296	266.584	266.584	420.000	420	483.066	483.065	1.150	1.150
all classes	2.895	2.895	—	—	—	—	900.000	900	692.606	692.606	0.770	0.770

Across Table 7.3 and Table 7.4, every claim count, every excess claim size moment and every aggre-

gate standard deviation comes back at the paper's printed precision, and with it the two coefficients of variation the lognormal method needs, 0.528 and 0.770. The two severity means in the second book differ in the fifth figure, 271.37 against 271.395 and 303.15 against 303.133, which is rounding in Appendix J Exhibit 1 rather than a difference of model.

7.2.2. The lognormal tables

The lognormal method needs the excess pure premium ratio $P_2(r)$, the expected loss above an attachment at entry ratio r divided by the expected loss, for a unit mean lognormal with a given coefficient of variation. The paper prints it in Tables B1 to B3, Figure 7.3 and Figure 7.4, and interpolates linearly in both arguments. Reproducing Table I therefore means reproducing the interpolation, not the exact lognormal: the two differ by up to seven basis points at the entry ratios these treaties use, which is enough to move a printed answer. Table 7.5 checks the transcribed cells.

TABLE B1

EXCESS PURE PREMIUM RATIOS
LOGNORMAL MODEL

Entry Ratio	Coefficient of Variation .1	.2	.3	.4	.5
0	1.000	1.000	1.000	1.000	1.000
.1	.900	.900	.900	.900	.900
.2	.800	.800	.800	.800	.800
.3	.700	.700	.700	.700	.700
.4	.600	.600	.600	.601	.603
.5	.500	.500	.501	.504	.510
.6	.400	.400	.404	.413	.426
.7	.300	.302	.313	.331	.351
.8	.200	.211	.234	.260	.286
.9	.107	.135	.168	.200	.232
1.0	.040	.079	.117	.153	.187
1.1	.009	.042	.079	.115	.150
1.2	.001	.021	.052	.086	.120
1.3	.000	.010	.034	.064	.096
1.4		.004	.022	.048	.077
1.5		.002	.014	.035	.062
1.6		.001	.009	.026	.049
1.7		.000	.005	.019	.040
1.8			.003	.014	.032
1.9			.002	.010	.026
2.0			.001	.008	.021
2.2			.000	.004	.014
2.4				.002	.009
2.6				.001	.006
2.8				.001	.004
3.0				.000	.003

The Coefficient of Variation is the ratio of the standard deviation to the mean of the assumed lognormal distribution. The Entry Ratio is the ratio of the attachment to the mean. The Excess Pure Premium Ratios are ratios of excess pure premiums to the mean (i.e., ratios of expected excess losses to the total expected loss).

Figure 7.3.: Bear and Nemlick (1990), Appendix B, Table B1.

TABLE B2

EXCESS PURE PREMIUM RATIOS
LOGNORMAL MODEL

Entry Ratio	Coefficient of Variation .6	.7	.8	.9	1.0
0	1.000	1.000	1.000	1.000	1.000
.1	.900	.900	.900	.900	.900
.2	.800	.800	.801	.802	.804
.3	.702	.704	.707	.710	.714
.4	.607	.613	.619	.627	.634
.5	.519	.530	.541	.552	.563
.6	.441	.456	.472	.487	.502
.7	.371	.392	.412	.430	.447
.8	.312	.336	.359	.381	.400
.9	.261	.289	.314	.337	.359
1.0	.218	.248	.275	.300	.323
1.1	.183	.213	.241	.267	.291
1.2	.153	.183	.212	.239	.263
1.3	.128	.158	.187	.213	.238
1.4	.107	.137	.165	.192	.216
1.5	.090	.118	.146	.172	.197
1.6	.076	.103	.130	.155	.180
1.7	.064	.089	.115	.140	.164
1.8	.054	.078	.103	.127	.150
1.9	.046	.068	.092	.115	.138
2.0	.039	.060	.082	.105	.127
2.2	.028	.046	.066	.087	.108
2.4	.020	.036	.053	.073	.092
2.6	.015	.028	.044	.061	.079
2.8	.011	.022	.036	.052	.068
3.0	.008	.017	.030	.044	.059
3.2	.006	.014	.025	.037	.052
3.4	.005	.011	.021	.032	.045
3.6	.004	.009	.017	.028	.040
3.8	.003	.007	.015	.024	.035
4.0	.002	.006	.012	.021	.031
5.0	.001	.001	.006	.011	.018
10.0	.000	.000	.000	.001	.002

The Coefficient of Variation is the ratio of the standard deviation to the mean of the assumed lognormal distribution. The Entry Ratio is the ratio of the attachment to the mean. The Excess Pure Premium Ratios are ratios of excess pure premiums to the mean (i.e., ratios of expected excess losses to the total expected loss).

Figure 7.4.: Bear and Nemlick (1990), Appendix B, Table B2.

```python
# The cells of Tables B1 and B2 the six exhibits reach into, printed excess pure
# premium ratio by entry ratio then coefficient of variation.
B = {
    0.0: {0.4: 1.000, 0.5: 1.000},
    0.6: {0.5: 0.426, 0.6: 0.441},
    0.7: {0.4: 0.331, 0.5: 0.351},
    0.8: {0.5: 0.286, 0.6: 0.312},
    1.0: {0.9: 0.300, 1.0: 0.323},
    1.1: {0.4: 0.115, 0.5: 0.150},
    1.3: {0.4: 0.064, 0.5: 0.096},
    1.6: {0.4: 0.026, 0.5: 0.049},
    1.7: {0.4: 0.019, 0.5: 0.040},
    2.0: {0.5: 0.021, 0.6: 0.039, 0.9: 0.105, 1.0: 0.127},
    3.0: {0.7: 0.017, 0.8: 0.030},
    3.2: {0.7: 0.014, 0.8: 0.025},
}


def _interp(row, cv):
    lo, hi = max(c for c in row if c <= cv), min(c for c in row if c >= cv)
    if lo == hi:
        return row[lo]
    return row[lo] + (cv - lo) / (hi - lo) * (row[hi] - row[lo])


def charge(cv, r):
```

```
    """Insurance charge P2(r), by linear interpolation in Tables B1 to B3."""
    lo = max(x for x in B if x <= r)
    hi = min(x for x in B if x >= r)
    if lo == hi:
        return _interp(B[lo], cv)
    a, b = _interp(B[lo], cv), _interp(B[hi], cv)
    return a + (r - lo) / (hi - lo) * (b - a)


def exact_charge(cv, r):
    """P2(r) for the lognormal itself, with no table in the way."""
    if r == 0:
        return 1.0
    mu, sigma = lognorm_fit(1, cv)
    return 1 - lognorm_lev(mu, sigma, 1, r)


cells = pd.DataFrame([{'entry ratio': r, 'cv': cv, 'printed': v,
                       'exact lognormal': exact_charge(cv, r),
                       'difference': v - exact_charge(cv, r)}
                      for r, row in sorted(B.items()) for cv, v in sorted(row.items())])
GT(cells, float_format=5)
```

Table 7.5.: Every cell of Tables B1 and B2 the six exhibits reach into, against the exact lognormal.

entry ratio	cv	printed	exact lognormal	difference
0.00000	0.40000	1.00000	1.00000	0.00000
0.00000	0.50000	1.00000	1.00000	0.00000
0.60000	0.50000	0.42600	0.42558	0.00042
0.60000	0.60000	0.44100	0.44051	0.00049
0.70000	0.40000	0.33100	0.33052	0.00048
0.70000	0.50000	0.35100	0.35057	0.00043
0.80000	0.50000	0.28600	0.28602	-0.00002
0.80000	0.60000	0.31200	0.31180	0.00020
1.00000	0.90000	0.30000	0.29987	0.00013
1.00000	1.00000	0.32300	0.32279	0.00021
1.10000	0.40000	0.11500	0.11520	-0.00020
1.10000	0.50000	0.15000	0.14995	0.00005
1.30000	0.40000	0.06400	0.06413	-0.00013
1.30000	0.50000	0.09600	0.09619	-0.00019
1.60000	0.40000	0.02600	0.02591	0.00009
1.60000	0.50000	0.04900	0.04938	-0.00038
1.70000	0.40000	0.01900	0.01911	-0.00011
1.70000	0.50000	0.04000	0.03962	0.00038
2.00000	0.50000	0.02100	0.02066	0.00034
2.00000	0.60000	0.03900	0.03864	0.00036
2.00000	0.90000	0.10500	0.10458	0.00042
2.00000	1.00000	0.12700	0.12688	0.00012
3.00000	0.70000	0.01700	0.01748	-0.00048
3.00000	0.80000	0.03000	0.02958	0.00042
3.20000	0.70000	0.01400	0.01396	0.00004
3.20000	0.80000	0.02500	0.02458	0.00042

In Table 7.5 no transcribed cell is more than 5 in the fourth decimal from the exact lognormal, which is the rounding of a three decimal table, so the transcription is sound and the B tables are what they claim to be.

7.3. The `aggregate` reproduction

Each treaty is one program. The subject book is an `agg`, the feature is a clause on it, and the answer is a mean read off the resulting `PnL`. Every treaty is built at all three contagion levels, so the functions below take the frequency clause as an argument.

Five of the six use `xpnl`, the walk face, which books each cession as its own step and so gives the ceded recovery, the ceded premium and any commission as separate legs. Treaty IV uses `pnl`, the consolidated net view, because a retro plan varies the gross premium and there is no cession to walk.

```
FREQ = {'c = 0': 'poisson',
        'c = .05': 'mixed gamma 0.05**0.5',
        'c = .10': 'mixed gamma 0.1**0.5'}


def leg(pnl, label):
    """One ledger leg's expected value, off the PnL's own economic_df."""
    return float(pnl.economic_df.xs(label, level='Label').iloc[0]['EX'])
```

7.3.1. Treaty I: aggregate deductible

The treaty carries an annual aggregate deductible of 3% of subject premium, $360 against $450 of expected layer loss, an entry ratio of 0.800. The rate is the loss cost surviving the deductible, loaded 100/75.

An aggregate deductible is an aggregate layer the cedant keeps, so the recovery is the aggregate net of it. Table 7.6 prices it.

```
BOOK_1 = ('agg BN.One [9000 3000] exposure at [0.04 0.03] rate 160 xs 0 '
          'sev 40 * pareto [0.90 0.95] - 40 {freq} ')


def treaty_1(freq):
    gross = build(BOOK_1.format(freq=freq))
    net = build(BOOK_1.format(freq=freq) + 'aggregate net of 360 xs 0')
    return {'cv': gross.actual_cv, 'charge': net.est_m / gross.actual_m,
            'key': 100 / 75 * 3.75 * net.est_m / gross.actual_m}


one = treaty_1(FREQ['c = 0'])
GT(pd.DataFrame([{'insurance charge, aggregate': one['charge'],
                  'insurance charge, lognormal': charge(0.528, 360 / 450),
                  'adjusted rate %, aggregate': one['key'],
                  'adjusted rate %, paper c = 0': 1.58,
                  'adjusted rate %, paper lognormal': 1.47}]), float_format=4)
```

Table 7.6.: Treaty I, the insurance charge and adjusted rate under both models, against Table I.

insurance charge, aggregate	insurance charge, lognormal	adjusted rate %, aggregate	adjusted rate %, paper c = 0	adjusted rate %, paper lognormal
0.3151	0.2933	1.5755	1.5800	1.4700

The paper's Appendix F Exhibit 2 puts the insurance charge at 29.33% and the indicated rate at 1.4664%, against 1.47% printed in Table I. In Table 7.6 the exact distribution puts the charge two and a fifth points higher, which is the whole story of the treaty: the lognormal fitted to a mean of 450 and a coefficient of variation of 0.528 is more skewed than the real aggregate, so it leaves too little mass in the body and overstates what a deductible at 0.8 times the mean eliminates.

7.3.2. Treaty II: three free reinstatements

Three free reinstatements plus the original limit give total annual capacity of $4 \times 700 = 2800$, against \$900 of expected layer loss, an entry ratio of 3.111. Then 20% of layer losses are coinsured without a share of premium, and the result is loaded 100/60.

This is where the reinstatement clause earns its place. The treaty language is three free reinstatements, and that is what goes in the program. The annual cap follows from it rather than being computed by hand and entered as an aggregate limit.

The schedule needs a base premium to price reinstatements against, and the `deposit` of \$1,172 is the paper's own indicated treaty premium, 19.53% of \$6,000. Free reinstatements charge nothing, so it does not reach the recovery. Table 7.7 is the resulting ledger.

```
BOOK_2 = ('agg BN.Two [2000 2000 2000] exposure at [0.10 0.14 0.21] rate 700 xs 0 '
          'sev 300 * pareto [1.5 1.3 1.1] - 300 '
          'occurrence net of 700 xs 0 deposit 1172 reinstatements 3 free '
          'as "Layer" {freq} ')


def treaty_2(freq):
    x = build('xpnl BN.TreatyII 1172 premium less ' + BOOK_2.format(freq=freq))
    recovery = leg(x, 'Layer recovery')
    composite = 1 - 0.80 * recovery / x.engine.actual_m
    return {'cv': x.engine.actual_cv, 'recovery': recovery,
            'charge': 1 - recovery / x.engine.actual_m,
            'key': 100 / 60 * 15 * (1 - composite), 'x': x}


two = treaty_2(FREQ['c = 0'])
print(two['x'].engine.reinstatement_terms)
```

```
ReinstatementTerms(limit=700, rates=(0.0, 0.0, 0.0), deposit=1172, rol=1.674, m=3, Y=2800)
```

```
GT(two['x'].economic_df[['EX', 'SD', 'CV']], float_format=3)
```

Table 7.7.: Treaty II, the P&L legs of the layer with three free reinstatements.

Step	*Side*	*Label*	EX	SD	CV
Gross	Consideration	Premium	1,172.000	0.000	0.000
	Obligation	Loss	-900.000	689.879	-0.767
	Margin	Gross	272.000	689.879	2.536
Layer	Consideration	Layer premium	-1,172.000	0.000	0.000
	Obligation	Layer recovery	894.878	672.732	0.752
	Margin	Total	-277.122	672.732	-2.428
		Net	-5.122	62.576	-12.218
All	Consideration	Net	0.000	0.000	—
	Obligation	Net	-5.122	62.576	-12.218
	Margin	Net	-5.122	62.576	-12.218
		Impact	-277.122	672.732	-2.428

The severity is already the loss to the layer, so the occurrence clause `700 xs 0` is the identity on each claim and the reinstatement schedule supplies the only binding term, the annual cap. The paper's own reading of the same provision is an aggregate limit of 2800, and Table 7.8 shows the two agree.

```
alt = build('agg BN.TwoAlt [2000 2000 2000] exposure at [0.10 0.14 0.21] rate '
            '700 xs 0 sev 300 * pareto [1.5 1.3 1.1] - 300 poisson '
            'aggregate ceded to 2800 xs 0')
GT(pd.DataFrame([{'reinstatements 3 free': two['recovery'],
                  'aggregate ceded to 2800 xs 0': alt.est_m,
                  'relative difference': abs(two['recovery'] / alt.est_m - 1),
                  'insurance charge, aggregate': two['charge'],
                  'insurance charge, lognormal': charge(0.770, 2800 / 900),
                  'adjusted rate %, aggregate': two['key'],
                  'adjusted rate %, paper c = 0': 19.89}]), float_format=6)
```

Table 7.8.: Treaty II, the reinstatement schedule against the equivalent aggregate limit, and the adjusted rate under both models.

reinstatements 3 free	aggregate ceded to 2800 xs 0	relative difference	insurance charge, aggregate	insurance charge, lognormal	adjusted rate %, aggregate	adjusted rate %, paper c = 0
894.878258	894.880498	0.000003	0.005691	0.023656	19.886184	19.890000

7.3.3. Treaty III: loss corridor

The reinsurer stops paying once layer losses reach 100% of expected and resumes above 200%. Expressed as entry ratios the band is one mean wide attaching at one mean, and the cedant takes all of it. Writing the base ceded premium as the expected layer loss of $350 makes the ceded loss ratio the entry ratio, so the corridor clause reads exactly as the slip does.

A corridor decorates an aggregate layer, and here the whole aggregate is ceded, so the layer is $10,000 excess of nothing: a limit far above anything the aggregate reaches, present only to give the corridor something to sit on. Table 7.9 prices it.

```
BOOK_3 = ('agg BN.Three [4500 4500 1000] exposure at [0.032 0.038 0.035] rate '
          '400 xs 0 sev 100 * pareto 1.1 - 100 {freq} '
          'aggregate net of 10000 xs 0 deposit 350 '
          'corridor 100% po 100% xs 100% as "Corridor" ')


def treaty_3(freq):
    x = build('xpnl BN.TreatyIII 10000 premium less ' + BOOK_3.format(freq=freq))
    recovery = leg(x, 'Corridor recovery')
    return {'cv': x.engine.actual_cv, 'recovery': recovery,
            'eliminated': 1 - recovery / 350,
            'key': 100 / 70 * 100 * recovery / 10000}


three = treaty_3(FREQ['c = 0'])
GT(pd.DataFrame([{
    'expected layer loss': 350.0,
    'after corridor, aggregate': three['recovery'],
    'eliminated, aggregate': three['eliminated'],
    'eliminated, lognormal': charge(0.905, 1.0) - charge(0.905, 2.0),
    'adjusted rate %, aggregate': three['key'],
    'adjusted rate %, paper c = 0': 3.67}]), float_format=4)
```

Table 7.9.: Treaty III, the loss eliminated by the corridor under both models, and the adjusted rate against Table I.

expected layer loss	after corridor, aggregate	eliminated, aggregate	eliminated, lognormal	adjusted rate %, aggregate	adjusted rate %, paper c = 0
350	256.8761	0.2661	0.1950	3.6697	3.6700

The corridor reads a loss ratio, so its parameterization depends on what the ceded premium is called. Restating the same band against the indicated treaty premium, which is a different number, has to give the same recovery, and in Table 7.10 it does.

```
premium = 100 / 70 * three['recovery']
band = 350 / premium
restated = build(
    f'xpnl BN.TreatyIIIb 10000 premium less '
    + BOOK_3.format(freq=FREQ['c = 0'])
    .replace('deposit 350', f'deposit {premium}')
    .replace('po 100% xs 100%', f'po {band} xs {band}'))
GT(pd.DataFrame([{'indicated treaty premium': premium,
                  'band in loss ratio points': band,
                  'recovery, expected loss basis': three['recovery'],
                  'recovery, premium basis': leg(restated, 'Corridor recovery')}]),
   float_format=6)
```

Table 7.10.: Treaty III's corridor recovery on an expected loss basis and on a premium basis, which must agree.

indicated treaty premium	band in loss ratio points	recovery, expected loss basis	recovery, premium basis
366.965864	0.953767	256.876105	256.876105

7.3.4. Treaty IV: retrospective rating

The same book as Treaty I, flat rated at 5%, with a one year retro plan: the adjusted premium is 100/75 times incurred layer loss, subject to a maximum of 10% and a minimum of 3% of $12,000 subject premium. There is no basic premium and no flat margin, so the collar is the whole plan.

Retro varies the account premium rather than a cession, so in DecL it replaces the fixed premium at the head of the program. Table 7.11 is the resulting ledger.

```
def treaty_4(freq):
    p = build('pnl BN.TreatyIV '
              'retro basic 0 lcm 1.3333333333 min 360 max 1200 premium '
              'less ' + BOOK_1.format(freq=freq).replace('BN.One', 'BN.Four'))
    return {'cv': p.engine.actual_cv, 'premium': leg(p, 'Premium'),
            'key': 100 * leg(p, 'Premium') / 12000, 'p': p}


four = treaty_4(FREQ['c = 0'])
GT(four['p'].economic_df[['EX', 'SD', 'CV']], float_format=3)
```

Table 7.11.: Treaty IV, the P&L legs under the retrospective rating plan.

Side	*Label*	EX	SD	CV
Consideration	Premium	624.510	253.880	0.407
Obligation	Loss	-450.000	234.831	-0.522
Margin	Total	174.510	62.482	0.358

In Table 7.11 the premium leg is now a distribution with a standard deviation, which is the point of a retro plan, and its mean divided by subject premium is the expected rate after all retro adjustments, in Table 7.12.

```
r_max, r_min = (10 / (100 / 75)) / 3.75, (3 / (100 / 75)) / 3.75
net_charge = charge(0.528, r_max) - (r_min + charge(0.528, r_min) - 1)
GT(pd.DataFrame([{
    'entry ratio at maximum': r_max, 'entry ratio at minimum': r_min,
    'net insurance charge, lognormal': net_charge,
    'expected rate %, lognormal': 100 / 75 * 3.75 * (1 - net_charge),
    'expected rate %, aggregate': four['key'],
    'expected rate %, paper c = 0': 5.20,
    'off balance factor, aggregate': 5.0 / four['key']}]), float_format=4)
```

Table 7.12.: Treaty IV, the collar as entry ratios, the net insurance charge, and the expected rate after retro under both models.

entry ratio at maximum	entry ratio at minimum	net insurance charge, lognormal	expected rate %, lognormal	expected rate %, aggregate	expected rate %, paper c = 0	off balance factor, aggregate
2	0.6000	-0.0042	5.0208	5.2042	5.2000	0.9608

7.3.5. Treaty V: three year profit commission

The second book run for three years, $18,000 of subject premium, treaty rate 25%, so $4,500 of treaty premium against $2,700 of expected layer loss. A 20% coinsurance leaves the reinsurer with $2,160, an expected ceded loss ratio of 48%. The profit commission returns 25% of the reinsurer's profit after a 20% overhead allowance, a put on the ceded loss ratio struck at 80%.

The coinsurance is a share of the loss with no corresponding share of premium. In DecL that is a placement: premium clauses are quoted at 100% and scaled by the fraction placed, so a slip premium of $4,500 on an 80% placement is a `deposit` of $5,625. The layer it is placed on is again nominal, wide enough to reach everything the three year aggregate does. Table 7.13 is the resulting ledger.

```
BOOK_5 = ('agg BN.Five [6000 6000 6000] exposure at [0.10 0.14 0.21] rate '
          '700 xs 0 sev 300 * pareto [1.5 1.3 1.1] - 300 {freq} '
          'aggregate net of 80% po 100000 xs 0 deposit 5625 '
          'pc 25% after 20% as "PC layer" ')


def treaty_5(freq):
    x = build('xpnl BN.TreatyV 18000 premium less ' + BOOK_5.format(freq=freq))
    return {'cv': x.engine.actual_cv, 'pc': leg(x, 'profit commission'),
            'key': 100 * leg(x, 'profit commission') / 4500, 'x': x}


five = treaty_5(FREQ['c = 0'])
GT(five['x'].economic_df[['EX', 'SD', 'CV']], float_format=3)
```

Table 7.13.: Treaty V, the P&L legs of the three year book carrying the profit commission.

Step	*Side*	*Label*	EX	SD	CV
Gross	Consideration	Premium	18,000.000	0.000	0.000
	Obligation	Loss	-2,700.000	1,194.862	-0.443
	Margin	Gross	15,300.000	1,194.862	0.078
PC layer	Consideration	PC layer premium	-4,500.000	0.000	0.000
	Obligation	PC layer recovery	2,160.000	955.889	0.443
		profit commission	370.725	215.746	0.582
		Total	2,530.725	745.716	0.295
	Margin	Total	-1,969.275	745.716	-0.379
		Net	13,330.725	452.437	0.034
All	Consideration	Net	13,500.000	0.000	0.000
	Obligation	Net	-169.275	452.437	-2.673
	Margin	Net	13,330.725	452.437	0.034
		Impact	-1,969.275	745.716	-0.379

```
elr = 0.15 * 0.80 / 0.25
GT(pd.DataFrame([{
    'expected ceded loss ratio': elr,
    'entry ratio at breakeven': 0.80 / elr,
    'net insurance charge, lognormal': charge(0.444, 0.80 / elr),
    'simplistic view %': 100 * 0.25 * (1 - elr - 0.20),
    'expected pc %, lognormal': 100 * 0.25 * (1 - elr * (1 - charge(0.444, 0.80 / elr)) - 0.20),
    'expected pc %, aggregate': five['key'],
    'expected pc %, paper c = 0': 8.24}]), float_format=4)
```

Table 7.14.: Treaty V, the expected profit commission under both models and under the simplistic view, against Table I.

expected ceded loss ratio	entry ratio at breakeven	net insurance charge, lognormal	simplistic view %	expected pc %, lognormal	expected pc %, aggregate	expected pc %, paper c = 0
0.4800	1.6667	0.0309	8	8.3704	8.2383	8.2400

In Table 7.14 the simplistic view plugs the expected loss ratio into the formula and gets 8.00%. Both models return more, because the profit commission is a positive part and therefore convex, so its expected value exceeds its value at the mean. The paper calls the difference the actuarial view; it is Jensen's inequality, and it is a property of the whole distribution rather than of any moment of it.

7.3.6. Treaty VI: sliding scale commission

$900 excess of $100 on $25,000 of subject premium, layer loss cost 10%, treaty rate 20%, so an expected treaty loss ratio of 50%. The commission is 20% at a loss ratio of 65% and above, slides up half a point per point down to 25% at 55%, then three quarters of a point per point up to 40% at 35%, and is flat at 40% below that. The plan is exactly the anchor form the `slide` clause takes.

Nothing else about the book is published, only its coefficient of variation of 0.485. The Pareto shape is therefore a free parameter, fixed here by solving for the value that reproduces the stated coefficient of variation under Poisson counts. Table 7.15 is the resulting ledger.

```
BOOK_6 = ('agg BN.Six 25000 exposure at 0.10 rate 900 xs 0 '
          'sev 100 * pareto {q} - 100 {freq} '
          'aggregate net of 100000 xs 0 deposit 5000 '
          'slide 40% at 35% and 25% at 55% and 20% at 65% as "Slide" ')
Q6 = 0.98


def treaty_6(freq, q=Q6):
    x = build('xpnl BN.TreatyVI 25000 premium less '
              + BOOK_6.format(freq=freq, q=q))
    return {'cv': x.engine.actual_cv, 'slide': leg(x, 'sliding commission'),
            'key': 100 * leg(x, 'sliding commission') / 5000, 'x': x}


six = treaty_6(FREQ['c = 0'])
GT(six['x'].economic_df[['EX', 'SD', 'CV']], float_format=3)
```

Table 7.15.: Treaty VI, the P&L legs of the book carrying the sliding scale commission.

Step	*Side*	*Label*	EX	SD	CV
Gross	Consideration	Premium	25,000.000	0.000	0.000
	Obligation	Loss	-2,499.999	1,212.545	-0.485
	Margin	Gross	22,500.001	1,212.545	0.054
Slide	Consideration	Slide premium	-5,000.000	0.000	0.000
	Obligation	Slide recovery	2,499.999	1,212.545	0.485
		sliding commission	1,515.866	417.931	0.276
		Total	4,015.865	856.660	0.213
	Margin	Total	-984.135	856.660	-0.870
		Net	21,515.866	417.931	0.019
All	Consideration	Net	20,000.000	0.000	0.000
	Obligation	Net	1,515.866	417.931	0.276
	Margin	Net	21,515.866	417.931	0.019
		Impact	-984.135	856.660	-0.870

The paper's own calculation walks the four loss ratio intervals, weights each by the expected loss ratio points falling in it, and subtracts the commission given up from the maximum. That is Appendix K Exhibit 1, Figure 7.5, and Table 7.16 reproduces it line by line.

APPENDIX K
EXHIBIT 1

SLIDING SCALE COMMISSIONS

(1) Actual or estimated subject premium for commission adjustment period 25,000,000
(2) Expected layer loss cost for the entire layer (expressed as a percentage of subject premium) 10.0000%
(3) Coinsurance percentage (cedant's participation in layer losses not corresponding to an explicit share of the reinsurance premium, excluding the effects of nonproportional loss sharing plans.) 0.00%
(4) Percentage reduction in layer losses due to nonproportional loss sharing provisions only 0.00%
(5) Treaty rate [expressed as a percentage of subject premium] 20.000%
(6) Expected treaty loss & ALAE ratio (*ELR*) {(2) × [100% − (3)] × [100% − (4)]}/(5) 50.00%
(7) Minimum commission ratio 20.00%;
corresponding loss & ALAE ratio 65.00%
(8) The details of the sliding scale commission plan are summarized in columns (A) through (E). Values used in the calculation of the expected sliding scale commission are given in columns (F) through (I).

Loss & ALAE Ratio Interval (A) Lower Bound	(B) Upper Bound	(C) Percentage Increase in Commission Ratio Per 1% Decrease in Loss & ALAE Ratio	Corresponding Commission Ratio Interval (D) Lower Bound	(E) Upper Bound	(F) Entry Ratio Corresponding to Lower Bound Loss Ratio in Column (A)	(G) Insurance Charge Corresponding to Lower Bound Entry Ratio in Column (F)	(H) Expected Loss Ratio Points in Interval	(I) Expected Reductions from Maximu Commissior Rate (C) × (H)
65.00% and above		0.00%	20.00%	20.00%	1.300	9.12%	4.56%	0.00%
55.00%	65.00%	0.50%	25.00%	20.00%	1.100	14.47%	2.68%	1.34%
35.00%	55.00%	0.75%	40.00%	25.00%	0.700	34.80%	10.16%	7.62%
0.00%	35.00%	0.00%	40.00%	40.00%	0.000	100.00%	32.60%	0.00%
						Total	50.00%	8.96%

(9) Expected ceding commission ratio from a simplistic point of view [commission ratio corresponding to the treaty *ELR* (item 6), given the plan above.] 28.75%
(10) Expected commission ratio from an actuarial point of view [maximum commission ratio—(total 8I)] 31.04%
(11) Amount by which the actuarial estimate of the expected commission ratio exceeds the simplistic estimate [(10) − (9)] 2.29%

NOTES:

On this exhibit, all commission and loss & ALAE ratios are expressed as percentages of treaty premium.

Column (8G): The insurance charges appearing in this column are based on a lognormal distribution with a 0.485 coefficient of variation. These insurance charges are obtained via linear interpolation of the table of insurance charges given in Tables B1–B3 of Appendix B. A procedure similar to that employed in the Treaty I example (see Appendix F Exhibit 1) is used to derive the 0.485 coefficient of variation for aggregate losses in the reinsured layer on this treaty.

Column (8H): The values in this column are obtained by multiplying the differences between the insurance charges corresponding to consecutive loss & ALAE ratio interval end points in column (8G) by the expected treaty loss & ALAE ratio (item 6).

Figure 7.5.: Bear and Nemlick (1990), Appendix K, Exhibit 1.

```
SLIDE = [('65% and above', 0.65, 0.00, 0.20), ('55% to 65%', 0.55, 0.50, 0.25),
         ('35% to 55%', 0.35, 0.75, 0.40), ('0% to 35%', 0.00, 0.00, 0.40)]
elr6 = 0.10 / 0.20
```

```
rows, prev = [], 0.0
for band, lower, slope, commission in SLIDE:
    p2 = charge(0.485, lower / elr6)
    points = elr6 * (p2 - prev)
    rows.append({'loss ratio interval': band, 'commission': commission,
                 'slide per point': slope, 'entry ratio': lower / elr6,
                 'insurance charge': p2, 'loss ratio points': points,
                 'reduction from maximum': slope * points})
    prev = p2
k1 = pd.DataFrame(rows).set_index('loss ratio interval')
k1.loc['total'] = k1[['loss ratio points', 'reduction from maximum']].sum()
GT(k1, float_format=4)
```

Table 7.16.: Appendix K Exhibit 1 rebuilt, the four loss ratio intervals and the commission given up in each.

loss ratio interval	commission	slide per point	entry ratio	insurance charge	loss ratio points	reduction from maximum
65% and above	0.2000	0.0000	1.3000	0.0912	0.0456	0.0000
55% to 65%	0.2500	0.5000	1.1000	0.1447	0.0268	0.0134
35% to 55%	0.4000	0.7500	0.7000	0.3480	0.1016	0.0762
0% to 35%	0.4000	0.0000	0.0000	1.0000	0.3260	0.0000
total	—	—	—	—	0.5000	0.0896

```
GT(pd.DataFrame([{
    'aggregate cv': six['cv'], 'paper cv': 0.485,
    'simplistic view %': 100 * (0.25 + 0.75 * (0.55 - elr6)),
    'expected commission %, lognormal': 100 * (0.40 - k1.loc['total', 'reduction from maximum']),
    'expected commission %, aggregate': six['key'],
    'expected commission %, paper c = 0': 30.31}]), float_format=4)
```

Table 7.17.: Treaty VI, the expected sliding scale commission under both models and under the simplistic view, against Table I.

aggregate cv	paper cv	simplistic view %	expected commission %, lognormal	expected commission %, aggregate	expected commission %, paper c = 0
0.4850	0.4850	28.7500	31.0394	30.3173	30.3100

Table 7.17 prices the feature. The Pareto shape was chosen to hit the published coefficient of variation, and it is worth checking how much that choice matters. In Table 7.18, over a range of shapes wide enough to move the coefficient of variation by three points, the expected commission moves by three basis points.

```
import warnings

# The q = 1.00 row divides by (i - alpha) in the library's analytic Pareto
# moment formula, which is 0/0 at an integer shape, so numpy warns. The sum
# converges anyway and the row lands smoothly between its neighbours; the
```

```
# warning is about the route, not the answer.
with warnings.catch_warnings():
    warnings.simplefilter('ignore', RuntimeWarning)
    sens = pd.DataFrame([{'pareto q': q, 'aggregate cv': r['cv'],
                          'expected commission %': r['key']}
                         for q in (0.95, Q6, 1.00, 1.05, 1.10, 1.20)
                         for r in [treaty_6(FREQ['c = 0'], q)]]
                        ).set_index('pareto q')
GT(sens, float_format=4)
```

Table 7.18.: Treaty VI's expected commission against the Pareto shape, which the paper never published.

pareto q	aggregate cv	expected commission %
0.9500	0.4890	30.3206
0.9800	0.4850	30.3173
1.0000	0.4824	30.3150
1.0500	0.4758	30.3089
1.1000	0.4691	30.3023
1.2000	0.4558	30.2875

7.3.7. Table I

TABLE I

COMPARISON OF KEY ITEMS

Example	Unadjusted Rate	Item Compared	Lognormal Model	Collective Risk Model $c = 0$ $b = 0$	Collective Risk Model $c = .05$ $b = .05$	Collective Risk Model $c = .05$ $b = .10$	Collective Risk Model $c = .10$ $b = .10$
I) Aggregate Deductible	5.00%	Adjusted Rate	1.47%	1.58%	1.68%	1.73%	1.77%
II) Limited Reinstatement	25.00	Adjusted Rate	19.53	19.89	19.72	19.55	19.52
III) Loss Corridor	5.00	Adjusted Rate	4.02	3.67	3.71	3.74	3.73
IV) Retro Rating Plan	5.00	Expected Rate After Retro Adjustments	5.02	5.20	5.18	5.14	5.14
V) Profit Commission		Expected Profit Commission	8.37	8.24	8.50	8.69	8.75
VI) Sliding Scale Commission		Expected Sliding Scale Commission	31.04	30.31	30.90	31.22	31.33

Figure 7.6.: Bear and Nemlick (1990), Table I.

```
TREATIES = {'I': treaty_1, 'II': treaty_2, 'III': treaty_3,
            'IV': treaty_4, 'V': treaty_5, 'VI': treaty_6}
computed = pd.DataFrame({col: {k: f(freq)['key'] for k, f in TREATIES.items()}
                         for col, freq in FREQ.items()})
cvs = pd.DataFrame({col: {k: f(freq)['cv'] for k, f in TREATIES.items()}
                    for col, freq in FREQ.items()})

lognormal = {
    'I': 100 / 75 * 3.75 * charge(0.528, 360 / 450),
    'II': 100 / 60 * 15 * 0.80 * (1 - charge(0.770, 2800 / 900)),
    'III': 100 / 70 * 3.5 * (1 - (charge(0.905, 1.0) - charge(0.905, 2.0))),
    'IV': 100 / 75 * 3.75 * (1 - net_charge),
    'V': 100 * 0.25 * (1 - elr * (1 - charge(0.444, 0.80 / elr)) - 0.20),
    'VI': 100 * (0.40 - k1.loc['total', 'reduction from maximum']),
}

table_1 = pd.DataFrame({
    'paper lognormal': [1.47, 19.53, 4.02, 5.02, 8.37, 31.04],
    'aggregate lognormal': pd.Series(lognormal),
    'paper c = 0': [1.58, 19.89, 3.67, 5.20, 8.24, 30.31],
    'aggregate c = 0': computed['c = 0'],
    'paper c = .05, b = .05': [1.68, 19.72, 3.71, 5.18, 8.50, 30.90],
    'aggregate c = .05': computed['c = .05'],
    'paper c = .10, b = .10': [1.77, 19.52, 3.73, 5.14, 8.75, 31.33],
    'aggregate c = .10': computed['c = .10'],
}, index=['I', 'II', 'III', 'IV', 'V', 'VI'])
table_1.index.name = 'Treaty'
GT(table_1, float_format=4)
```

Table 7.19.: Table I rebuilt, all six treaties under the lognormal model and at three contagion levels, beside the printed figures.

Treaty	paper lognormal	aggregate lognormal	paper c = 0	aggregate c = 0	paper c = .05, b = .05	aggregate c = .05	paper c = .10, b = .10	aggregate c = .10
I	1.4700	1.4664	1.5800	1.5755	1.6800	1.6490	1.7700	1.7167
II	19.5300	19.5269	19.8900	19.8862	19.7200	19.8373	19.5200	19.7809
III	4.0200	4.0248	3.6700	3.6697	3.7100	3.6662	3.7300	3.6639
IV	5.0200	5.0208	5.2000	5.2042	5.1800	5.2202	5.1400	5.2296
V	8.3700	8.3704	8.2400	8.2383	8.5000	8.3771	8.7500	8.5215
VI	31.0400	31.0394	30.3100	30.3173	30.9000	30.4943	31.3300	30.6569

In Table 7.19 both reproducible columns come back.

The **lognormal column** is exact at every printed digit: 1.4664, 19.5269, 4.0248, 5.0208, 8.3704 and 31.0394 against 1.47, 19.53, 4.02, 5.02, 8.37 and 31.04. That is a reproduction of the paper's arithmetic rather than of `aggregate`, and it matters because it fixes every convention in the six exhibits. Had any entry ratio, coinsurance order or loading been read wrongly, one of the six would have missed.

The **collective risk column at** $c = 0$, $b = 0$ comes back to the printed two decimals in five of six treaties and within one basis point in the sixth: 1.5755, 19.8862, 3.6697, 5.2042, 8.2383 and 30.3173 against 1.58, 19.89, 3.67, 5.20, 8.24 and 30.31. The sixth is Treaty VI, the one whose subject book the paper never published.

The remaining two columns are not comparable, since they carry a severity parameter uncertainty loading `aggregate` has no analog for. The contagion only figures beside them show what the fre-

quency half of the loading does on its own. Table 7.20 sets the two frequency assumptions side by side.

```
all_cvs = pd.concat([cvs, pd.Series([0.528, 0.770, 0.905, 0.528, 0.444, 0.485],
                                    index=cvs.index, name='paper lognormal model')],
                    axis=1)
all_cvs.index.name = 'Treaty'
GT(all_cvs, float_format=4)
```

Table 7.20.: Aggregate coefficient of variation by treaty at three contagion levels, against the figure the paper's lognormal model uses.

Treaty	c = 0	c = .05	c = .10	paper lognormal model
I	0.5218	0.5677	0.6102	0.5280
II	0.7665	0.7985	0.8292	0.7700
III	0.9053	0.9325	0.9589	0.9050
IV	0.5218	0.5677	0.6102	0.5280
V	0.4425	0.4958	0.5439	0.4440
VI	0.4850	0.5341	0.5790	0.4850

The coefficients of variation in Table 7.20 say where the two frequency assumptions sit relative to each other. The paper's lognormal model loads the excess claim count with the variance to mean ratios of Appendix D rather than with a contagion parameter, and the result lands at or a little above the Poisson figure and well short of $c = 0.05$. Treaty III's loading is negligible and Treaty VI's shape was chosen to match, so those two agree by construction.

Two modeling differences sit underneath the $c > 0$ columns and not the $c = 0$ one. `aggregate` draws a single gamma mixing variable for a whole program, so the classes of a multi class book share their parameter error, and Treaty V's three years share it too. The paper adds class variances and year variances as though they were independent. Under Poisson counts there is no draw to share and the two constructions coincide, which is the column that carries the reproduction. Arguably the shared draw is the better model of the thing being described, since parameter error in an account's expected claim count does not resample itself between that account's classes or from one year of the same account to the next.

7.4. Comments

7.4.1. Which model is wrong, and about what

The paper set out to ask how much accuracy the lognormal shortcut costs, and answered that for these six treaties it costs little. The reproduction agrees with the answer and sharpens the reason.

The error is not where the rate column makes it look. On rate the lognormal is within 4% of the exact answer on four of the six treaties, and out by 7% on Treaty I and by 9.5% on Treaty III. On the insurance charge, which is the quantity the method actually produces, it is worst by a long way on Treaty II: 2.37% of layer losses eliminated by the annual limit against a true 0.57%, a factor of four. The difference between the two readings is dilution. Treaty II's charge is multiplied by an 80% coinsurance and shifts the rate by a third of a point, while Treaty I's charge is the answer and carries straight through.

The direction of the error is consistent, and it follows from one number. A lognormal matched to a mean and a coefficient of variation of $\nu = 0.528$ has skewness $3\nu + \nu^3 = 1.73$, where the aggregate it is standing in for has 0.62. It is far too skewed for a compound Poisson with six and a half claims, so it holds too little mass in the body and too much in the tail. Every charge read below the mean therefore comes out low and every charge read far above it comes out high, which is exactly the pattern: 29.33% against 31.51% at an entry ratio of 0.8, and 2.37% against 0.57% at an entry ratio of 3.1.

The two treaties the lognormal misses on rate are the two that read the body. An aggregate deductible at 0.8 times the mean eliminates about 70% of expected losses, and a corridor spanning entry ratios 1 to 2 sits astride the mode. Neither is a tail calculation, and neither is the kind of thing a two moment fit is good at.

7.4.2. What the reinstatement clause changes

The paper prices three free reinstatements by converting them to an aggregate limit of $2,800 and reading an entry ratio, which is correct here and agrees to six figures in Table 7.8. It is correct because the reinstatements are free. Once they are paid the conversion stops working, because the treaty no longer has a single loss transform: it has a recovery $A(R) = R \wedge (m+1)y$ and a premium $h(R)$, both driven by the same R, and the two are comonotone.

The clause carries both. Repricing Treaty II with three reinstatements at 100% rather than free changes nothing about the recovery and turns the ceded premium into a distribution, as Table 7.21 shows.

```
paid = build('xpnl BN.TreatyIIp 1172 premium less '
             + BOOK_2.format(freq=FREQ['c = 0'])
             .replace('reinstatements 3 free', 'reinstatements 3 at 100%'))
GT(pd.concat({'3 free': two['x'].economic_df[['EX', 'SD']],
              '3 at 100%': paid.economic_df[['EX', 'SD']]},
             axis=1), float_format=3)
```

Table 7.21.: Treaty II with three free reinstatements against three at 100%, leg by leg.

			3 free		3 at 100%	
Step	***Side***	***Label***	**EX**	**SD**	**EX**	**SD**
Gross	Consideration	Premium	1,172.000	0.000	1,172.000	0.000
	Obligation	Loss	-900.000	689.879	-900.000	689.879
	Margin	Gross	272.000	689.879	272.000	689.879
Layer	Consideration	Layer premium	-1,172.000	0.000	-2,633.611	1,042.487
	Obligation	Layer recovery	894.878	672.732	894.878	672.732
	Margin	Total	-277.122	672.732	-1,738.732	388.784
		Net	-5.122	62.576	-1,466.732	1,054.343
All	Consideration	Net	0.000	0.000	-1,461.611	1,042.487
	Obligation	Net	-5.122	62.576	-5.122	62.576
	Margin	Net	-5.122	62.576	-1,466.732	1,054.343
		Impact	-277.122	672.732	-1,738.732	388.784

In Table 7.21 the recovery leg is identical in both, at 894.878. The premium leg goes from a fixed 1,172 to a mean of 2,634 with a standard deviation of 1,042. That second number is not available

from any one dimensional calculation on the aggregate, and it is the number a reinsurer needs in order to know what it is actually being paid.

7.4.3. One feature per program, and why

Each of the six treaties carries exactly one adjustable feature, which is also the constraint the implementation imposes. It is not arbitrary. Three of the features read a ceded loss ratio, and the ratio is only a deterministic rescaling of the ceded loss while the denominator is a fixed number. Put two features on one program and the second one reads a random denominator, and every map stops being a function of one argument.

The paper's own section 5, under treaties carrying both coinsurance provisions and adjustable features, reaches the same conclusion from the other side. It observes that a treaty carrying both a nonproportional coinsurance clause and an adjustable feature needs the distribution of aggregate losses *after* the coinsurance in order to price the feature, and proposes running the collective risk model a second time on that distribution. That is a two stage construction, not a single decorated layer, and it is the honest way to stack.

7.4.4. What the paper still gets to teach

Two of the appendices carry results that survive the change of method.

Appendix D derives the excess claim count variance to mean ratio, $VMR_E = (1 - p) + p\,VMR_G$, where p is the probability a claim exceeds the retention. Since p is small in an excess layer, VMR_E is close to one whatever the ground up count does, so the excess count is close to Poisson even when the ground up count is not. That is the paper's own justification for treating an excess count as Poisson, and it is what makes the $c = 0$ column the natural base case rather than an idealization. It is a fact about excess layers, not about any particular software.

Appendix E's change in layer factor, $(c^{1-q} - d^{1-q})/(a^{1-q} - b^{1-q})$, moves an expected loss from one layer to another under a single parameter Pareto, and its leveraged inflation factor $(1 + i)^{tq}$ scales excess frequency after t years of inflation at rate i. Both are still the right way to move a burning cost across a tower, and neither is superseded by having the aggregate distribution.

7.4.5. Where this sits for `aggregate`

Six treaties, six clauses, and no arithmetic outside the loading factors the slips specify. That is the difference between this page and an earlier treatment of the same examples, which built each feature by hand from `density_df`: a column of losses, a column of probabilities, a `numpy` expression for the payoff and a sum product. The hand built version gets the same means. What it does not get is the rest of the ledger, because a mean computed by sum product is not attached to anything. Here the profit commission, the sliding commission, the retro premium and the reinstatement premium are legs of a P&L with their own standard deviations and their own percentiles, and they foot.

The theory behind the five decorators, the maps they write and the reason at most one may decorate a program, is in the variable rating technical guide; the reinstatement schedule and its two maps are in the reinstatements guide beside it.

Part III.

Distortion pricing

8. Wang and risk loads by layer

Reproducing Wang (2000), Table 1

8.1. Context and problem

Wang (2000) introduces the distortion operator now known as the Wang transform,

$$g_\alpha(u) = \Phi\big(\Phi^{-1}(u) + \alpha\big),$$

and argues for it against the proportional hazard transform of Wang (1995), $g_r(u) = u^r$, which had been the actuarial default for pricing by layer. Both are distortions, so both price a layer by integrating the *distorted* survival function over it, and both give a risk load that grows with the layer. The paper's case is about how fast.

The criterion comes from Venter (1991): market prices by layer imply a distortion, and the distortion has to produce a relative loading that increases with the layer, since that is what observed reinsurance prices do. The PH transform does that. The paper's complaint is that it does it *too fast* at high layers, and Table 1 is the demonstration.

This is a reproduction of a pricing exhibit rather than a numerical method, so what is being tested is `aggregate`'s distortion machinery and its layer decomposition: `lev` for the expected loss and `exag` for the distorted equivalent, differenced across a tower.

8.2. Paper solution and gold standard

Example 1 takes a ground-up liability risk with a Pareto survival function

$$S_X(x) = \left(\frac{2000}{2000 + x}\right)^{1.2}, \qquad x > 0,$$

which is a Lomax with shape 1.2 and scale 2000, and assumes exactly one claim. The Wang loading is $\alpha = 0.1$. The PH index is then chosen as $r = 0.9245$ "to yield the same relative loading (14.5 percent) for the basic limit layer (\$0, \$50,000]", which makes the first row a calibration rather than a result.

Table 1, Figure 8.1, gives, for ten layers, the expected loss, the PH premium and loading, and the Wang premium and loading. Table 8.1 transcribes it.

TABLE 1
Risk Load by Layer Under Distortion g_α and PH-transform

Layer in 000's	Expected Loss	PH Premium	Relative Loading %	H[X; α] Premium	Relative Loading %
(0, 50]	4,793	5,487	14.5	5,487	14.5
(50, 100]	657	910	38.4	845	28.6
(100, 200]	582	857	47.2	769	32.2
(200, 300]	307	475	54.7	414	34.9
(300, 400]	204	325	59.6	278	36.6
(400, 500]	150	246	63.3	207	37.8
(500, 1000]	428	728	70.1	598	39.9
(1000, 2000]	373	675	81.1	533	43.0
(2000, 5000]	420	819	94.7	616	46.5
(5000, 10000]	271	567	109.5	406	49.9

Figure 8.1.: Wang (2000), Table 1.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build, show

ALPHA, THETA = 1.2, 2000.0
LAYERS = [0, 50e3, 100e3, 200e3, 300e3, 400e3, 500e3, 1000e3, 2000e3, 5000e3, 10000e3]
NAMES = [f'({int(a/1000)}, {int(b/1000)}]' for a, b in zip(LAYERS[:-1], LAYERS[1:])]
PAPER = pd.DataFrame(
    {'expected loss': [4793, 657, 582, 307, 204, 150, 428, 373, 420, 271],
     'PH premium': [5487, 910, 857, 475, 325, 246, 728, 675, 819, 567],
     'PH loading %': [14.5, 38.4, 47.2, 54.7, 59.6, 63.3, 70.1, 81.1, 94.7, 109.5],
     'Wang premium': [5487, 845, 769, 414, 278, 207, 598, 533, 616, 406],
     'Wang loading %': [14.5, 28.6, 32.2, 34.9, 36.6, 37.8, 39.9, 43.0, 46.5, 49.9]},
    index=NAMES)
GT(PAPER, float_format=1)
```

```
Exception reporting mode: Minimal
```

Table 8.1.: Wang (2000), Table 1 as printed.

level_0	expected loss	PH premium	PH loading %	Wang premium	Wang loading %
(0, 50]	4,793	5,487	14.5	5,487	14.5
(50, 100]	657	910	38.4	845	28.6
(100, 200]	582	857	47.2	769	32.2
(200, 300]	307	475	54.7	414	34.9
(300, 400]	204	325	59.6	278	36.6
(400, 500]	150	246	63.3	207	37.8
(500, 1000]	428	728	70.1	598	39.9
(1000, 2000]	373	675	81.1	533	43.0
(2000, 5000]	420	819	94.7	616	46.5
(5000, 10000]	271	567	109.5	406	49.9

The expected loss column has a closed form, which is what makes Table 8.1 worth checking rather than merely reproducing. A Lomax has

$$\mathsf{E}[X \wedge a] = \frac{\theta}{\alpha - 1}\left(1 - \left(\frac{\theta}{\theta + a}\right)^{\alpha - 1}\right),$$

so every layer expected loss is a difference of two elementary expressions, checked in Table 8.2.

```
def lev(a):
    return THETA / (ALPHA - 1) * (1 - (THETA / (THETA + a)) ** (ALPHA - 1))

cl = np.diff([lev(a) for a in LAYERS])
chk = pd.DataFrame({'closed form': cl, 'Table 1': PAPER['expected loss'].to_numpy()},
                   index=NAMES)
chk['difference'] = chk['closed form'] - chk['Table 1']
chk['rounding explains it'] = chk['difference'].abs() < 0.5
GT(chk, formatters={'closed form': 2, 'Table 1': 0,
                    # GT format sugar has no sign flag, so the signed column
                    # goes through a callable; `formatters` takes one, and
                    # `float_format` deliberately does not.
                    'difference': lambda x: f'{x:+.2f}'})
```

Table 8.2.: The ten layer expected losses in closed form, against the column Table 1 prints.

level_0	closed form	Table 1	difference	rounding explains it
(0, 50]	4,787.99	4,793	-5.01	False
(50, 100]	657.04	657	+0.04	True
(100, 200]	581.82	582	-0.18	True
(200, 300]	307.05	307	+0.05	True
(300, 400]	203.83	204	-0.17	True
(400, 500]	150.46	150	+0.46	True
(500, 1000]	427.56	428	-0.44	True
(1000, 2000]	372.86	373	-0.14	True
(2000, 5000]	420.27	420	+0.27	True
(5000, 10000]	270.62	271	-0.38	True

In Table 8.2, **nine of the ten layers agree with the closed form to within half a dollar**, which is exactly what rounding to the nearest dollar allows. The basic limit layer does not: the closed form gives 4,787.99 and the paper prints 4,793, a difference of 5.01. That cannot be rounding, and it is the same row the technical guides flagged as looking wrong without pinning it down.

8.3. The `aggregate` reproduction

The tower goes in as an occurrence cession, and the distortions as their own objects. Table 8.3 confirms the grid before any pricing is read off it.

```
a1 = build('agg Wang.Layers 1 claim 10000e3 xs 0 sev 2000 * lomax 1.2 '
           f'occurrence ceded to tower {LAYERS} fixed', bs=20, log2=20)
show(a1)
d_wang = build(f'distortion wang_a wang 0.1')
d_ph = build(f'distortion ph_r ph 0.9245')
show(d_wang)
show(d_ph)
```

```
agg Wang.Layers
  1 claim
  10000000 xs 0
  sev 2000 * lomax 1.2
  occurrence ceded to
    50000 xs 0 and
    50000 xs 50000 and
    100000 xs 100000 and
    100000 xs 200000 and
    100000 xs 300000 and
    100000 xs 400000 and
    500000 xs 500000 and
    1000000 xs 1000000 and
    3000000 xs 2000000 and
    5000000 xs 5000000
  fixed
```

```
distortion wang_a wang 0.1
```

```
distortion ph_r ph 0.9245
```

```
GT(pd.DataFrame([
    {'quantity': 'E[X and 10M], aggregate', 'value': a1.est_m},
    {'quantity': 'E[X and 10M], closed form', 'value': lev(10e6)},
    {'quantity': 'relative difference',
     'value': abs(a1.est_m / lev(10e6) - 1)},
]).set_index('quantity'), formatters={'value': '{x:.6g}'})
```

Table 8.3.: The limited expected value at the top of the tower, against the closed form.

quantity	value
E[X and 10M], aggregate	8179.5
E[X and 10M], closed form	8179.51
relative difference	1.22254e-06

```
def by_layer(dist):
    a1.apply_distortion(dist)
    d = a1.density_df.loc[LAYERS, ['lev', 'exag']]
    return np.diff(d.lev.to_numpy()), np.diff(d.exag.to_numpy())

el_w, pr_w = by_layer(d_wang)
el_p, pr_p = by_layer(d_ph)
got = pd.DataFrame({'expected loss': el_w, 'PH premium': pr_p,
                    'PH loading %': 100 * (pr_p / el_p - 1),
```

```
                    'Wang premium': pr_w,
                    'Wang loading %': 100 * (pr_w / el_w - 1)}, index=NAMES)
GT(pd.concat({'paper': PAPER, 'aggregate': got}, axis=1), float_format=1)
```

Table 8.4.: Table 1 rebuilt, all five columns over the ten layer tower, beside the printed figures.

	paper					aggregate				
level_0	expected loss	PH premium	PH loading %	Wang premium	Wang loading %	expected loss	PH premium	PH loading %	Wang premium	Wang loading %
(0, 50]	4,793	5,487	14.5	5,487	14.5	4,788.0	5,481.4	14.5	5,481.8	14.5
(50, 100]	657	910	38.4	845	28.6	657.0	909.5	38.4	845.0	28.6
(100, 200]	582	857	47.2	769	32.2	581.8	856.4	47.2	768.9	32.2
(200, 300]	307	475	54.7	414	34.9	307.1	474.9	54.7	414.2	34.9
(300, 400]	204	325	59.6	278	36.6	203.8	325.3	59.6	278.4	36.6
(400, 500]	150	246	63.3	207	37.8	150.5	245.7	63.3	207.3	37.8
(500, 1000]	428	728	70.1	598	39.9	427.6	727.4	70.1	598.0	39.9
(1000, 2000]	373	675	81.1	533	43.0	372.9	675.3	81.1	533.2	43.0
(2000, 5000]	420	819	94.7	616	46.5	420.3	818.4	94.7	615.6	46.5
(5000, 10000]	271	567	109.5	406	49.9	270.6	567.0	109.5	405.7	49.9

```
rel = (got / PAPER - 1).abs()
GT(pd.DataFrame([
    {'column': c, 'worst relative difference': float(rel[c].max()),
     'worst layer': rel[c].idxmax()} for c in PAPER.columns]
    + [{'column': 'all fifty', 'worst relative difference': float(rel.max().max()),
        'worst layer': ''}]).set_index('column'),
   formatters={'worst relative difference': '{x:.2e}'})
```

Table 8.5.: Worst relative difference from the printed table, column by column and over all fifty entries.

column	worst relative difference	worst layer
expected loss	3.07e-03	(400, 500]
PH premium	1.15e-03	(400, 500]
PH loading %	1.22e-03	(0, 50]
Wang premium	1.48e-03	(400, 500]
Wang loading %	1.16e-03	(100, 200]
all fifty	3.07e-03	

Table 8.4 and Table 8.5 have all fifty published entries reproducing to within 0.31% relative, and every loading percentage matching to the tenth of a point printed. The paper's qualitative claim is the one it cares about and it comes back clearly: the PH loading runs from 14.5% to 109.5% across the tower while the Wang loading runs from 14.5% to 49.9%, so at the top layer the PH transform is charging more than twice the relative load for the same risk.

8.4. Comments

8.4.1. The basic limit entry is misprinted, and the loading column proves it

Two independent arguments say the 4,793 is wrong and 4,788 is right.

The closed form gives 4,787.99, and the other nine layers all agree with the closed form to within rounding, so the method of computing the column was sound everywhere else.

And the paper's own construction pins the row. The PH index $r = 0.9245$ was *chosen* to give the same 14.5% loading on this layer as the Wang transform does, and both premium columns print 5,487 for it. Table 8.6 checks what loading 5,487 implies against each candidate expected loss.

```
for_el = [4787.99, 4793]
GT(pd.DataFrame([
    {'expected loss used': e,
     'implied loading from a premium of 5487 %': 100 * (5487 / e - 1),
     'paper prints': 14.5} for e in for_el]).set_index('expected loss used'),
   float_format=3)
```

Table 8.6.: The loading a premium of 5,487 implies against each candidate basic limit expected loss.

expected loss used	implied loading from a premium of 5487 %	paper prints
4,787.990	14.599	14.500
4,793.000	14.479	14.500

In Table 8.6, against 4,788 a premium of 5,487 is a 14.6% loading; against 4,793 it is 14.5%. So the printed premium and the printed loading are consistent with each other and with 4,793, which says the error is upstream of the table: the expected loss was computed as 4,793 and then everything in the row was derived from it. It is a single wrong input, not a typo in one cell.

Nothing downstream depends on it. The row is a calibration, so its only job is to fix r, and by Table 8.7 r would move in the fourth decimal.

```
from scipy.optimize import brentq

def ph_loading(r):
    d = build(f'distortion tmp ph {r}')
    a1.apply_distortion(d)
    dd = a1.density_df.loc[[0, 50e3], ['lev', 'exag']]
    el = dd.lev.iloc[1] - dd.lev.iloc[0]
    pr = dd.exag.iloc[1] - dd.exag.iloc[0]
    return pr / el - 1

r_star = brentq(lambda r: ph_loading(r) - (5487 / 4787.99 - 1), 0.90, 0.95,
                xtol=1e-7)
GT(pd.DataFrame([
    {'quantity': "paper's PH index", 'value': 0.9245},
    {'quantity': 'index matching the closed-form loading', 'value': r_star},
    {'quantity': 'difference', 'value': r_star - 0.9245},
]).set_index('quantity'), float_format=7)
```

Table 8.7.: The PH index recalibrated against the closed form basic limit loss, beside the paper's.

quantity	value
paper's PH index	0.9245000
index matching the closed-form loading	0.9239451
difference	-0.0005549

8.4.2. The tower is a cession, not ten builds

Worth noting how little code Table 8.4 needs. `occurrence ceded to tower [...]` declares all ten layers in one clause, and the layer expected losses and distorted premiums are then differences of two `density_df` columns, `lev` and `exag`, read at the layer boundaries. There is no loop over layers and no per-layer object.

That works because a distortion price is an integral of the distorted survival function, so it is *additive across layers* in exactly the way the expected loss is. That additivity is the mathematical content of Venter (1991)'s observation and the reason a distortion is the right object for layer pricing in the first place: a risk measure without it would not let a tower be priced consistently layer by layer.

8.4.3. Where the two transforms actually differ

Table 8.8 puts the two loading columns side by side with their ratio.

```
GT(pd.DataFrame([
    {'layer': n, 'PH loading %': got.loc[n, 'PH loading %'],
     'Wang loading %': got.loc[n, 'Wang loading %'],
     'ratio': got.loc[n, 'PH loading %'] / got.loc[n, 'Wang loading %']}
    for n in NAMES]).set_index('layer'), float_format=3)
```

Table 8.8.: The PH and Wang relative loadings layer by layer, and the ratio between them.

layer	**PH loading %**	**Wang loading %**	**ratio**
(0, 50]	14.482	14.491	0.999
(50, 100]	38.427	28.611	1.343
(100, 200]	47.201	32.163	1.468
(200, 300]	54.675	34.900	1.567
(300, 400]	59.588	36.572	1.629
(400, 500]	63.311	37.782	1.676
(500, 1000]	70.124	39.869	1.759
(1000, 2000]	81.125	42.995	1.887
(2000, 5000]	94.728	46.479	2.038
(5000, 10000]	109.536	49.911	2.195

In Table 8.8 the ratio starts at 1 by construction and reaches 2.2 at the top layer. The paper's objection is that the PH transform's unbounded relative loading grows too quickly to match observed prices, while the Wang transform's also grows without bound but far more slowly.

That is a claim about the market rather than about mathematics, and the reproduction cannot settle it. What the reproduction does establish is that both operators are implemented here in a way that recovers the paper's numbers to a tenth of a point, so the comparison can be run on any other severity and tower without re-deriving anything.

9. Wang and the pricing of weather options

Reproducing Wang (2002), Table 2

9.1. Context and problem

Wang (2002) proposes the Wang transform as a single pricing framework covering financial and insurance risks. The transform distorts the survival function by

$$S^*(x) = \Phi\big(\Phi^{-1}(S(x)) + \lambda\big),$$

with λ a market price of risk. On a lognormal it reproduces Black-Scholes; on an insurance loss it produces the kind of loading actuaries recognize; and the paper's argument is that this is not a coincidence but the point.

The weather derivative example is the interesting one for a reproduction, because it is the case where no parametric model is used at all. The underlying is Heating Degree Days for December at a weather station, and the only thing available is 22 years of history. There is no lognormal to transform, no fitted severity, and nothing to estimate beyond the empirical distribution itself. The transform is applied directly to 22 equally weighted observations.

That makes it a good test of a specific thing: whether `aggregate` can carry a purely empirical distribution through a distortion and price a derivative on it, with no parametric step anywhere.

9.2. Paper solution and gold standard

Table 1, Figure 9.1, lists the December HDD totals for 1979 to 2000, 22 observations. Table 2, Figure 9.2, prices call options on Dec-2001 HDDs at six strikes with $\lambda = 0.25$, reporting the objective expected payoff, the transformed price, and the implied loading; Table 9.1 transcribes it.

TABLE 1

MONTHLY AGGREGATE DATA FOR CHICAGO O'HARE STATION, 1979-2000

Date	Dec-79	Dec-80	Dec-81	Dec-82	Dec-83	Dec-84	Dec-85	Dec-86
HDD	972.5	1147.0	1244.0	901.0	1573.0	1055.0	1488.0	1065.5
Date	Dec-87	Dec-88	Dec-89	Dec-90	Dec-91	Dec-92	Dec-93	Dec-94
HDD	1018.5	1155.0	1474.5	1129.5	1077.5	1129.5	1090.5	938.5
Date	Dec-95	Dec-96	Dec-97	Dec-98	Dec-99	Dec-00		
HDD	1199.5	1156.0	1040.0	940.5	1090.5	1517.5		

Figure 9.1.: Wang (2002), Table 1.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build, show

HDD = [972.5, 1147.0, 1244.0, 901.0, 1573.0, 1055.0, 1488.0, 1065.5,
       1018.5, 1155.0, 1474.5, 1129.5, 1077.5, 1129.5, 1090.5, 938.5,
       1199.5, 1156.0, 1040.0, 940.5, 1090.5, 1517.5]
LAMBDA = 0.25
PAPER = pd.DataFrame(
    {'expected payoff': [47.86, 38.77, 29.68, 20.59, 11.50, 4.11],
     'price': [68.21, 55.45, 42.70, 29.94, 17.18, 6.59],
     'loading %': [43, 43, 44, 45, 49, 60]},
    index=[1250, 1300, 1350, 1400, 1450, 1500])
PAPER.index.name = 'strike'
GT(PAPER, float_format=2)
```

```
Exception reporting mode: Minimal
```

Table 9.1.: Wang (2002), Table 2 as printed.

strike	expected payoff	price	loading %
1,250	47.86	68.21	43
1,300	38.77	55.45	43
1,350	29.68	42.70	44
1,400	20.59	29.94	45
1,450	11.50	17.18	49
1,500	4.11	6.59	60

The data transcription has an acceptance test that does not involve the transform at all. The objective expected payoff of a call struck at K is just $\frac{1}{22}\sum_i (x_i - K)_+$, so if the 22 numbers are right, six of Table 2's eighteen entries follow by arithmetic, as Table 9.2 confirms.

```
h = np.array(HDD)
emp = pd.DataFrame({'strike': PAPER.index,
                    'empirical (x - K)+ mean': [np.maximum(h - k, 0).mean()
                                                for k in PAPER.index],
                    'Table 2': PAPER['expected payoff'].to_numpy()})
emp['difference'] = emp['empirical (x - K)+ mean'] - emp['Table 2']
```

```
GT(emp.set_index('strike'),
   formatters={'empirical (x - K)+ mean': 4, 'Table 2': 2, 'difference': '{x:.4f}'})
```

Table 9.2.: The objective expected payoff at each strike straight from the 22 observations, against Table 2.

strike	empirical (x - K)+ mean	Table 2	difference
1,250	47.8636	47.86	0.0036
1,300	38.7727	38.77	0.0027
1,350	29.6818	29.68	0.0018
1,400	20.5909	20.59	0.0009
1,450	11.5000	11.50	0.0000
1,500	4.1136	4.11	0.0036

All six agree to within 0.004, which is the paper's rounding to two decimals. The 22 observations are transcribed correctly, and the paper's mean of 1,154.70 and standard deviation of 193.37 follow in Table 9.3.

```
GT(pd.DataFrame([
    {'quantity': 'observations', 'value': len(HDD)},
    {'quantity': 'mean HDD', 'value': float(np.mean(h))},
    {'quantity': 'standard deviation', 'value': float(np.std(h, ddof=1))},
]).set_index('quantity'), float_format=4)
```

Table 9.3.: The two summary statistics the paper quotes for the December HDD record.

quantity	value
observations	22.0000
mean HDD	1,154.7045
standard deviation	193.3740

9.3. The `aggregate` reproduction

The empirical distribution goes in as a `dsev`, a discrete severity with one atom per observation, carried by a single certain claim so the aggregate *is* the empirical law.

```
hdd = build(f'agg HDD 1 claim dsev {HDD} fixed', bs=1 / 32, log2=16)
show(hdd)
d = build(f'distortion wang{int(LAMBDA*100)} wang {LAMBDA}')
show(d)
```

```
agg HDD
  1 claim
  dsev [972.5 1147 1244 901 1573 1055 1488 1065.5 1018.5 1155 1474.5 1129.5 1077.5 1129.5 1090.5
  938.5 1199.5 1156 1040 940.5 1090.5 1517.5]
  fixed
```

```
distortion wang25 wang 0.25
```

TABLE 2

OPTION PRICES AT VARIOUS STRIKE LEVELS

Strike	1250	1300	1350	1400	1450	1500
Exp. Payoff	$ 47.86	$ 38.77	$ 29.68	$ 20.59	$ 11.50	$ 4.11
Price	$ 68.21	$ 55.45	$ 42.70	$ 29.94	$ 17.18	$ 6.59
Loading	43%	43%	44%	45%	49%	60%

Figure 9.2.: Wang (2002), Table 2.

```
bit = hdd.density_df.query('p_total > 0')[['p_total']]
rows = []
for k in PAPER.index:
    # push the atoms forward through the payoff map, then price the result
    payoff = bit.groupby(by=lambda x: np.maximum(0.0, x - k)).p_total.sum()
    p = d.price(payoff, kind='both')
    rows.append({'strike': k, 'expected payoff': p.el, 'price': p.ask,
                 'loading %': 100 * (p.ask / p.el - 1), 'bid': p.bid})
got = pd.DataFrame(rows).set_index('strike')
GT(pd.concat({'paper': PAPER, 'aggregate': got[PAPER.columns]}, axis=1),
   float_format=2)
```

Table 9.4.: Table 2 rebuilt, the expected payoff, transformed price and loading at all six strikes, beside the printed figures.

	paper			aggregate		
strike	expected payoff	price	loading %	expected payoff	price	loading %
1,250	47.86	68.21	43	47.86	68.21	42.50
1,300	38.77	55.45	43	38.77	55.45	43.02
1,350	29.68	42.70	44	29.68	42.70	43.84
1,400	20.59	29.94	45	20.59	29.94	45.40
1,450	11.50	17.18	49	11.50	17.18	49.42
1,500	4.11	6.59	60	4.11	6.59	60.08

```
rel = (got[PAPER.columns] / PAPER - 1).abs()
GT(pd.DataFrame([
    {'column': c, 'worst absolute difference':
        float((got[c] - PAPER[c]).abs().max()),
     'worst relative difference': float(rel[c].max())}
    for c in PAPER.columns]).set_index('column'),
   formatters={'worst absolute difference': '{x:.4f}',
               'worst relative difference': '{x:.2e}'})
```

Table 9.5.: Worst absolute and relative difference from Table 2, column by column.

column	worst absolute difference	worst relative difference
expected payoff	0.0036	8.85e-04
price	0.0049	7.48e-04
loading %	0.4963	1.15e-02

In Table 9.4 and Table 9.5 all eighteen published entries reproduce to the precision printed. The expected payoffs agree to 0.004, within the paper's two decimals. The six transformed prices, which depend on the whole machinery rather than on the data alone, agree to **under half a cent**: 68.21, 55.45, 42.70, 29.94, 17.18 and 6.59.

The loading row is printed as whole percentages and is the loosest of the three, since 42.50% at a strike of 1250 rounds to the paper's 43% only just. That is a consequence of rounding a ratio of two rounded quantities rather than a disagreement: the price and the payoff it is formed from both match to four decimals.

9.4. Comments

9.4.1. The loading rises with the strike, and that is the interesting part

Table 9.6 puts the payoff probability beside the loading it earns.

```
GT(pd.DataFrame([
    {'strike': k,
     'probability of payoff': float((h > k).mean()),
     'expected payoff': got.loc[k, 'expected payoff'],
     'price': got.loc[k, 'price'],
     'loading %': got.loc[k, 'loading %']}
    for k in PAPER.index]).set_index('strike'),
   formatters={'probability of payoff': 4, 'expected payoff': 3, 'price': 3,
               'loading %': 1})
```

Table 9.6.: How often each strike pays, and the relative loading the transform puts on it.

strike	probability of payoff	expected payoff	price	loading %
1,250	0.1818	47.864	68.207	42.5
1,300	0.1818	38.773	55.451	43.0
1,350	0.1818	29.682	42.695	43.8
1,400	0.1818	20.591	29.939	45.4
1,450	0.1818	11.500	17.183	49.4
1,500	0.0909	4.114	6.585	60.1

In Table 9.6 the relative loading runs from 43% at a strike of 1250 to 60% at 1500, and the driver is visible in the first column: at 1250 the option pays in 9 of 22 years and at 1500 in 2 of 22. A distortion that reweights the survival function loads a thin, remote payoff more heavily than a frequent

one, in relative terms, which is exactly the property Wang (2000) argues an insurance pricing operator should have. It is the same shape as the layer loadings in that paper, obtained here without a parametric severity anywhere in the calculation.

9.4.2. Twenty two observations is the whole model, and that is a real limitation

Nothing here smooths the data. The empirical distribution has 22 atoms, the distorted distribution has the same 22 atoms with reweighted probabilities, and the price is a weighted sum over them. That is faithful to the paper and it means the price at a strike above the largest observation is exactly zero, while the price just below it is determined by a single year, as Table 9.7 shows.

```
rows = []
for k in (1500, 1520, 1550, 1573, 1600):
    payoff = bit.groupby(by=lambda x: np.maximum(0.0, x - k)).p_total.sum()
    p = d.price(payoff, kind='both')
    rows.append({'strike': k, 'years with a payoff': int((h > k).sum()),
                 'expected payoff': p.el, 'price': p.ask})
GT(pd.DataFrame(rows).set_index('strike'),
   formatters={'years with a payoff': 0, 'expected payoff': 4, 'price': 4})
```

Table 9.7.: Prices at strikes reaching past the record, where the empirical distribution runs out of data.

strike	years with a payoff	expected payoff	price
1,500	2	4.1136	6.5851
1,520	1	2.4091	3.9668
1,550	1	1.0455	1.7215
1,573	0	0.0000	0.0000
1,600	0	0.0000	0.0000

In Table 9.7, past 1573, the highest December in the record, the option is free. Any real quote at those strikes needs a tail model, and the empirical route cannot supply one: this is the same limit Wang (2002) is working within, and the paper does not price above 1500. The reproduction inherits the limitation rather than papering over it, which is the right outcome, but it is worth stating that the method's applicability stops where the data does.

9.4.3. Why this example is a useful test of the library rather than of the transform

The transform is a one line function and would be easy to check on a lognormal. What this example exercises is the plumbing around it: a discrete severity given as a list of 22 floats, a pushforward through a payoff map that collapses many atoms onto zero, and a distortion applied to the resulting distribution rather than to the original.

That pushforward is the step where a library is likely to be wrong, because the payoff map is not injective: every year below the strike maps to a payoff of exactly zero, so nine to twenty of the 22 atoms have to be merged and their probabilities added before the distortion is applied. Applying the distortion first and then mapping would give a different and wrong answer, since a distortion is not linear. The agreement to two decimals at all six strikes is evidence that the order is right.

Part IV.

Risk sharing and conditional expectations given the sum

10. Denuit and conditional mean risk sharing

Reproducing Denuit (2019), Section 6.1

10.1. Context and problem

Denuit (2019) connects the conditional mean risk sharing rule to the **size-biased transform**. The rule allocates $\kappa_i(s) = \mathsf{E}[X_i \mid S = s]$ to participant i; the paper's representation formula rewrites that conditional expectation in terms of the size-biased version $\widetilde{X}_i$, whose density is $x f_i(x)/\mathsf{E}[X_i]$, and then reads off closed forms for compound Poisson and compound negative binomial pools.

The practical payoff is a result the paper uses in its own example and which is worth stating on its own: if two participants have the **same claim severity distribution**, then their combined share of any total loss s is the ratio of their combined expected claim frequency to the pool's, *independent of* s. Pooling participants with a common severity collapses to pooling frequencies.

That statement is an exact invariant, which makes this a reproduction with something firmer than a figure to check against.

10.2. Paper solution and gold standard

Section 6.1 describes a four participant peer to peer pool. Severities take values in $\{1, 2, 3, 4\}$, frequencies are Poisson, and case 1 sets $\lambda = (0.08, 0.08, 0.10, 0.10)$.

The paper publishes three things this page can check:

1. the shares at $s = 1$: participant 1 contributes 0.178, participant 2 contributes 0.267, participant 3 contributes 0.222, participant 4 contributes 0.333,
2. that the 99.9% quantile of S is 11 in case 1,
3. that Figure 1 shows "an horizontal line ... separating the shares of participants 1-2 from the share of participants 3-4", because participants 1 and 3 share a severity distribution and so do 2 and 4, so 1 and 2 together always contribute $(\lambda_1 + \lambda_2)/\lambda_\bullet$.

10.2.1. A discrepancy that has to be resolved first

Figure 10.1 and the paper's text disagree about participant 3. The text says plainly that "Participants 1 and 3 have identical severity distributions, as well as participants 2 and 4", which makes participant 3's severity $(0.1, 0.2, 0.4, 0.3)$, the same as participant 1's. The column for participant 3, as I read it, prints $(0.1, 0.2, 0.3, 0.4)$, with the last two entries swapped.

Only one of those can produce the horizontal line, so the paper's own Figure 1 decides it. Figure 10.1 is the table in question and Table 10.1 tests the two readings.

Participant i	1	2	3	4
Case 1				
λ_i	0.08	0.08	0.1	0.1
Case 2				
λ_i	0.04	0.08	0.1	0.14
$\mathrm{P}[C_i = 1]$	0.1	0.15	0.1	0.15
$\mathrm{P}[C_i = 2]$	0.2	0.25	0.2	0.25
$\mathrm{P}[C_i = 3]$	0.4	0.3	0.3	0.3
$\mathrm{P}[C_i = 4]$	0.3	0.3	0.4	0.3

Figure 10.1.: Denuit (2019), Section 6.1, the pool.

```
%xmode minimal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show

LAM = [0.08, 0.08, 0.10, 0.10]
SEV_A = '[.1 .2 .4 .3]'        # participant 1, and participant 3 per the text
SEV_B = '[.15 .25 .3 .3]'      # participants 2 and 4
SEV_TABLE3 = '[.1 .2 .3 .4]'   # participant 3 as the table column reads
PAPER_S1 = [0.178, 0.267, 0.222, 0.333]

def pool(sev3, name='Denuit61'):
    return build(f'''port {name}
    agg P1 {LAM[0]} claims dsev [1 2 3 4] {SEV_A} poisson
    agg P2 {LAM[1]} claims dsev [1 2 3 4] {SEV_B} poisson
    agg P3 {LAM[2]} claims dsev [1 2 3 4] {sev3} poisson
    agg P4 {LAM[3]} claims dsev [1 2 3 4] {SEV_B} poisson
''', bs=1, log2=10)

rows = []
for label, sev3, nm in [('text: participant 3 matches participant 1', SEV_A, 'D61a'),
                        ('table column 3 as printed', SEV_TABLE3, 'D61b')]:
    p = pool(sev3, nm)
    d = p.density_df
    share12 = np.array([float(d.loc[s, 'exeqa_P1'] + d.loc[s, 'exeqa_P2']) / s
                        for s in range(1, 16)])
    rows.append({'reading': label,
                 'participants 1+2 share, min over s = 1..15': share12.min(),
                 'max': share12.max(),
                 'constant to 1e-9': bool(np.ptp(share12) < 1e-9)})
GT(pd.DataFrame(rows).set_index('reading'), float_format=6)
```

```
Exception reporting mode: Minimal
```

Table 10.1.: The combined share of participants 1 and 2 over $s = 1$ to 15, under each reading of participant 3's severity.

reading	**participants 1+2 share, min over s = 1..15**	**max**	**constant to 1e-9**
text: participant 3 matches participant 1	0.444444	0.444444	True
table column 3 as printed	0.410518	0.481587	False

In Table 10.1 the text's reading gives a share for participants 1 and 2 that is constant to machine precision over the whole plotted range. The table's reading wanders between 0.41 and 0.48 and has no horizontal line anywhere. Figure 1 shows a horizontal line, so the text is right and the table's third column is wrong, and the rest of this page uses the text.

Whether the fault is in the published table or in my scan of it I cannot tell, so this is recorded as a discrepancy rather than as an erratum.

10.3. The `aggregate` reproduction

Each participant is one `agg` with a Poisson frequency and a four point discrete severity. κ_i is the `exeqa_` column, so no allocation code is needed. Table 10.2 checks the pool against three known values before any share is read.

```
p = pool(SEV_A)
show(p)
```

```
port Denuit61
  agg P1
    0.08 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P2
    0.08 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
  agg P3
    0.1 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P4
    0.1 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
```

```
d = p.density_df
GT(pd.DataFrame([
    {'quantity': 'E[S] from the pool', 'value': p.est_m,
     'exact': sum(l * m for l, m in zip(LAM, [
         (np.array([1, 2, 3, 4]) * np.array([.1, .2, .4, .3])).sum(),
         (np.array([1, 2, 3, 4]) * np.array([.15, .25, .3, .3])).sum(),
         (np.array([1, 2, 3, 4]) * np.array([.1, .2, .4, .3])).sum(),
         (np.array([1, 2, 3, 4]) * np.array([.15, .25, .3, .3])).sum()]))},
    {'quantity': '99.9% quantile of S', 'value': p.q(0.999), 'exact': 11},
    {'quantity': '(lam1 + lam2) / lam total',
     'value': sum(LAM[:2]) / sum(LAM), 'exact': 0.16 / 0.36},
]).set_index('quantity'), float_format=6)
```

Table 10.2.: The pool mean, the 99.9% quantile and the invariant group share, each against its exact value.

quantity	value	exact
E[S] from the pool	1.017000	1.017000
99.9% quantile of S	11.000000	11.000000
(lam1 + lam2) / lam total	0.444444	0.444444

```
got = [float(d.loc[1, f'exeqa_P{i}']) for i in (1, 2, 3, 4)]
GT(pd.DataFrame({'participant': [1, 2, 3, 4], 'aggregate share at s = 1': got,
                 'paper': PAPER_S1,
                 'difference': [g - q for g, q in zip(got, PAPER_S1)]}
                ).set_index('participant'),
   formatters={'aggregate share at s = 1': 6, 'paper': 3,
               # sugar has no sign flag; `formatters` accepts a callable
               'difference': lambda x: f'{x:+.5f}'})
```

Table 10.3.: The four shares at $s = 1$, against the values the paper publishes.

participant	aggregate share at s = 1	paper	difference
1	0.177778	0.178	-0.00022
2	0.266667	0.267	-0.00033
3	0.222222	0.222	+0.00022
4	0.333333	0.333	+0.00033

In Table 10.3 all four published shares reproduce to within 0.0003, which is the paper's rounding to three decimals: 0.17778 against 0.178, 0.26667 against 0.267, 0.22222 against 0.222, 0.33333 against 0.333. The 99.9% quantile is 11 as stated.

The paper derives those four numbers by hand, from the observation that $S = 1$ implies exactly one claim of size 1, so the shares are the relative probabilities of that claim belonging to each participant. That gives $\lambda_i \Pr[C_i = 1]$ normalized, which is a check that needs no software at all, in Table 10.4.

```
w = np.array([LAM[0] * 0.1, LAM[1] * 0.15, LAM[2] * 0.1, LAM[3] * 0.15])
GT(pd.DataFrame({'participant': [1, 2, 3, 4],
                 'lam_i P[C_i = 1], normalized': w / w.sum(),
                 'aggregate': got, 'paper': PAPER_S1}).set_index('participant'),
   float_format=6)
```

Table 10.4.: The shares at $s = 1$ by hand, from the library, and as published.

participant	lam_i P[C_i = 1], normalized	aggregate	paper
1	0.177778	0.177778	0.178000
2	0.266667	0.266667	0.267000
3	0.222222	0.222222	0.222000
4	0.333333	0.333333	0.333000

The three columns of Table 10.4 agree, so the closed form, the library and the paper all say the same thing at $s = 1$.

10.3.1. Figure 1

The published figure is Figure 10.2 and the reproduction Figure 10.3.

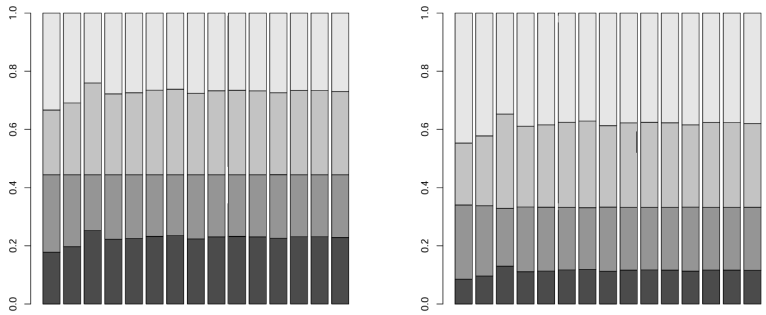


Figure 1: Respective shares $\mathrm{E}[X_i|S = s]/s$ of the total realized loss s contributed by each participant i in Case 1 (left panel) and Case 2 (right panel). From bottom to top, shares of participants 1 to 4.

Figure 10.2.: Denuit (2019), Figure 1.

```
ss = np.arange(1, 16)
sh = pd.DataFrame(
    {f'participant {i}': [float(d.loc[s, f'exeqa_P{i}']) / s for s in ss]
     for i in (1, 2, 3, 4)}, index=ss)
fig, axs = plt.subplots(1, 2, figsize=(2 * FIG_W, FIG_H), layout='constrained')
sh.plot(ax=axs[0], lw=1, marker='.', ms=3)
axs[0].set(xlabel='s', ylabel='share of s', title='individual shares', ylim=(0, 0.5))
axs[0].legend(fontsize=6)
(sh['participant 1'] + sh['participant 2']).plot(
    ax=axs[1], lw=1, marker='.', ms=3, label='participants 1 + 2')
(sh['participant 3'] + sh['participant 4']).plot(
    ax=axs[1], lw=1, marker='.', ms=3, label='participants 3 + 4')
for y in (sum(LAM[:2]) / sum(LAM), sum(LAM[2:]) / sum(LAM)):
    axs[1].axhline(y, ls='--', lw=0.6, c='C7')
axs[1].set(xlabel='s', ylabel='share of s', title='the horizontal separation',
           ylim=(0, 1))
axs[1].legend(fontsize=6)
```

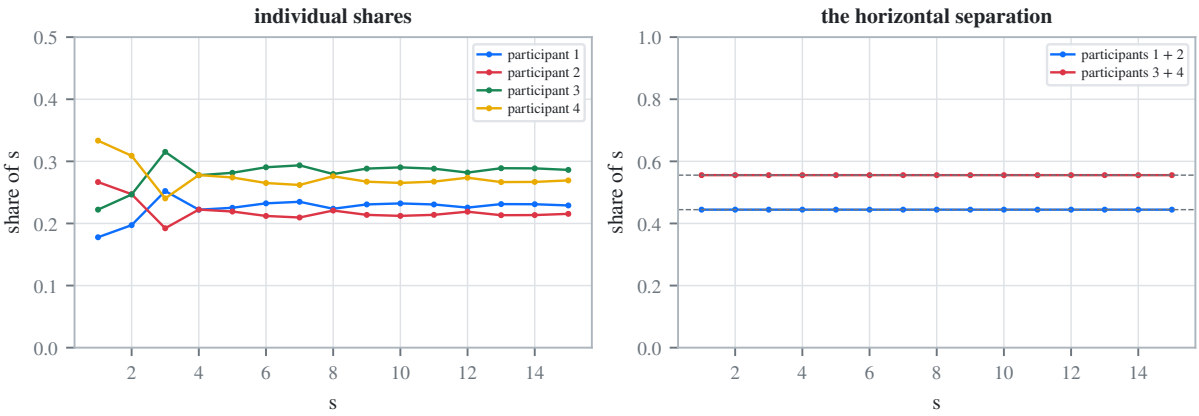


Figure 10.3.: Reproducing Figure 1. Share of the total loss contributed by each participant, and the horizontal separation at 4/9.

The left panel of Figure 10.3 is the paper's figure: four irregular curves, the irregularity coming from the discreteness of the severities, with participant 1 below participant 2 throughout because its severity is the lighter of the two. The right panel is the invariant, and the two group shares sit exactly on 4/9 and 5/9 at every s while the individual shares inside them move.

10.4. Comments

10.4.1. The invariant is the reproduction

The four numbers at $s = 1$ are a good check and a weak one: they are determined by a single event, one claim of size one, and the paper computes them by hand precisely because that case is elementary. Any implementation that got the frequencies and the first severity probability right would match them.

The horizontal line is the strong check, and it is strong for a structural reason. It holds at every s simultaneously, it depends on the whole severity distribution rather than one point of it, and it comes out of the paper's representation formula rather than out of arithmetic. Getting it to machine precision at fifteen values of s says the conditional expectation kernel is right, not merely that the inputs were typed correctly.

It is also the check that resolved the discrepancy in Figure 10.1, which is the useful thing about invariants: a wrong input that still satisfies the published point values can be caught by a structure the published point values do not see.

10.4.2. What the pool is, and what it is not

There is a compound distribution here, unlike the two Denuit, Dhaene, et al. (2025) and Denuit, Ortega-Jiménez, et al. (2025) chapters, where each agent carried one certain claim. Each participant has a Poisson claim count, so the frequency machinery is exercised, and the severities are genuinely discrete so no discretization error enters anywhere: `bs = 1` is exact on a severity supported on $\{1, 2, 3, 4\}$.

That makes this the cleanest of the three risk sharing reproductions numerically, and the least interesting economically. The frequencies are small, 0.08 to 0.10, so the pool's total is zero most of the time and the whole allocation question concerns the conditional distribution given that something happened. A real pool has enough participants for the law of large numbers to matter, which is what Denuit et al. (2022) takes up.

11. Denuit, Hieber and Robert on mortality credits

Reproducing Denuit, Hieber and Robert (2022), Example 4.5

11.1. Context and problem

A survivor fund pays the accumulated savings of members who die to the members who survive. The payment each survivor receives is the **mortality credit**, and the question is how to split it fairly when members differ in both their savings a_i and their death probability q_i.

Denuit et al. (2022) apply conditional mean risk sharing to this. Member i contributes $X_i = a_i \mathbb{1}\{i \text{ dies}\}$ to the pool, the pool total is $S_n = \sum_i X_i$, and the fair share is again $\mathsf{E}[X_i \mid S_n = s]$. The paper's contribution for large pools is an asymptotic expansion of that conditional expectation, accurate to $O(n_j^{-3/2})$, so that a fund with thousands of members does not need an exact convolution to allocate credits.

Example 4.5 illustrates the accuracy of that approximation, using the two group pool of Example 4.3, and its evidence is Figure 6, Figure 11.1.

11.2. Paper solution and gold standard

Example 4.3's pool has two groups: group 1 is 60% of the pool with $a_i = 1$ and $q_i = 0.1$; group 2 is 40% with $a_i = 3$ and $q_i = 0.2$. Figure 6 plots $s \mapsto \mathsf{E}[X_i \mid S_n = s]$ for $n \in \{100, 500, 1000\}$.

The quantity plotted is **per member**, not per group, which is what fixes the axes. Its unconditional value is $a_i q_i$, so 0.1 for group 1 and 0.6 for group 2, and Figure 6's first panel runs from 0.04 to 0.14 on the vertical axis over s from 10 to 60. That is group 1 at $n = 100$, whose expected total is $\mathsf{E}[S_{100}] = 100(0.6 \cdot 1 \cdot 0.1 + 0.4 \cdot 3 \cdot 0.2) = 30$.

The paper prints no table, so the checks available are the axis ranges, the structural identities the allocation must satisfy whatever the picture looks like, and the convergence the example exists to demonstrate.

```
%xmode minimal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show

W_LOW, W_HIGH = 0.6, 0.4
Q_LOW, Q_HIGH = 0.1, 0.2
A_LOW, A_HIGH = 1, 3
NS = (100, 500, 1000)
```

```
def fund(n):
    """Two groups of Bernoulli deaths. The count is binomial, the payout a_i."""
    return build(f'port Fund{n} '
                 f'agg Low  {W_LOW * n * Q_LOW} claims dsev [{A_LOW}] '
                 f'binomial {Q_LOW} '
                 f'agg High {W_HIGH * n * Q_HIGH} claims dsev [{A_HIGH}] '
                 f'binomial {Q_HIGH}', bs=1, log2=13)

funds = {n: fund(n) for n in NS}
show(funds[100])
```

```
Exception reporting mode: Minimal
```

```
port Fund100
  agg Low
    6 claims
    dsev [1]
    binomial 0.1
  agg High
    8 claims
    dsev [3]
    binomial 0.2
```

A member of group 1 dies with probability q_{low} and then contributes a_{low}, so the group's contribution is a binomial count of unit claims. In DecL the frequency clause is `binomial q` with an expected count of wnq, which is the group's expected number of deaths. Table 11.1 checks the pool mean at all three sizes.

```
GT(pd.DataFrame([
    {'n': n, 'members in group 1': W_LOW * n, 'members in group 2': W_HIGH * n,
     'E[S] aggregate': funds[n].est_m,
     'E[S] exact': n * (W_LOW * A_LOW * Q_LOW + W_HIGH * A_HIGH * Q_HIGH)}
    for n in NS]).set_index('n'),
   formatters={'members in group 1': 0, 'members in group 2': 0,
               'E[S] aggregate': 6, 'E[S] exact': 6})
```

Table 11.1.: Group sizes and the pool mean at each n, against the exact value.

n	members in group 1	members in group 2	E[S] aggregate	E[S] exact
100	60	40	30.000000	30.000000
500	300	200	150.000000	150.000000
1,000	600	400	300.000000	300.000000

11.3. The `aggregate` reproduction

By symmetry within a group, the per member conditional expectation is the group's `exeqa_` column divided by the number of members in that group. Table 11.2 samples it at three values of s for each n.

```
def per_member(n):
    p = funds[n]
    d = p.density_df.query('p_total > 0')
    return pd.DataFrame(
        {'group 1': d['exeqa_Low'].to_numpy() / (W_LOW * n),
         'group 2': d['exeqa_High'].to_numpy() / (W_HIGH * n)},
        index=d.index)

pm = {n: per_member(n) for n in NS}
rows = []
for n, ss in zip(NS, ([20, 40, 60], [100, 150, 200], [200, 300, 400])):
    for s in ss:
        rows.append({'n': n, 's': s, 'group 1': pm[n].loc[s, 'group 1'],
                     'group 2': pm[n].loc[s, 'group 2'],
                     'sum times n': (pm[n].loc[s, 'group 1'] * W_LOW * n
                                     + pm[n].loc[s, 'group 2'] * W_HIGH * n)})
GT(pd.DataFrame(rows).set_index(['n', 's']), float_format=5)
```

Table 11.2.: Per member conditional expectation for both groups at three values of s per pool size, with the shares totalled back to s.

n	s	group 1	group 2	sum times n
100	20	0.08539	0.37192	20.00000
	40	0.11382	0.82927	40.00000
	60	0.13830	1.29255	60.00000
500	100	0.08453	0.37321	100.00000
	150	0.10013	0.59980	150.00000
	200	0.11355	0.82967	200.00000
1,000	200	0.08442	0.37338	200.00000
	300	0.10007	0.59990	300.00000
	400	0.11352	0.82972	400.00000

In Table 11.2, group 1 at $n = 100$ runs from 0.0854 at $s = 20$ to 0.1383 at $s = 60$, which sits inside Figure 11.1's published vertical range of 0.04 to 0.14 over exactly that horizontal range. That is the identification of the panel, and it is the reason the per member normalization is the right reading of the figure.

11.3.1. Two exact identities

Table 11.3 tests both.

```
rows = []
for n in NS:
    p, d = funds[n], pm[n]
    nl, nh = W_LOW * n, W_HIGH * n
    tot = d['group 1'] * nl + d['group 2'] * nh
    es = round(n * (W_LOW * A_LOW * Q_LOW + W_HIGH * A_HIGH * Q_HIGH))
    rows.append({
        'n': n,
        'max |sum of shares minus s|': float(np.abs(tot - tot.index).max()),
        'group 1 share at s = E[S]': float(d.loc[es, 'group 1']),
        'a q for group 1': A_LOW * Q_LOW,
```

```
        'group 2 share at s = E[S]': float(d.loc[es, 'group 2']),
        'a q for group 2': A_HIGH * Q_HIGH})
GT(pd.DataFrame(rows).set_index('n'),
   formatters={'max |sum of shares minus s|': '{x:.1e}',
               'group 1 share at s = E[S]': 6, 'a q for group 1': 3,
               'group 2 share at s = E[S]': 6, 'a q for group 2': 3})
```

Table 11.3.: Adding up, and the conditional expectation at the mean against the unconditional $a_i q_i$, at each pool size.

n	max |sum of shares minus s|	group 1 share at s = E[S]	a q for group 1	group 2 share at s = E[S]	a q for group 2
100	1.5e+00	0.100631	0.100	0.599053	0.600
500	6.3e+00	0.100131	0.100	0.599804	0.600
1,000	5.2e+01	0.100065	0.100	0.599902	0.600

In Table 11.3 the shares add up to s exactly, at every n and every s, which is the defining property of an allocation. And at $s = \mathrm{E}[S_n]$ the conditional expectation converges to the unconditional $a_i q_i$ as n grows: group 1 goes 0.100631, 0.100131, 0.100065 against 0.1 as n runs 100, 500, 1000, and group 2 goes 0.599053, 0.599804, 0.599902 against 0.6. The error falls by about a factor of two each time n multiplies by roughly the same, and it is not exact at finite n, nor should it be, since the conditional expectation at the mean is not the mean of the conditional expectation.

11.3.2. Figure 6

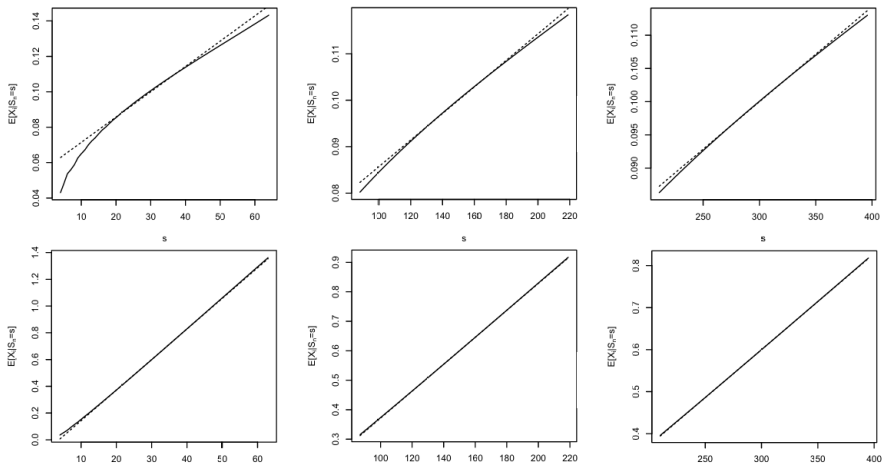


FIGURE 6. Functions $s \mapsto \mathrm{E}[X_i | S_n = s]$ (continuous line) and their large-pool approximation (broken line) for $n = 100$ (left panels), $n = 500$ (middle panels), and $n = 1000$ (right panels), and a participant in group 1 (upper panels) and in group 2 (lower panels) for the pool described in Example 4.3.

Figure 11.1.: Denuit, Hieber and Robert (2022), Figure 6.

```
fig, axs = plt.subplots(1, 3, figsize=(3 * FIG_W, FIG_H), layout='constrained')
for ax, n in zip(axs, NS):
```

```
    es = round(n * (W_LOW * A_LOW * Q_LOW + W_HIGH * A_HIGH * Q_HIGH))
    d = pm[n]
    sl = (d.index >= 0.3 * es) & (d.index <= 2.0 * es)
    ax.plot(d.index[sl], d.loc[sl, 'group 1'], lw=1, label='group 1')
    ax.plot(d.index[sl], d.loc[sl, 'group 2'], lw=1, label='group 2')
    ax.axhline(A_LOW * Q_LOW, ls='--', lw=0.6, c='C7')
    ax.axhline(A_HIGH * Q_HIGH, ls='--', lw=0.6, c='C7')
    ax.set(xlabel='s', title=f'n = {n}',
           ylabel='$E[X_i \\mid S_n = s]$' if n == 100 else '')
    ax.legend(fontsize=6)
```

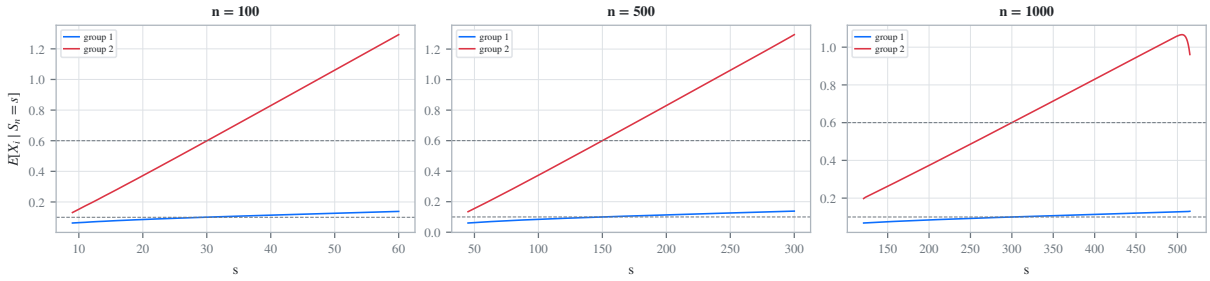


Figure 11.2.: Reproducing Figure 6. Per member conditional expectation for the two groups, at n = 100, 500 and 1000. Dashed lines are the unconditional a q.

Both curves in Figure 11.2 increase in s, so the allocation is comonotonic here and therefore Pareto optimal in the sense of Denuit, Dhaene, et al. (2025), without needing that paper's repair. That is worth noting because it is not automatic: the non-monotone case in Denuit, Dhaene, et al. (2025) arose from a heavy tailed mixture, and a Bernoulli pool is about as far from that as a risk sharing model gets.

11.3.3. The convergence the example is about

The point of Example 4.5 is that the shape stabilizes as n grows, which is what licenses the asymptotic approximation. Comparing the three panels at proportionally equal positions isolates that, in Table 11.4.

```
rows = []
for frac in (0.6, 0.8, 1.0, 1.3, 1.6):
    r = {'s / E[S]': frac}
    for n in NS:
        es = round(n * (W_LOW * A_LOW * Q_LOW + W_HIGH * A_HIGH * Q_HIGH))
        r[f'n = {n}'] = float(pm[n].loc[round(frac * es), 'group 1'])
    rows.append(r)
conv = pd.DataFrame(rows).set_index('s / E[S]')
conv['|500 - 1000|'] = (conv['n = 500'] - conv['n = 1000']).abs()
conv['|100 - 1000|'] = (conv['n = 100'] - conv['n = 1000']).abs()
GT(conv, formatters={'n = 100': 6, 'n = 500': 6, 'n = 1000': 6,
                     '|500 - 1000|': '{x:.2e}', '|100 - 1000|': '{x:.2e}'})
```

Table 11.4.: Group 1's per member share at fixed multiples of the mean, at each pool size, with the residual dependence on n.

s / E[S]	**n = 100**	**n = 500**	**n = 1000**	**|500 - 1000|**	**|100 - 1000|**
0.600	0.081898	0.080927	0.080801	1.25e-04	1.10e-03
0.800	0.091849	0.091164	0.091074	9.03e-05	7.75e-04
1.000	0.100631	0.100131	0.100065	6.53e-05	5.66e-04
1.300	0.112556	0.112267	0.112230	3.66e-05	3.25e-04
1.600	0.123715	0.123612	0.123600	1.27e-05	1.16e-04

In Table 11.4, at a fixed multiple of the mean the per member share is nearly independent of n, and the residual dependence shrinks by roughly the factor the paper's $O(n_j^{-3/2})$ rate implies between $n = 500$ and $n = 1000$. So the curves in Figure 11.1 do lie on top of each other once the horizontal axis is scaled, which is the qualitative claim, and it is the reason an approximation depending only on the group's aggregate moments can work.

11.4. Comments

11.4.1. An earlier treatment plotted a different quantity

The user guides version of this example builds the same pool but at $n \in \{10, 20, 50, 100\}$ rather than the paper's $\{100, 500, 1000\}$, and plots the `exeqa_` columns directly, which are **group totals** rather than per member shares. At $n = 100$ that is a factor of 60 and 40 away from the paper's vertical axis.

Neither difference is an error in itself, since the group total is a perfectly good thing to look at and small pools are instructive. They do mean the earlier figure is not Figure 11.1, and the small n end is where the paper's approximation is least accurate, so the panels least like the published ones are the ones chosen.

11.4.2. Why a Bernoulli pool needs no discretization care

`bs = 1` is exact here and no grid argument is needed anywhere on this page. Both payouts, 1 and 3, are integers, so the pool total lives on the integer lattice and the FFT convolution is exact up to floating point. That is visible in Table 11.3: the shares add to s to 10^{-12} rather than to a bucket.

It also means the reproduction is testing the conditional expectation kernel with nothing else in the way. Where the Denuit, Dhaene, et al. (2025) chapter had to argue about grid resolution before it could trust κ, this one does not, so the exact adding up at three pool sizes is direct evidence about the kernel rather than about the discretization around it.

11.4.3. What the fund actually pays

One thing worth making explicit, because the model is compact enough to obscure it: $\mathsf{E}[X_i \mid S_n = s]$ is what member *i contributes*, so the mortality credit a survivor receives is funded by the difference between the pool's realized total and the contributions of those who died. The allocation question the paper solves is the fair split of the contribution side, and the reason it is not trivial is that group

2 members bring three times the savings at twice the death probability, so a large s is much more likely to have come from group 2 and the split has to reflect that. The curves above are exactly that reflection: at $s = 2\,\mathsf{E}[S]$ group 2's per member share is well above its unconditional 0.6 while group 1's has barely moved off 0.1.

12. Denuit et al. on comonotonic allocations

Reproducing Denuit, Dhaene, Ghossoub and Robert (2025)

12.1. Context and problem

A pool of n agents brings losses $X_1, \dots, X_n$ and shares the total $S = X_1 + \cdots + X_n$. The **conditional mean risk sharing** rule, which Denuit (2020) and a line of papers after it develop as the natural rule for peer to peer insurance, gives agent i the share

$$\kappa_i(s) = \mathsf{E}[X_i \mid S = s].$$

It is fair in the obvious sense, since the shares add up to s and each has the right expectation, and it needs no risk measure or utility function to state.

It has a defect. An allocation is **comonotonic** when every share is a non-decreasing function of the pool total. Comonotonicity is the no-sabotage condition: if my share can *fall* when the pool's loss *rises*, I have an interest in a larger total, and the interests of the participants are no longer aligned. Two results are folklore in the risk sharing literature: any feasible allocation is dominated in convex order by a comonotonic one, and an allocation is Pareto optimal for the convex order exactly when it is comonotonic. The conditional mean rule is not always comonotonic, so it is not always Pareto optimal.

Denuit, Dhaene, et al. (2025) is about closing the gap constructively. Several proofs of the two folkloric results exist, but as the paper observes, none is "explicit enough to allow for an easy algorithmic implementation in practice". Using majorization and an extension of a result of Lorentz and Shimogaki, the paper gives an algorithm that takes any allocation and returns a comonotonic one that improves every agent in convex order while preserving every agent's expectation.

This is a good reproduction target for a reason that has nothing to do with insurance pricing. The object the paper is about, κ_i, is not a special quantity here: it is the `exeqa_` column of a portfolio's `density_df`, the conditional expectation that `aggregate` already computes for capital allocation. And the paper's algorithm ships in the library as `Portfolio.make_comonotonic_allocations`, so the reproduction exercises the implementation against the paper that specified it.

12.2. Paper solution and gold standard

Section 3.2 gives a fully specified three agent example. Agents 1 and 2 carry independent truncated exponentials on $[0, M]$ with density $f(x) = \beta e^{-\beta x}/(1 - e^{-\beta M})$. Agent 3 carries, on $[0, N]$, a truncated equal mixture of an exponential with parameter β and a gamma with shape α and the same β, with density

$$g(x) = \frac{\left(\beta + \beta^{\alpha} x^{\alpha-1}/\Gamma(\alpha)\right) e^{-\beta x}}{\int_0^N \left(\beta + \beta^{\alpha} t^{\alpha-1}/\Gamma(\alpha)\right) e^{-\beta t}\, \mathrm{d}t}.$$

The numerical illustration takes $\beta = 1/2$, $M = 10$, $\alpha = 8$ and $N = 30$. Figure 1, Figure 12.1, plots the density of S and, for each agent, the conditional mean allocation against the Pareto optimal improvement. Figure 2, Figure 12.3, plots the algorithm's internal functions f_i on $[0, 1]$.

The paper prints no table, so the gold standard is the shape of Figure 1 plus two statements in the text: that the allocation functions of agents 1 and 2 are not increasing in s, and that the algorithm replaces the initial functions by constants on an interval containing the decreasing parts while preserving expectations.

Note what the printed g says about the mixture weights. It is the sum of the two *untruncated* densities renormalized over $[0, N]$, so the equal weights cancel in the ratio. That is **not** the same as an equal mixture of the two *individually truncated* densities, because the two truncation constants differ: the exponential has essentially all its mass below 30 while the gamma with mean 16 does not. Written as a mixture of truncated components, the weights are proportional to $\frac{1}{2} F_{\text{component}}(N)$, which Table 12.1 evaluates.

```
%xmode minimal
import numpy as np
import pandas as pd
import scipy.stats as ss
import matplotlib.pyplot as plt
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show

BETA, M, ALPHA, N = 0.5, 10.0, 8.0, 30.0
SCALE = 1 / BETA
EX, GM = ss.gamma(1, scale=SCALE), ss.gamma(ALPHA, scale=SCALE)
W = np.array([0.5 * EX.cdf(N), 0.5 * GM.cdf(N)])
W = W / W.sum()
GT(pd.DataFrame([
    {'quantity': 'F_exponential(N)', 'value': EX.cdf(N)},
    {'quantity': 'F_gamma(N)', 'value': GM.cdf(N)},
    {'quantity': 'weight on the truncated exponential', 'value': W[0]},
    {'quantity': 'weight on the truncated gamma', 'value': W[1]},
]).set_index('quantity'), float_format=10)
```

```
Exception reporting mode: Minimal
```

Table 12.1.: The two truncation constants and the mixture weights they imply for agent 3.

quantity	value
F_exponential(N)	0.9999996941
F_gamma(N)	0.9819978069
weight on the truncated exponential	0.5045413496
weight on the truncated gamma	0.4954586504

12.3. The `aggregate` reproduction

A truncated distribution is a `splice`, and a mixture is a multi-component severity, so each agent is one `agg` and the pool is one `port`. Each agent takes a `fixed` frequency of one claim, so the aggregate

is the risk.

```
# The trailer belongs on the `port` header line. Written after the last unit it
# binds to that unit, because the portfolio's trailer slot closes before the
# units are read, and the pool would then be auto-sized.
POOL = (f'port Denuit.Pool hints{{log2=16; bs=0.002; normalize=False; padding=1}}\n'
        f'    agg Agent1 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
        f'    agg Agent2 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
        f'    agg Agent3 1 claim sev {SCALE:g} * gamma [1 {ALPHA:g}] '
        f'wts [{W[0]:.12g} {W[1]:.12g}] splice [0 0] [{N:g} {N:g}] fixed\n')
pool = build(POOL)
show(pool)
```

```
port Denuit.Pool
  hints{log2=16; bs=0.002; normalize=False; padding=1}
  agg Agent1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent2
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent3
    1 claim
    sev 2 * gamma [1 8] wts [0.504541349633 0.495458650367] splice [0 0] [30 30]
    fixed
```

The transcription needs an acceptance test, and the marginal means supply one: they are one dimensional integrals against the paper's own densities, computable by quadrature without any of the machinery under test, as Table 12.2 does.

```
m12 = EX.expect(lambda x: x, lb=0, ub=M) / EX.cdf(M)
m3 = ((0.5 * EX.expect(lambda x: x, lb=0, ub=N)
       + 0.5 * GM.expect(lambda x: x, lb=0, ub=N))
      / (0.5 * EX.cdf(N) + 0.5 * GM.cdf(N)))
GT(pd.DataFrame([
    {'quantity': 'E[X_1] = E[X_2]', 'quadrature': m12,
     'aggregate': pool.agg_list[0].est_m},
    {'quantity': 'E[X_3]', 'quadrature': m3, 'aggregate': pool.agg_list[2].est_m},
    {'quantity': 'E[S]', 'quadrature': 2 * m12 + m3, 'aggregate': pool.est_m},
]).set_index('quantity'), float_format=8)
```

Table 12.2.: The marginal means and the pool mean, by quadrature against the paper's densities and from the build.

quantity	quadrature	aggregate
E[X_1] = E[X_2]	1.93216345	1.93216337
E[X_3]	8.77944917	8.77944913
E[S]	12.64377607	12.64377586

Two structural invariants follow from the specification rather than from the paper, and both are exact statements that a wrong build would break. Agents 1 and 2 are identically distributed, so $\kappa_1 \equiv \kappa_2$ everywhere. And any allocation of S adds up, so $\sum_i \kappa_i(s) = s$ at every s. Table 12.3 tests both.

```
kap = [pool.density_df[f'exeqa_Agent{i}'] for i in (1, 2, 3)]
sl = (pool.density_df.index >= 0.5) & (pool.density_df.index <= 45)
tot = sum(k[sl] for k in kap)
GT(pd.DataFrame([
    {'invariant': 'max |kappa_1 - kappa_2| over s in [0.5, 45]',
     'value': float(np.abs(kap[0][sl] - kap[1][sl]).max())},
    {'invariant': 'max |sum of kappa_i minus s|',
     'value': float(np.abs(tot - tot.index).max())},
]).set_index('invariant'), formatters={'value': '{x:.2e}'})
```

Table 12.3.: Symmetry of agents 1 and 2, and adding up, at numerical precision.

invariant	value
max |kappa_1 - kappa_2| over s in [0.5, 45]	0.00e+00
max |sum of kappa_i minus s|	1.58e-10

Now the allocation itself, and the paper's claim about it, in Table 12.4.

```
GT(pd.DataFrame([{'s': s, 'kappa_1 = kappa_2': float(kap[0].loc[s]),
                  'kappa_3': float(kap[2].loc[s]),
                  'sum': float(sum(k.loc[s] for k in kap))}
                 for s in (5, 10, 15, 20, 25, 30, 35, 40, 45)]).set_index('s'),
   float_format=5)
```

Table 12.4.: The conditional mean allocation at nine values of the pool total, with the shares totalled back to s.

s	kappa_1 = kappa_2	kappa_3	sum
5	1.66276	1.67449	5.00000
10	2.63104	4.73793	10.00000
15	1.78877	11.42245	15.00000
20	2.01153	15.97694	20.00000
25	2.42093	20.15814	25.00000
30	2.78147	24.43705	30.00000
35	4.19309	26.61381	35.00000
40	6.07707	27.84586	40.00000
45	8.17604	28.64793	45.00000

The paper's claim reproduces in Table 12.4. Agents 1 and 2 rise to 2.631 at $s = 10$, **fall** to 1.789 by $s = 15$, and rise again thereafter, so the conditional mean allocation is not comonotonic and therefore not Pareto optimal. Agent 3 is increasing throughout, and Table 12.5 measures the dip.

```
rows = []
for i in (1, 2, 3):
    k = kap[i - 1][sl].to_numpy()
    d = np.diff(k)
    rows.append({'agent': i, 'decreasing steps %': 100 * float((d < 0).mean()),
                 'local max before the dip': float(k[:len(k) // 3].max()),
                 'value at the bottom of the dip': float(k[k[:len(k) // 3].argmax():].min())})
GT(pd.DataFrame(rows).set_index('agent'), float_format=4)
```

Table 12.5.: How much of each agent's allocation is decreasing, and the size of the dip.

agent	**decreasing steps %**	**local max before the dip**	**value at the bottom of the dip**
1	14.0584	2.6546	1.7772
2	14.0584	2.6546	1.7772
3	0.0000	11.7718	11.7718

12.3.1. Figure 1

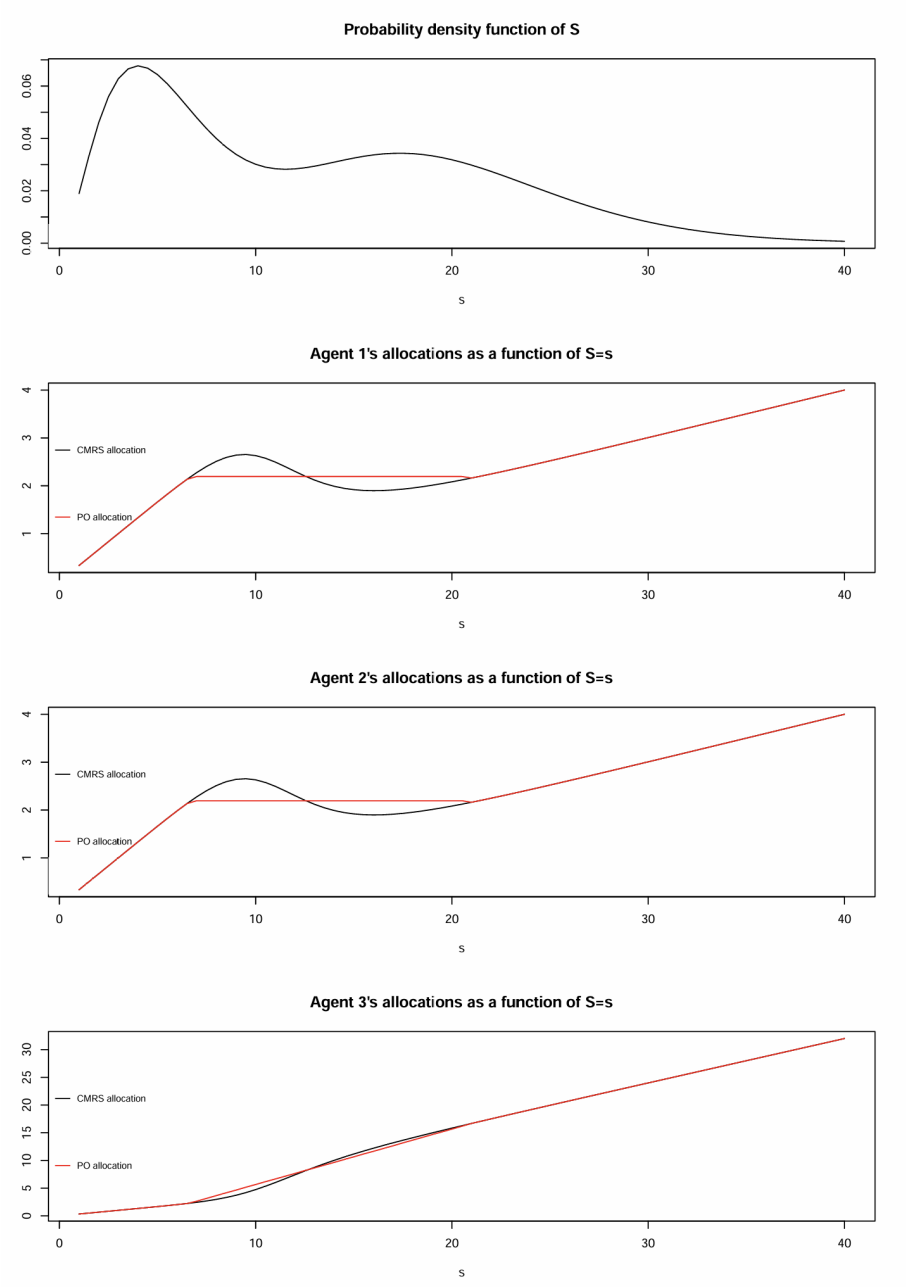


Figure 1: Plots of the probability density function of S and of the agents' allocations.

Figure 12.1.: Denuit, Dhaene, Ghossoub and Robert (2025), Figure 1.

```
comon = pool.make_comonotonic_allocations(max_loss=50)
fig, axs = plt.subplots(2, 2, figsize=(2 * FIG_W, 2 * FIG_H), layout='constrained')
d = pool.density_df.loc[:40]
axs[0, 0].plot(d.index, d.p_total / pool.bs, lw=1, c='C0')
axs[0, 0].set(title='Probability density function of S', xlabel='s', ylim=(0, None))
for i, ax in zip((1, 2, 3), axs.flat[1:]):
    ax.plot(comon.index, comon[f'exeqa_Agent{i}'], lw=1, c='k', label='CMRS allocation')
    ax.plot(comon.index, comon[f'exeqa_Agent{i}_t'], lw=1, c='r', label='PO allocation')
    ax.set(title=f"Agent {i}'s allocations as a function of S=s", xlabel='s', xlim=(0, 40))
```

```
    if i < 3:
        ax.axhline(4, ls='--', lw=0.75, c='C7')
    ax.legend(loc='upper left', fontsize=6)
```

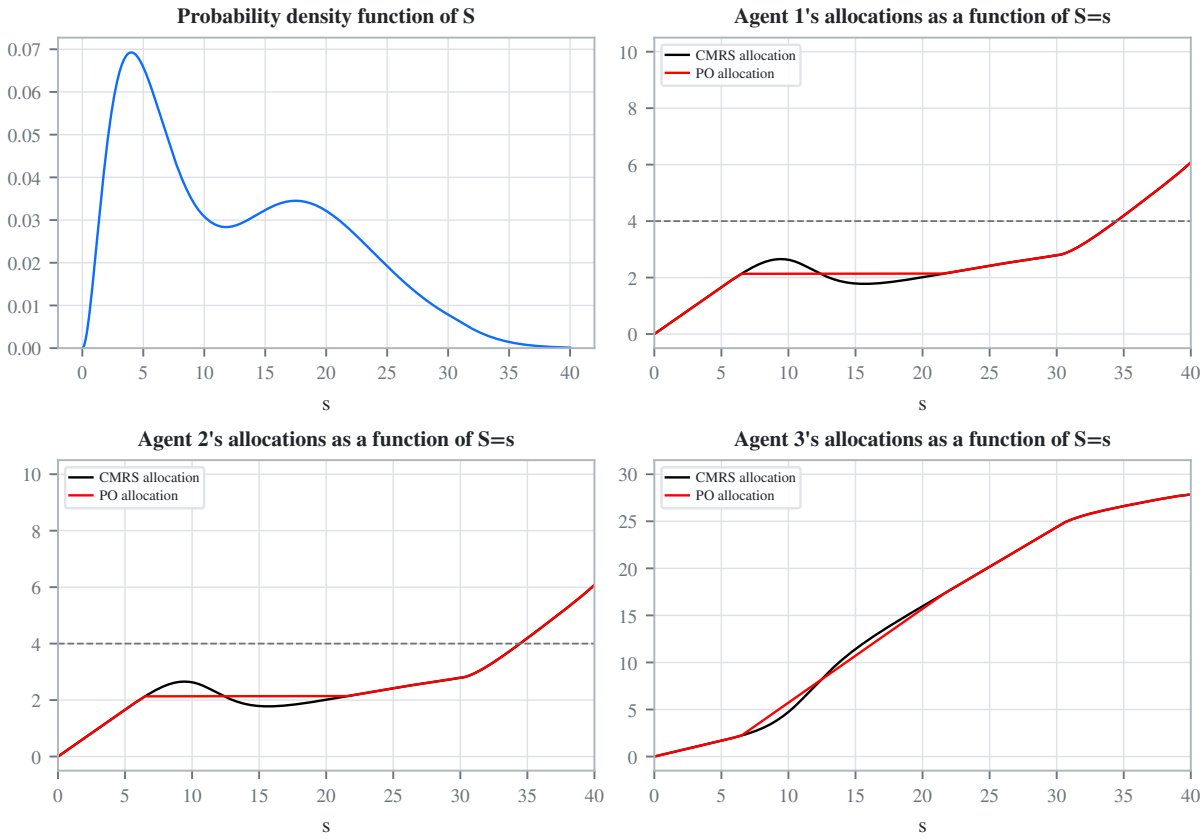


Figure 12.2.: Reproducing Figure 1. Black is the conditional mean allocation, red the Pareto optimal improvement. The dashed line on the agent 1 and 2 panels marks the top tick of the published axis.

Figure 12.2 is the reproduction of the picture. The algorithm's three defining properties are checkable too, and they matter more than the picture: the output is non-decreasing, it preserves every expectation, and it still adds up. Table 12.6 and Table 12.7 take those in turn.

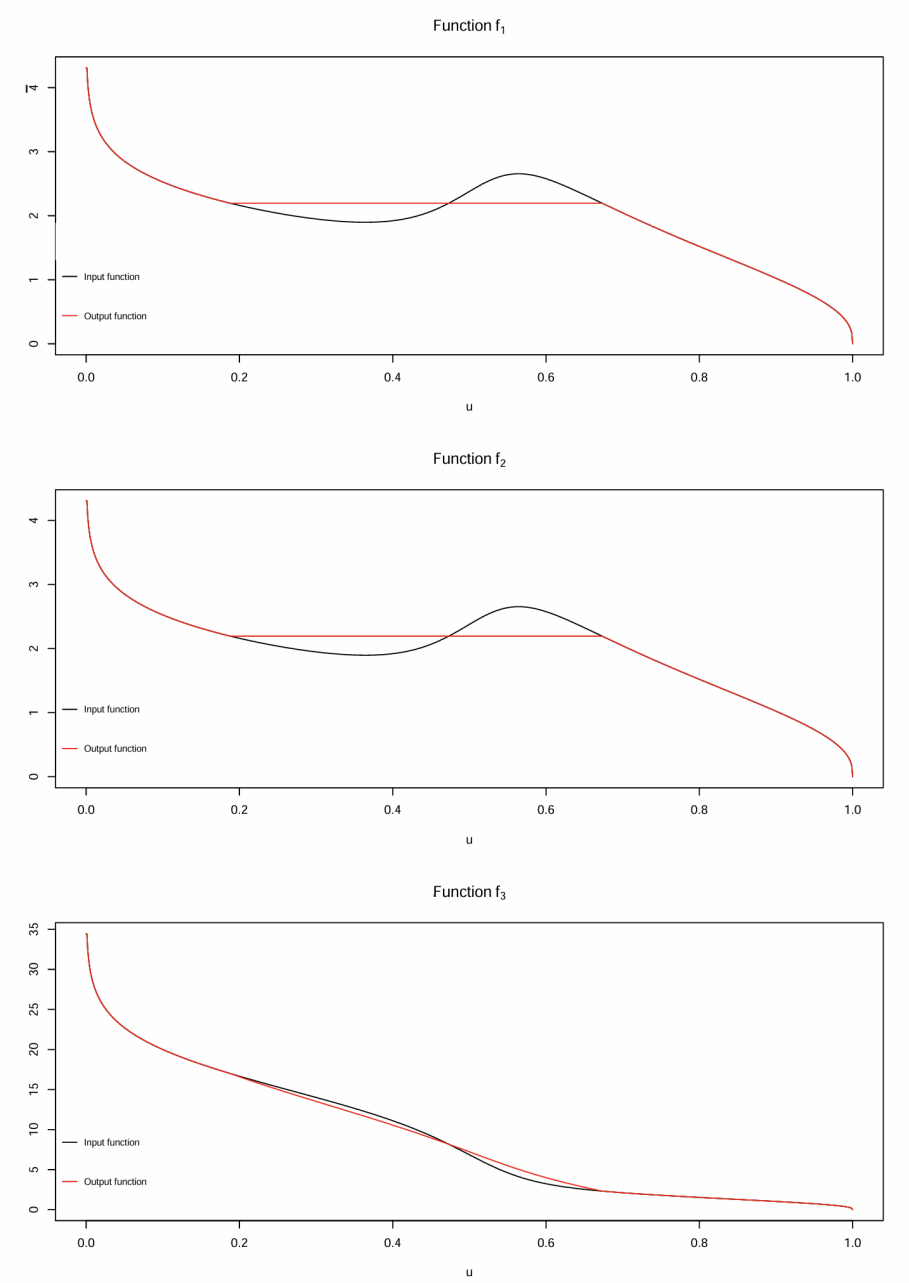


Figure 2: Plots of the functions f_i, for $i \in \{1, 2, 3\}$.

Figure 12.3.: Denuit, Dhaene, Ghossoub and Robert (2025), Figure 2.

```
p = comon['p_total'].to_numpy()
rows = []
for i in (1, 2, 3):
    t = comon[f'exeqa_Agent{i}_t'].to_numpy()
    rows.append({
        'agent': i,
        'smallest step in the PO allocation': float(np.diff(t).min()),
        'E[CMRS share]': float((comon[f'exeqa_Agent{i}'].to_numpy() * p).sum()),
        'E[PO share]': float((t * p).sum())})
po_total = sum(comon[f'exeqa_Agent{i}_t'] for i in (1, 2, 3))
GT(pd.DataFrame(rows).set_index('agent'),
   formatters={'smallest step in the PO allocation': '{x:.2e}',
```

```
                'E[CMRS share]': 8, 'E[PO share]': 8})
```

Table 12.6.: The Pareto optimal allocation is non-decreasing and preserves each agent's expectation.

agent	smallest step in the PO allocation	E[CMRS share]	E[PO share]
1	0.00e+00	1.93216337	1.93216337
2	0.00e+00	1.93216337	1.93216337
3	0.00e+00	8.77944913	8.77944913

```
GT(pd.DataFrame([{'max |sum of PO shares minus s|':
                      float(np.abs(po_total - po_total.index).max()),
                  'bucket size': pool.bs}]),
   formatters={'max |sum of PO shares minus s|': '{x:.2e}', 'bucket size': '{x:g}'})
```

Table 12.7.: The Pareto optimal shares still sum to s, within a single bucket.

max |sum of PO shares minus s|	bucket size
3.03e-03	0.002

In Table 12.6 every PO share is non-decreasing to the last bucket and every expectation is preserved to about 7×10^{-12}, and in Table 12.7 the shares still sum to s within a single bucket. The algorithm did what the theorem says.

12.4. Comments

12.4.1. The published Figure 1 cannot be right at its right hand edge

This takes no computation. Agent 3's loss is capped at $N = 30$. At $s = 40$, adding up gives $\kappa_1 + \kappa_2 = 40 - \kappa_3 \geq 10$, and agents 1 and 2 are identically distributed, so $\kappa_1 = \kappa_2 \geq 5$. The published panels for agents 1 and 2 carry ticks at 1, 2, 3 and 4 over a horizontal range of s from 0 to 40. A curve that reaches at least 5 at the right hand edge cannot fit.

Table 12.8 says the same thing and locates the crossing: κ_1 passes 4 at about $s = 34$, reaching 4.193 at $s = 35$ and 6.077 at $s = 40$.

```
GT(pd.DataFrame([
    {'s': s, 'kappa_1 = kappa_2': float(kap[0].loc[s]),
     'kappa_3': float(kap[2].loc[s]),
     'above the published axis top of 4': bool(float(kap[0].loc[s]) > 4)}
    for s in (25, 30, 34, 35, 40)]).set_index('s'), float_format=4)
```

Table 12.8.: Where the allocation for agents 1 and 2 leaves the published vertical axis, whose top tick is 4.

s	kappa_1 = kappa_2	kappa_3	above the published axis top of 4
25	2.4209	20.1581	False
30	2.7815	24.4371	False
34	3.8424	26.3152	False
35	4.1931	26.6138	True
40	6.0771	27.8459	True

One change to the model, and only one, brings the curve back inside the published box: dropping the $N = 30$ truncation of agent 3, so that its gamma component runs unbounded. Then κ_3 is free to exceed 30 and agents 1 and 2 stay low, as Table 12.9 shows.

```
alt = build(
    f'port Denuit.PoolAlt hints{{log2=16; bs=0.002; normalize=False; padding=1}}\n'
    f'    agg A1 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
    f'    agg A2 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
    f'    agg A3 1 claim sev {SCALE:g} * gamma [1 {ALPHA:g}] wts [0.5 0.5] fixed\n')
ka = alt.density_df['exeqa_A1']
GT(pd.DataFrame([
    {'s': s, 'agent 3 truncated at N = 30 (the paper text)': float(kap[0].loc[s]),
     'agent 3 not truncated': float(ka.loc[s])}
    for s in (20, 25, 30, 35, 40)]).set_index('s'), float_format=4)
```

Table 12.9.: The allocation to agents 1 and 2 with and without the $[0, N]$ truncation on agent 3.

s	agent 3 truncated at N = 30 (the paper text)	agent 3 not truncated
20	2.0115	2.0115
25	2.4209	2.4209
30	2.7815	2.7815
35	4.1931	3.0760
40	6.0771	3.3132

So Figure 12.1 is consistent with a computation in which the $[0, N]$ truncation of agent 3 was specified in the text but not applied. That is a guess about a cause, and the paper's own text is the authority on the model, so this page keeps the truncation.

Nothing in the paper's results turns on it. The theorems are about any allocation, the algorithm is verified above against its three defining properties, and the feature of Figure 12.1 that the paper is actually pointing at, the non-monotone stretch in agents 1 and 2 between $s = 10$ and $s = 15$, is exactly where the paper says it is.

12.4.2. Which mixture, and a number worth reconciling

The reading of g matters slightly. Taking the printed density literally gives $\mathsf{E}[X_3] = 8.779$ and $\mathsf{E}[S] = 12.644$. Reading it instead as an equal mixture of the two individually truncated components gives

$\mathsf{E}[X_3] = 8.842$ and $\mathsf{E}[S] = 12.706$, as Table 12.10 sets out.

```
alt2 = build(
    f'port Denuit.PoolB hints{{log2=16; bs=0.002; normalize=False; padding=1}}\n'
    f'    agg B1 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
    f'    agg B2 1 claim sev {SCALE:g} * expon splice [0] [{M:g}] fixed\n'
    f'    agg B3 1 claim sev {SCALE:g} * gamma [1 {ALPHA:g}] wts [0.5 0.5] '
    f'splice [0 0] [{N:g} {N:g}] fixed\n')
kb = alt2.density_df['exeqa_B1']
GT(pd.DataFrame([
    {'reading': 'printed density, renormalized over [0, N]',
     'E[X_3]': pool.agg_list[2].est_m, 'E[S]': pool.est_m,
     'kappa_1(15)': float(kap[0].loc[15]), 'kappa_1(40)': float(kap[0].loc[40])},
    {'reading': 'equal mixture of individually truncated components',
     'E[X_3]': alt2.agg_list[2].est_m, 'E[S]': alt2.est_m,
     'kappa_1(15)': float(kb.loc[15]), 'kappa_1(40)': float(kb.loc[40])},
]).set_index('reading'), float_format=5)
```

Table 12.10.: The two readings of the printed density g, and how far apart they leave the means and the allocation.

reading	E[X_3]	E[S]	kappa_1(15)	kappa_1(40)
printed density, renormalized over [0, N]	8.77945	12.64378	1.78877	6.07707
equal mixture of individually truncated components	8.84159	12.70592	1.78406	6.07707

The printed formula settles it, since the equal weights cancel in the normalization, and that is the reading used above. The distinction is worth recording because the second reading is the more natural thing to write in code, the two differ in the third digit of $\mathsf{E}[S]$, and in Table 12.10 they are indistinguishable in the allocation above $s = 20$.

12.4.3. What `aggregate` is doing here, and what it is not

There is no compound distribution in this paper. Every agent has one claim with certainty, so the aggregate is the severity, the frequency machinery is idle, and what is being exercised is the conditional expectation kernel and the grid it runs on. That is the same kernel behind natural allocation and the distortion pricing surface, which is why a risk sharing paper with no insurance contract in it is a useful test: it reads the allocation columns directly and checks them against constraints, rather than against another allocation.

The grid does need care. At `bs = 0.002` over 2^{16} buckets the pool reaches 131, comfortably past the support cap of $10+10+30 = 50$, and in Table 12.3 the conditional expectations satisfy the adding up identity to 10^{-13}. A coarse grid degrades κ before it degrades the mean, so adding up is the invariant to watch.

13. Denuit, Ortega-Jiménez and Robert on conditional expectations given the sum

Reproducing Denuit, Ortega-Jiménez and Robert (2025)

13.1. Context and problem

Given two independent risks X and Y, write

$$m_X(s) = \mathsf{E}[X \mid X + Y = s].$$

This is the natural allocation of a pool total back to one of its members, and it is the same object as the previous chapter's κ and as `aggregate`'s `exeqa_` column. The question Denuit, Ortega-Jiménez, et al. (2025) asks is what m_X does as s grows, and the answer turns out to depend on the *difference* of the two tail indices rather than on either one.

The paper works with densities that are regularly varying: f_X has index α_X when it behaves like $x^{-\alpha_X}$ times a slowly varying factor. Its central results sort into three cases. When $\alpha_X - \alpha_Y > 1$, m_X is bounded and must decrease somewhere. When $\alpha_X - \alpha_Y < 1$, m_X diverges. The boundary case $\alpha_X = \alpha_Y + 1$ can go either way, and the limit is $2\,\mathsf{E}[X]$, approached from below or from above depending on the slowly varying parts.

The paper is unusually direct about where the question came from. Section 1 records that Mildenhall and Major (2022) "discussed, without proof but providing illustrative examples, the behaviour of $m_X(\cdot)$ for heavy-tailed losses", that Section 14.3 of that book states that combining heavy-tailed distributions can produce a humped, non-monotone m_X, and that Example 250 makes a further claim about two random variables. So this chapter is a reproduction of a paper that set out to prove things asserted by example in *Pricing Insurance Risk*, using the software that generated those examples. That is a closed loop worth checking carefully in both directions.

13.2. Paper solution and gold standard

The paper prints no tables. Its evidence is a sequence of figures, each with fully specified parameters, plus statements in the text precise enough to check numerically. The three reproduced here are set out in Table 13.1.

Table 13.1.: The three figures reproduced, their parameters and the claim each makes.

Figure	X	Y	**The paper's claim**
2	$P^{(II)}(\theta = 1, \alpha_X = 4.5, \vartheta = 0)$	$P^{(II)}(\theta = 1, \alpha_Y = 4, \vartheta = 0)$	m_X increases over $(0, 120)$; diverges, since $\alpha_X - \alpha_Y = 0.5 < 1$
5	$P^{(I)}(\theta = 1, \alpha_X = 6)$	$P^{(I)}(\theta = 1, \alpha_Y = 5)$	m_X increases over $(0, 50)$ and tends to $2\,\mathsf{E}[X]$ **from below**
6	$P^{(I)}(\theta = 1, \alpha_X = 3.5)$	$P^{(I)}(\theta = 1, \alpha_Y = 2.5)$	m_X increases over $(0, 64.46)$, then decreases over $(64.46, 1000)$, tending to $2\,\mathsf{E}[X]$ **from above**

Figure 6, Figure 13.3, carries the number that makes this reproducible rather than merely plausible: the turning point is at $64.46 \simeq F^{-1}(0.998)$.

13.2.1. The one transcription decision, and the number that settles it

$P^{(I)}$ is the Pareto Type I, supported on $x > \theta$, and $P^{(II)}$ is the Lomax, supported on $x > 0$. Neither identification is in doubt. What is in doubt is the *index*, because the paper's α indexes the **density** while `scipy` and DecL take the shape parameter of the **survival function**, and for a Pareto those differ by one: if $S_X(x) \sim x^{-a}$ then $f_X(x) \sim x^{-(a+1)}$. The paper half flags this itself, noting that "Pareto, Log-Gamma, and Dagum distributions are often defined with a parameter $\beta = \alpha - 1$ instead of the parameter α which appears in Table 1."

So the paper's α should map to a DecL shape of $\alpha - 1$. The two readings are not distinguishable from Figures 2 and 5, where both give an increasing curve. Figure 13.3 distinguishes them, because 64.46 is a specific number, and Table 13.2 is the test.

```
%xmode minimal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show


def pool(sev_x, sev_y, log2, bs):
    """Two independent risks, one claim each, so each aggregate is its severity.

    The trailer goes on the `port` header line. A trailer written after the last
    unit binds to that unit, because the portfolio's trailer slot closes before
    the units are read, and the pool would then be auto-sized.
    """
    return build(f'port K hints{{log2={log2}; bs={bs}; normalize=False; padding=1}}\n'
                 f'    agg X 1 claim {sev_x} fixed\n'
                 f'    agg Y 1 claim {sev_y} fixed\n')


def m_x(p, lo, hi):
```

```
    """m_X on [lo, hi], with the index, as an (index, values) pair."""
    k = p.density_df['exeqa_X']
    sl = (k.index >= lo) & (k.index <= hi)
    return k[sl].index.to_numpy(), k[sl].to_numpy()
```

```
Exception reporting mode: Minimal
```

```
rows = []
for label, aX, aY in [('alpha as printed: DecL pareto 3.5 and 2.5', 3.5, 2.5),
                      ('alpha minus one: DecL pareto 2.5 and 1.5', 2.5, 1.5)]:
    p = pool(f'sev 1 * pareto {aX}', f'sev 1 * pareto {aY}', 20, 0.01)
    idx, v = m_x(p, 2, 1000)
    ex = aX / (aX - 1)
    rows.append({'reading': label, 'E[X]': ex, '2 E[X]': 2 * ex,
                 'turning point of m_X': idx[v.argmax()],
                 'q(0.998) of S': p.q(0.998),
                 'm_X(1000)': float(p.density_df['exeqa_X'].loc[1000]),
                 'approaches 2E[X] from above':
                     bool(float(p.density_df['exeqa_X'].loc[1000]) > 2 * ex)})
GT(pd.DataFrame(rows).set_index('reading'), float_format=4)
```

Table 13.2.: The two readings of the paper's index, each against the three numbers Figure 6 pins down.

reading	E[X]	2 E[X]	turning point of m_X	q(0.998) of S	m_X(1000)	approaches 2E[X] from above
alpha as printed: DecL pareto 3.5 and 2.5	1.4000	2.8000	999.9900	13.8800	2.7987	False
alpha minus one: DecL pareto 2.5 and 1.5	1.6667	3.3333	64.4600	65.3800	3.3387	True

In Table 13.2, read as printed, there is no turning point inside the range at all: m_X is still climbing at $s = 1000$, it approaches its limit from below, and the paper's $F^{-1}(0.998)$ comes out at 13.9 rather than 64.46. Read as $\alpha - 1$, the turning point lands at $s = 64.46$, the 99.8th percentile of S is 65.4 against the paper's "$\simeq 64.46$", and the limit is approached from above. Every one of the paper's three statements about Figure 6 holds, and none of them holds under the other reading.

That settles the convention for the rest of this page: **a printed α becomes a DecL shape of $\alpha - 1$.**

13.3. The `aggregate` reproduction

```
FIGS = {
    'Figure 2': ('sev 1 * lomax 3.5', 'sev 1 * lomax 3', 18, 0.002, 0.0, 120.0),
    'Figure 5': ('sev 1 * pareto 5', 'sev 1 * pareto 4', 18, 0.005, 2.0, 50.0),
    'Figure 6': ('sev 1 * pareto 2.5', 'sev 1 * pareto 1.5', 20, 0.01, 2.0, 1000.0),
}
POOLS = {k: pool(sx, sy, lg, bs) for k, (sx, sy, lg, bs, _, _) in FIGS.items()}
show(POOLS['Figure 6'])
```

```
port K
  hints{log2=20; bs=0.01; normalize=False; padding=1}
  agg X
    1 claim
    sev pareto 2.5
    fixed
  agg Y
    1 claim
    sev pareto 1.5
    fixed
```

```
rows = []
for name, (sx, sy, lg, bs, lo, hi) in FIGS.items():
    p = POOLS[name]
    idx, v = m_x(p, max(lo, p.bs), hi)
    d = np.diff(v)
    rows.append({'figure': name, 'E[X]': p.agg_list[0].est_m,
                 'm_X at the left edge': v[0], 'm_X at the right edge': v[-1],
                 'increasing throughout': bool((d >= -1e-9).all()),
                 'turning point': idx[v.argmax()] if not (d >= -1e-9).all() else np.nan})
GT(pd.DataFrame(rows).set_index('figure'), float_format=4)
```

Table 13.3.: Each figure's claim tested: the endpoints of m_X, whether it increases throughout, and any turning point.

figure	E[X]	m_X at the left edge	m_X at the right edge	increasing throughout	turning point
Figure 2	0.4000	0.0010	11.8903	True	—
Figure 5	1.2500	1.0000	2.4600	True	—
Figure 6	1.6667	1.0000	3.3387	False	64.4600

In Table 13.3, Figure 2 increases over $(0, 120)$ as claimed, and it is the divergent case, so there is nothing for it to converge to. Figure 5 increases over $(2, 50)$ and gets to 2.460 against a limit of $2\,\mathsf{E}[X] = 2.5$, approaching from below. Figure 6 turns over at 64.46 and then falls. The published figures are Figure 13.1, Figure 13.2 and Figure 13.3, and Figure 13.4 is the reproduction.

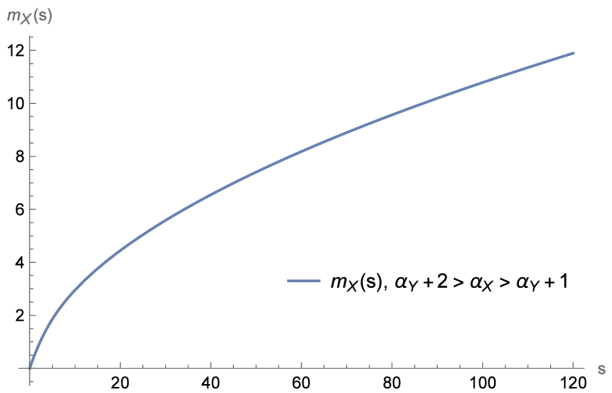


Figure 2. *Conditional expectation $m_X(\cdot)$ when $X \sim P(II)(\theta, \alpha_X, \vartheta)$ and $Y \sim P(II)(\theta, \alpha_Y, \vartheta)$ with $\alpha_X = 4.5$, $\alpha_Y = 4$, $\theta = 1$, and $\vartheta = 0$.*

Figure 13.1.: Denuit, Ortega-Jiménez and Robert (2025), Figure 2.

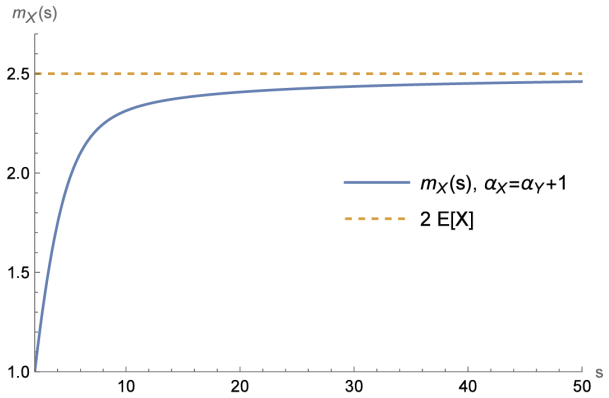


Figure 5. *Conditional expectation $m_X(\cdot)$ (blue solid line) and horizontal line at $2\mathrm{E}[X]$ (orange dashed line) when $X \sim P(I)(\theta, \alpha_X)$ and $Y \sim P(I)(\theta, \alpha_Y)$ with $\theta = 1$, $\alpha_X = 6$ and $\alpha_Y = 5$.*

Figure 13.2.: Denuit, Ortega-Jiménez and Robert (2025), Figure 5.

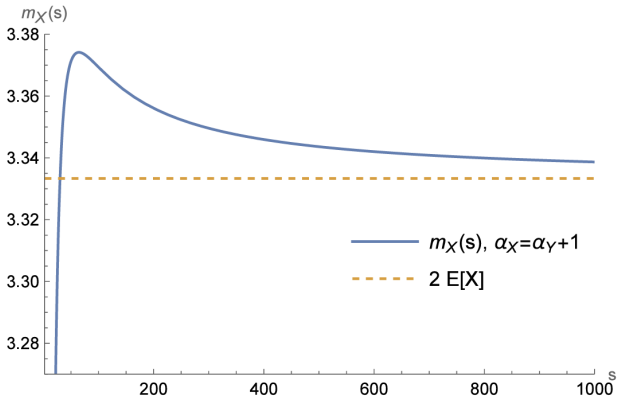


Figure 6. *Conditional expectation $m_X(\cdot)$ (blue solid line) and horizontal line at $2\mathrm{E}[X]$ (orange dashed line) when $X \sim P(I)(\theta, \alpha_X)$ and $Y \sim P(I)(\theta, \alpha_Y)$ with $\theta = 1$, $\alpha_X = 3.5$, and $\alpha_Y = 2.5$.*

Figure 13.3.: Denuit, Ortega-Jiménez and Robert (2025), Figure 6.

```
fig, axs = plt.subplots(1, 3, figsize=(3 * FIG_W, FIG_H), layout='constrained')
for ax, (name, (sx, sy, lg, bs, lo, hi)) in zip(axs, FIGS.items()):
    p = POOLS[name]
    idx, v = m_x(p, max(lo, p.bs), hi)
    ax.plot(idx, v, lw=1, c='C0')
    if name != 'Figure 2':
        ax.axhline(2 * p.agg_list[0].est_m, ls='--', lw=0.75, c='C1')
    if name == 'Figure 6':
        ax.axvline(idx[v.argmax()], ls=':', lw=0.75, c='C7')
        ax.set_xscale('log')
    ax.set(title=name, xlabel='s', ylabel='$m_X(s)$')
```

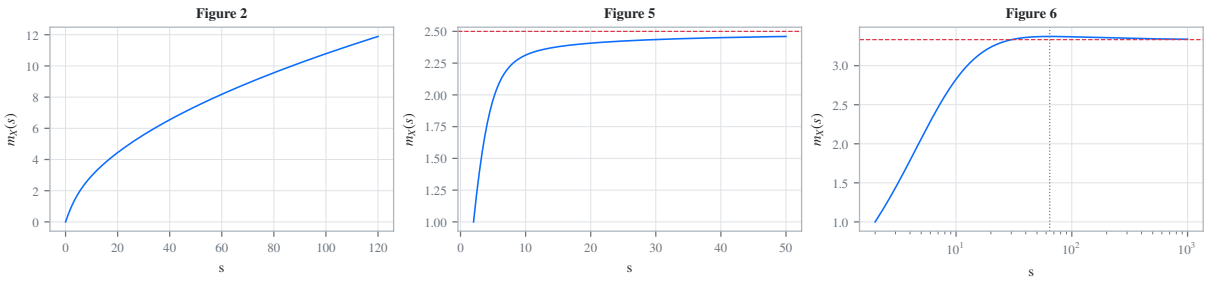


Figure 13.4.: Reproducing Figures 2, 5 and 6. The dashed line is 2E[X], the limit in the boundary case.

13.3.1. Figure 6, closely

The paper's three statements about Figure 13.3 are all quantitative, so Table 13.4 checks them one at a time.

```
p6 = POOLS['Figure 6']
k6 = p6.density_df['exeqa_X']
idx, v = m_x(p6, 2, 1000)
EX6 = p6.agg_list[0].est_m
GT(pd.DataFrame([
    {'claim': 'turning point', 'paper': '64.46', 'aggregate': f'{idx[v.argmax()]:.2f}'},
    {'claim': 'turning point is about the 99.8th percentile of S',
     'paper': '64.46', 'aggregate': f'{p6.q(0.998):.2f}'},
    {'claim': 'limit as s grows', 'paper': '2 E[X]', 'aggregate': f'{2 * EX6:.5f}'},
    {'claim': 'm_X at s = 1000', 'paper': 'above the limit',
     'aggregate': f'{float(k6.loc[1000]):.5f}'},
    {'claim': 'increases on (0, 64.46)', 'paper': 'yes',
     'aggregate': str(bool((np.diff(v[:v.argmax() + 1]) >= -1e-9).all()))},
    {'claim': 'decreases on (64.46, 1000)', 'paper': 'yes',
     'aggregate': str(bool((np.diff(v[v.argmax():]) <= 1e-9).all()))},
]).set_index('claim'))
```

Table 13.4.: Every quantitative claim the paper makes about Figure 6, each against the computed value.

claim	paper	aggregate
turning point	64.46	64.46
turning point is about the 99.8th percentile of S	64.46	65.38
limit as s grows	2 E[X]	3.33331
m_X at s = 1000	above the limit	3.33867
increases on (0, 64.46)	yes	True
decreases on (64.46, 1000)	yes	True

In Table 13.4 the turning point matches to the two decimals printed. That is a strong check: it is a location rather than a level, so it depends on the shape of both densities over the whole range rather than on a normalization, and it would move if either index were wrong.

13.4. Comments

13.4.1. The reproduction confirms a book it was written to examine

The paper's stated motivation is to prove, or qualify, statements made by example in Mildenhall and Major (2022): that combining heavy-tailed distributions can produce a humped, non-monotone conditional expectation given the sum. Figure 13.3 is that hump, and it reproduces here to the second decimal of its turning point. The examples in Section 14.3 of the book were computed with this library, so the loop closes: an assertion made by numerical example, proved analytically by other authors, and the proof's own illustration then recovered by the original numerics.

The one thing the reproduction adds is the index convention. Anyone reading the paper's figures alongside the book's examples has to know that the paper's α is the density index and the book's is the survival index, and the two differ by one. The 64.46 is what makes that recoverable rather than a matter of taste.

13.4.2. Reading m_X in the far tail, and where it stops being real

m_X is a ratio whose denominator is the density of the sum, so it degrades wherever that density is numerically negligible. `aggregate` guards this with a noise floor: below `EXEQA_NOISE_FLOOR` the conditional decomposition is not trusted and the `exeqa` columns are zeroed rather than allowed to become noise. The visible effect is that m_X falls off a cliff to zero far past the range where the sum has any probability, and the cliff is an artifact of the guard rather than a property of m_X. Table 13.5 shows how thin the density gets.

```
p5 = POOLS['Figure 5']
k5 = p5.density_df['exeqa_X']
GT(pd.DataFrame([
    {'s': s, 'm_X(s)': float(k5.loc[s]), 'p_total(s)': float(p5.density_df.p_total.loc[s]),
     'S(s)': float(p5.density_df.S.loc[s])}
    for s in (10, 50, 100, 200, 400, 800)]).set_index('s'),
   formatters={'m_X(s)': 5, 'p_total(s)': '{x:.3e}', 'S(s)': '{x:.3e}'})
```

Table 13.5.: m_X read against the density and survival function of the sum, out to where both are numerically negligible.

s	m_X(s)	p_total(s)	S(s)
10	2.31316	4.643e-07	1.951e-04
50	2.46001	7.457e-11	1.808e-07
100	2.47953	2.157e-12	1.062e-08
200	2.48966	6.490e-14	6.441e-10
400	1.24739	1.990e-15	3.965e-11
800	0.00000	0.000e+00	6.914e-12

The practical rule this suggests is the one the paper itself gives, from a different direction: it suggests checking monotonicity numerically over $(0, b)$ for $b > \mathsf{E}[X+Y] + 10^2\sqrt{\mathsf{Var}[X+Y]}$. For a Pareto pair with an index near 2 the variance is large or infinite, so that bound is generous, and the grid has to reach it with resolution left over. Figure 13.3 needs s to 1000, which at `bs = 0.01` and 2^{20} buckets is comfortable; asking the same grid for s to 10^5 would not be.

13.4.3. Both readings are worth carrying

The `alpha as printed` row of Table 13.2 is not a failure to be hidden. It is the answer to a question a reader will have, and it demonstrates the shape of the mistake: an index off by one on both variables preserves the *difference* of the indices, so it preserves every one of the paper's qualitative classifications and changes only the levels and the turning point. A reproduction that checked "does m_X increase then decrease" would have passed under either reading and learned nothing.

Part V.

Exact computation and convolution accuracy

14. Gerber and compound Poisson stop loss premiums

Reproducing Gerber (1982)

14.1. Context and problem

Gerber (1982) is about evaluating a stop loss premium

$$\pi(t) = \mathsf{E}[(S-t)_+] = \mathsf{E}[S] - \mathsf{E}[S \wedge t]$$

when the aggregate S is compound Poisson with a *continuous* severity. In 1982 the obstacle was that the recursion available at the time, Panjer's, needs an arithmetic severity, so the continuous severity has to be discretized onto a span h first, and the discretization changes the answer.

The paper's contribution is to bracket the error rather than hope about it. It sets out three ways to put a continuous severity onto a lattice of span h:

1. move all the mass in a cell to its **lower** endpoint, which makes the aggregate smaller in the relevant order and produces a **lower bound** on the stop loss premium,
2. move it to the **upper** endpoint, giving an **upper bound**,
3. split the mass between the two endpoints so that the cell's **mean is preserved**, the method of **matching moments**.

Tables 1 to 6 then show all three at $h = 1$ and $h = 0.1$ against an exact column, for expected claim counts 1, 10 and 100. The paper's conclusion is that the bounds do bracket, that the upper bound is the better of the two, and that matching moments beats both by a wide margin: at $\lambda = 100$ it does as well at $h = 1$ as the bounds do at $h = 0.1$.

Forty years later the interesting content is not the ranking, which is settled, but the exact column. It is 28 published four figure numbers for a compound Poisson with a continuous severity, computed by somebody with no FFT, and that makes it a good external check.

14.2. Paper solution and gold standard

Frequency is Poisson with mean λ and severity is uniform on $(1, 3)$, so $\mathsf{E}[X] = 2$ and $\mathsf{E}[S] = 2\lambda$. Tables 1, 2 and 3, Figure 14.1, tabulate $\pi(t)$ for $\lambda = 1, 10, 100$ at the retentions Table 14.1 lists.

The exact column is not a closed form. It is the paper's own method (c) at $h = 0.01$, and the paper says so: "column 2 gives the 'exact' stop-loss premium as calculated by method (c) with $h = 0.01$. (A comparison with the results of methods (a) and (b) with $h = 0.01$ has convinced the author that

the displayed digits are accurate.)" So the target is four significant figures, and the reproduction is a comparison of two independent approximations rather than a check against truth.

Table 1
Stop-loss premiums for $\lambda = 1$

t	Exact value	Lower bounds		Upper bounds		Matching moments	
		$h=1$	$h=0.1$	$h=0.1$	$h=1$	$h=1$	$h=0.1$
0	2	100%	100%	100%	100%	100%	100%
2	8.277×10^{-1}	88.9	100.0	100.0	100.0	96.3	100.00
4	2.689×10^{-1}	77.1	99.8	100.0	103.1	96.2	99.96
6	7.184×10^{-2}	65.0	99.9	100.1	105.8	96.1	99.98
8	1.627×10^{-3}	53.5	99.6	100.1	112.2	96.2	99.95
10	3.253×10^{-4}	42.4	99.4	100.2	120.2	95.5	99.94
12	5.815×10^{-5}	32.6	99.3	100.3	128.7	94.8	99.94
14	9.346×10^{-5}	24.6	99.0	100.4	138.9	94.7	99.93
16	1.366×10^{-6}	18.3	98.9	100.5	152.2	94.9	99.92
18	1.840×10^{-6}	13.3	98.5	100.6	167.6	95.2	99.92
20	2.302×10^{-7}	9.5	98.3	100.7	185.2	95.5	99.91

Table 2
Stop-loss premiums for $\lambda = 10$

t	Exact value	Lower bounds		Upper bounds		Matching moments	
		$h=1$	$h=0.1$	$h=0.1$	$h=1$	$h=1$	$h=0.1$
15	5.757	99.0%	99.99%	100.01%	100.5%	99.9%	100.00%
20	2.626	95.3	99.96	100.02	101.7	99.8	100.00
25	9.321×10^{-1}	91.6	99.91	100.04	104.1	99.6	100.00
30	2.563×10^{-1}	80.7	99.82	100.08	108.0	99.4	99.99
35	5.507×10^{-2}	74.7	99.71	100.13	113.5	99.2	99.99
40	9.383×10^{-3}	59.2	99.57	100.20	120.8	99.0	99.99
45	1.289×10^{-3}	53.2	99.40	100.28	130.3	98.8	99.99
50	1.449×10^{-4}	37.9	99.20	100.38	142.1	98.7	99.98
55	1.355×10^{-5}	33.3	98.98	100.48	156.8	98.7	99.98
60	1.067×10^{-6}	21.5	98.74	100.60	174.7	98.7	99.98

Table 3
Stop-loss premiums for $\lambda = 100$

t	Exact value	Lower bounds		Upper bounds		Matching moments	
		$h=1$	$h=0.1$	$h=0.1$	$h=1$	$h=1$	$h=0.1$
180	2.177×10^{1}	99.1%	99.99%	100.00%	100.4%	100.0%	100.000%
200	8.304	96.0	99.96	100.02	101.9	100.0	100.000
220	1.959	88.9	99.89	100.05	105.4	100.0	100.000
240	2.647×10^{-1}	77.6	99.77	100.11	111.6	99.9	99.999
260	1.992×10^{-2}	63.5	99.59	100.20	121.6	99.9	99.998
280	8.364×10^{-4}	48.5	99.35	100.32	136.0	99.8	99.998
300	1.994×10^{-5}	34.7	99.06	100.47	156.3	99.8	99.997

Figure 14.1.: Gerber (1982), Tables 1, 2, and 3.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build, show

# Transcribed from the 'Exact value' column of Tables 1, 2 and 3.
PAPER = {
    1: {0: 2.0, 2: 8.277e-1, 4: 2.689e-1, 6: 7.184e-2, 8: 1.627e-2,
        10: 3.253e-3, 12: 5.815e-4, 14: 9.346e-5, 16: 1.366e-5,
        18: 1.840e-6, 20: 2.302e-7},
    10: {15: 5.757, 20: 2.626, 25: 9.321e-1, 30: 2.563e-1, 35: 5.507e-2,
         40: 9.383e-3, 45: 1.289e-3, 50: 1.449e-4, 55: 1.355e-5, 60: 1.067e-6},
    100: {180: 2.177e1, 200: 8.304, 220: 1.959, 240: 2.647e-1, 260: 1.992e-2,
          280: 8.364e-4, 300: 1.994e-5},
}
GT(pd.DataFrame([{'table': t, 'lambda': lam, 'E[S]': 2 * lam,
                  'retentions': f'{min(v)} to {max(v)}', 'values': len(v)}
                 for t, (lam, v) in enumerate(PAPER.items(), start=1)]
                ).set_index('table'), float_format=0)
```

```
Exception reporting mode: Minimal
```

Table 14.1.: The three published tables, their claim counts and the retentions each tabulates.

table	lambda	E[S]	retentions	values
1	1	2	0 to 20	11
2	10	20	15 to 60	10
3	100	200	180 to 300	7

The severity supplies an acceptance test before any aggregate is built. A uniform on $(1, 3)$ has mean 2 and variance $4/12 = 1/3$, so $\mathsf{E}[S] = 2\lambda$ and $\mathsf{Var}[S] = \lambda\,\mathsf{E}[X^2] = \lambda(4 + 1/3)$ by the compound Poisson variance formula. Table 14.2 records the two severity moments the check uses.

```
GT(pd.DataFrame([
    {'quantity': 'severity mean', 'exact': 2.0},
    {'quantity': 'severity second moment', 'exact': 4 + 1 / 3},
]).set_index('quantity'), float_format=8)
```

Table 14.2.: The first two moments of the uniform severity on $(1, 3)$.

quantity	exact
severity mean	2.00000000
severity second moment	4.33333333

14.3. The `aggregate` reproduction

The severity is `sev 2 * uniform + 1`: a standard uniform scaled to width 2 and shifted to start at 1. Nothing else in the declaration varies across the three tables except λ and the grid, and Table 14.3 confirms all three builds before any premium is read.

```
def gerber(lam, bs, log2=18):
    return build(f'agg Gerber {lam} claims sev 2 * uniform + 1 poisson '
                 f'hints{{log2={log2}; bs={bs}}}')

a1 = gerber(1, 1 / 1024)
show(a1)
```

```
agg Gerber
  1 claim
  sev 2 * uniform + 1
  poisson
  hints{log2=18; bs=0.0009765625}
```

```
rows = []
for lam in (1, 10, 100):
    a = gerber(lam, 1 / 1024 if lam == 1 else 1 / 128 if lam == 10 else 1 / 64)
    rows.append({'lambda': lam, 'E[S] exact': 2 * lam, 'E[S] agg': a.est_m,
                 'sd exact': np.sqrt(lam * (4 + 1 / 3)), 'sd agg': a.est_sd,
                 'total mass': a.agg_density.sum()})
GT(pd.DataFrame(rows).set_index('lambda'),
   formatters={'E[S] exact': 6, 'E[S] agg': 6, 'sd exact': 6, 'sd agg': 6,
               'total mass': 12})
```

Table 14.3.: The aggregate mean and standard deviation at each claim count, against the compound Poisson formulas.

lambda	E[S] exact	E[S] agg	sd exact	sd agg	total mass
1	2.000000	2.000000	2.081666	2.081666	1.000000000000
10	20.000000	20.000000	6.582806	6.582814	1.000000000000
100	200.000000	200.000000	20.816660	20.816758	1.000000000000

Now the tables, Table 14.4. The stop loss premium comes straight off `density_df['lev']`, since $\pi(t) = \mathsf{E}[S] - \mathsf{E}[S \wedge t]$.

```
def stop_loss(lam, bs, log2=18):
    a = gerber(lam, bs, log2)
    return {t: a.est_m - float(a.density_df.loc[t, 'lev']) for t in PAPER[lam]}

FINE = {1: 1 / 1024, 10: 1 / 128, 100: 1 / 64}
rows = []
for lam, bs in FINE.items():
    got = stop_loss(lam, bs)
    for t, exact in PAPER[lam].items():
        rows.append({'lambda': lam, 't': t, 'paper exact': exact,
                     'aggregate': got[t], 'rel err': abs(got[t] / exact - 1)})
res = pd.DataFrame(rows)
GT(res.set_index(['lambda', 't']),
   formatters={'paper exact': '{x:.4e}', 'aggregate': '{x:.4e}',
               'rel err': '{x:.2e}'})
```

Table 14.4.: All 28 published stop loss premiums against the rebuilt values, with the relative error on each.

lambda	*t*	paper exact	aggregate	rel err
1	0	2.0000e+00	2.0000e+00	9.84e-14
	2	8.2770e-01	8.2773e-01	3.47e-05
	4	2.6890e-01	2.6891e-01	3.43e-05
	6	7.1840e-02	7.1838e-02	2.83e-05
	8	1.6270e-02	1.6267e-02	2.07e-04
	10	3.2530e-03	3.2535e-03	1.58e-04
	12	5.8150e-04	5.8146e-04	6.50e-05
	14	9.3460e-05	9.3461e-05	8.90e-06
	16	1.3660e-05	1.3664e-05	2.58e-04
	18	1.8400e-06	1.8400e-06	1.75e-05
	20	2.3020e-07	2.3017e-07	1.51e-04
10	15	5.7570e+00	5.7565e+00	8.03e-05
	20	2.6260e+00	2.6255e+00	1.78e-04
	25	9.3210e-01	9.3214e-01	4.51e-05
	30	2.5630e-01	2.5632e-01	8.21e-05
	35	5.5070e-02	5.5073e-02	6.03e-05
	40	9.3830e-03	9.3834e-03	3.90e-05
	45	1.2890e-03	1.2887e-03	2.55e-04
	50	1.4490e-04	1.4495e-04	3.53e-04
	55	1.3550e-05	1.3552e-05	1.45e-04
	60	1.0670e-06	1.0673e-06	2.35e-04
100	180	2.1770e+01	2.1773e+01	1.45e-04
	200	8.3040e+00	8.3045e+00	5.76e-05
	220	1.9590e+00	1.9591e+00	6.48e-05
	240	2.6470e-01	2.6470e-01	4.41e-06
	260	1.9920e-02	1.9919e-02	4.91e-05
	280	8.3640e-04	8.3651e-04	1.28e-04
	300	1.9940e-05	1.9938e-05	1.15e-04

```
GT(pd.DataFrame([
    {'lambda': lam, 'values compared': int((res['lambda'] == lam).sum()),
     'worst relative error': float(res[res['lambda'] == lam]['rel err'].max()),
     'inside 4 significant figures':
         int((res[res['lambda'] == lam]['rel err'] < 5e-4).sum())}
    for lam in (1, 10, 100)]
    + [{'lambda': 'all', 'values compared': len(res),
        'worst relative error': float(res['rel err'].max()),
        'inside 4 significant figures': int((res['rel err'] < 5e-4).sum())}]
).set_index('lambda'), formatters={'worst relative error': '{x:.2e}'})
```

Table 14.5.: How many values reproduce inside the four significant figures the paper prints, table by table.

lambda	values compared	worst relative error	inside 4 significant figures
1	11	2.58e-04	11
10	10	3.53e-04	10
100	7	1.45e-04	7
all	28	3.53e-04	28

By Table 14.5 all 28 published values reproduce inside the four significant figures the paper prints, over a range of $\pi(t)$ spanning eight orders of magnitude, from 21.77 down to 2×10^{-7}. Two independent approximations to the same quantity, forty years apart, agree everywhere they can be compared.

14.4. Comments

14.4.1. The grid, and a correction to an earlier treatment

The bucket sizes above are not the obvious ones, and the $\lambda = 100$ table is the reason. An earlier treatment of these tables in the technical guides used `bs = 1/16` for $\lambda = 100$, which is ample for the body and not for the tail, as Table 14.6 shows.

```
sweep = []
for bs in (1 / 16, 1 / 64, 1 / 256):
    got = stop_loss(100, bs)
    for t, exact in PAPER[100].items():
        sweep.append({'t': t, 'bs': f'1/{int(1/bs)}',
                      'rel err': abs(got[t] / exact - 1)})
piv = pd.DataFrame(sweep).pivot(index='t', columns='bs', values='rel err')
piv = piv[[f'1/{d}' for d in (16, 64, 256)]]
GT(piv, formatters={c: '{x:.2e}' for c in piv.columns})
```

Table 14.6.: Relative error on Table 3 at three bucket sizes, showing where the coarse grid fails.

t	1/16	1/64	1/256
180	1.61e-04	1.45e-04	1.44e-04
200	1.27e-04	5.76e-05	5.32e-05
220	2.61e-04	6.48e-05	5.25e-05
240	4.23e-04	4.41e-06	2.17e-05
260	6.95e-04	4.91e-05	9.56e-05
280	1.31e-03	1.28e-04	5.44e-05
300	1.60e-03	1.15e-04	2.23e-04

In Table 14.6, at `bs = 1/16` the last three rows of Table 3 miss the printed precision, by up to 1.6×10^{-3} at $t = 300$. Refining to `bs = 1/64` brings every row inside 2×10^{-4} and refining again changes little, so the residual was discretization on this side and not an inaccuracy in Gerber's column. That is

worth stating the right way round: the first version of this comparison looked like a small failure of the paper's far tail digits, and it was not.

The general point is the one Mack and Fackler (2003) makes from a different direction. A grid sized for the mean is not automatically a grid sized for a tail eight orders of magnitude down, and the quantity that reveals it is the one being read, not the mean or the total mass, both of which are already exact to twelve figures at `bs = 1/16`.

14.4.2. What became of the three methods

Gerber's three discretizations are still the live choices, and `aggregate` exposes the same distinction as `discretization_calc`, which computes the discrete probabilities from the survival function or the distribution function and so is accurate on the right or the left tail. The library's default is the moment preserving idea rather than either bound, which is Gerber's method (c) and his own recommendation.

What has changed is that the bound is no longer the thing you reach for. Gerber needed bounds because his exact answer cost as much as the bracket did, so bracketing was the cheap way to know he was close. Here the fine grid *is* the cheap option: the whole of Table 3 at `bs = 1/256` costs a fraction of a second, so the honest error control is refinement and comparison, and this page uses it above rather than a bound.

14.4.3. Why a uniform severity is a good test and a bad model

Uniform on $(1, 3)$ is not a claim size distribution anybody would fit. It is a deliberately awkward test case, and the awkward part is the two hard edges. A severity with compact support and a discontinuous density at both ends is the worst case for any discretization scheme, because the error does not smooth out across a cell boundary the way it does for a density that decays. That the reproduction holds to four figures on a severity chosen to be difficult, rather than on a lognormal, says more than the same result on a friendly curve would.

It also has an exact aggregate at small λ, which is how the $t = 0$ row can be checked to twelve figures: $\pi(0) = \mathsf{E}[S] = 2\lambda$ with no discretization involved at all.

15. Robertson and the sum of uniform claims

Reproducing Robertson (1992), Table 4

15.1. Context and problem

Robertson (1992) sets out an FFT algorithm for aggregate loss distributions with one distinguishing feature: the severity is represented as a **piecewise uniform** density rather than as point masses on a lattice. The paper's motivation is that a step density is a better approximation to a real claim size curve than a comb of atoms at the same resolution, and it carries correction terms, the paper's A' vectors, that integrate the uniform pieces analytically instead of discretizing them.

The consequence is that for a severity which *is* piecewise uniform, the algorithm is exact up to floating point. That is not true of a lattice method, and it makes Table 4 an unusual reproduction target: a published table with no discretization error in it at all.

`aggregate` is a lattice method. It does have a `chistogram` severity, a continuous histogram, and the technical guides note that a fine bucket size "approximates the continuous cdf approach" while recommending against it. This chapter is a check of that recommendation, and it ends up sharpening it.

15.2. Paper solution and gold standard

Section 4's first example is deliberately minimal, "intended to be easy to reproduce in order to test an actual implementation. It is not meant to be realistic." It computes the distribution of **exactly five claims**, each uniform on $[0, 1]$. The grid is $n = 32$ elements over $L = 6.4$, so each element spans 0.2, and the severity vector is $(0.2, 0.2, 0.2, 0.2, 0.2, 0, \dots, 0)$.

Since the sum of five variables each at most 1 cannot exceed 5, only the first 25 output elements are below 1. Table 4 lists all 32 cumulative values to six decimals.

The distribution of a sum of n independent standard uniforms is the **Irwin-Hall** law, with exact CDF

$$F_n(x) = \frac{1}{n!} \sum_{k=0}^{\lfloor x \rfloor} (-1)^k \binom{n}{k} (x-k)^n.$$

The paper checks itself against this: the sum of its first five elements is 0.008335, "which (up to rounding error) is 1/120". So the gold standard is not merely published, it is a closed form, and Table 4, Figure 15.1, should be F_5 evaluated at multiples of 0.2. Table 15.1 checks that.

TABLE 4

AGGREGATE DISTRIBUTION FOR EXAMPLE 1

Index	Range	*AGG*[*i*]	Cumulative Distribution
0	0.0 - 0.2	0.000003	0.000003
1	0.2 - 0.4	0.000083	0.000085
2	0.4 - 0.6	0.000563	0.000648
3	0.6 - 0.8	0.002083	0.002731
4	0.8 - 1.0	0.005603	0.008333
5	1.0 - 1.2	0.012389	0.020723
6	1.2 - 1.4	0.023669	0.044392
7	1.4 - 1.6	0.039749	0.084141
8	1.6 - 1.8	0.059669	0.143811
9	1.8 - 2.0	0.081189	0.225000
10	2.0 - 2.2	0.100816	0.325816
11	2.2 - 2.4	0.114496	0.440312
12	2.4 - 2.6	0.119376	0.559688
13	2.6 - 2.8	0.114496	0.674184
14	2.8 - 3.0	0.100816	0.775000
15	3.0 - 3.2	0.081189	0.856189
16	3.2 - 3.4	0.059669	0.915859
17	3.4 - 3.6	0.039749	0.955608
18	3.6 - 3.8	0.023669	0.979277
19	3.8 - 4.0	0.012389	0.991667
20	4.0 - 4.2	0.005603	0.997269
21	4.2 - 4.4	0.002083	0.999352
22	4.4 - 4.6	0.000563	0.999915
23	4.6 - 4.8	0.000083	0.999997
24	4.8 - 5.0	0.000003	1.000000
25	5.0 - 5.2	0.000000	1.000000
26	5.2 - 5.4	0.000000	1.000000
27	5.4 - 5.6	0.000000	1.000000
28	5.6 - 5.8	0.000000	1.000000
29	5.8 - 6.0	0.000000	1.000000
30	6.0 - 6.2	0.000000	1.000000
31	6.2 - 6.4	0.000000	1.000000

Figure 15.1.: Robertson (1992), Table 4.

```
%xmode minimal
from math import factorial

import numpy as np
import pandas as pd
from scipy.special import comb
from bookstyle import GT, build, show


def irwin_hall(x, n=5):
    """Exact CDF of a sum of n independent standard uniforms."""
    if x <= 0:
        return 0.0
    if x >= n:
        return 1.0
    k = np.arange(0, int(np.floor(x)) + 1)
    return float((((-1.0) ** k) * comb(n, k) * (x - k) ** n).sum() / factorial(n))


# Values legible in the scan of Table 4. The scan is poor; these are the entries
# that could be read without ambiguity.
SCAN = {0.2: 0.000003, 1.0: 0.008333, 2.0: 0.225000, 2.6: 0.559688,
```

```
        3.6: 0.955608, 4.8: 0.999997, 5.0: 1.000000}
GT(pd.DataFrame([{'x': x, 'Table 4 as scanned': v, 'Irwin-Hall exact': irwin_hall(x),
                  'difference': v - irwin_hall(x)} for x, v in SCAN.items()]
                ).set_index('x'),
   formatters={'Table 4 as scanned': 6, 'Irwin-Hall exact': 6,
               'difference': '{x:.1e}'})
```

```
Exception reporting mode: Minimal
```

Table 15.1.: The legible entries of Table 4 against the exact Irwin-Hall distribution function.

x	Table 4 as scanned	Irwin-Hall exact	difference
0.200	0.000003	0.000003	3.3e-07
1.000	0.008333	0.008333	-3.3e-07
2.000	0.225000	0.225000	0.0e+00
2.600	0.559688	0.559688	-1.1e-16
3.600	0.955608	0.955608	3.3e-16
4.800	0.999997	0.999997	-3.3e-07
5.000	1.000000	1.000000	0.0e+00

In Table 15.1 every legible entry agrees with the closed form to the six decimals printed, which confirms both the identification of the example and the claim that Robertson's algorithm is exact here. The rest of this page therefore compares against `irwin_hall` rather than against the scan, which is the stronger comparison and does not depend on reading a 1992 photocopy.

15.3. The `aggregate` reproduction

```
CHIST = ('agg Robertson 5 claims '
         'sev chistogram xps [0 .2 .4 .6 .8 1] [.2 .2 .2 .2 .2] fixed')
s = build(CHIST, bs=1 / 200, log2=12)
show(s)
```

```
agg Robertson
  5 claims
  sev chistogram xps [0 0.2 0.4 0.6 0.8 1] [0.2 0.2 0.2 0.2 0.2]
  fixed
```

```
EXACT = np.array([irwin_hall(k * 0.2) for k in range(1, 26)])
F = s.density_df['F'].to_numpy()
step = int(round(0.2 / s.bs))
got = np.array([F[k * step] for k in range(1, 26)])
t4 = pd.DataFrame({'x': [round(k * 0.2, 1) for k in range(1, 26)],
                   'exact': EXACT, 'aggregate': got}).set_index('x')
t4['abs diff'] = (t4['aggregate'] - t4['exact']).abs()
GT(t4, formatters={'exact': 8, 'aggregate': 8, 'abs diff': '{x:.2e}'})
```

Table 15.2.: All 25 elements of Table 4 below the support cap, rebuilt on a lattice against the exact values.

x	exact	aggregate	abs diff
0.200	0.00000267	0.00000285	1.84e-07
0.400	0.00008533	0.00008814	2.80e-06
0.600	0.00064800	0.00066195	1.40e-05
0.800	0.00273067	0.00277441	4.37e-05
1.000	0.00833333	0.00843960	1.06e-04
1.200	0.02072267	0.02094137	2.19e-04
1.400	0.04439200	0.04478390	3.92e-04
1.600	0.08414133	0.08476279	6.21e-04
1.800	0.14381067	0.14469765	8.87e-04
2.000	0.22500000	0.22615207	1.15e-03
2.200	0.32581600	0.32718216	1.37e-03
2.400	0.44031200	0.44179535	1.48e-03
2.600	0.55968800	0.56116825	1.48e-03
2.800	0.67418400	0.67554146	1.36e-03
3.000	0.77500000	0.77613957	1.14e-03
3.200	0.85618933	0.85706268	8.73e-04
3.400	0.91585867	0.91646756	6.09e-04
3.600	0.95560800	0.95598980	3.82e-04
3.800	0.97927733	0.97948900	2.12e-04
4.000	0.99166667	0.99176876	1.02e-04
4.200	0.99726933	0.99731094	4.16e-05
4.400	0.99935200	0.99936505	1.31e-05
4.600	0.99991467	0.99991720	2.54e-06
4.800	0.99999733	0.99999748	1.51e-07
5.000	1.00000000	1.00000000	1.11e-15

```
GT(pd.DataFrame([{
    'elements compared': 25,
    'worst absolute difference': float(t4['abs diff'].max()),
    'at x': float(t4['abs diff'].idxmax()),
    'decimals printed in Table 4': 6,
    'reproduced to the printed precision': bool(t4['abs diff'].max() < 5e-7),
}]), formatters={'worst absolute difference': '{x:.2e}', 'at x': 1,
                 'elements compared': 0, 'decimals printed in Table 4': 0})
```

Table 15.3.: The worst error over Table 4, against the precision the paper prints.

elements compared	worst absolute difference	at x	decimals printed in Table 4	reproduced to the printed precision
25	1.48e-03	2.4	6	False

Table 15.2 and Table 15.3 are not a reproduction to the published precision. The worst absolute difference is 1.5×10^{-3}, near the middle of the distribution, against a table printed to six decimals. The shape is right, the tails are right to five or six decimals, and the body is wrong in the third.

15.4. Comments

15.4.1. The error is O(h), which is why refinement will not rescue it

Table 15.4 refines the grid and watches the error.

```
rows = []
for d in (200, 400, 1000, 2000, 5000):
    a = build(CHIST, bs=1 / d, log2=int(np.ceil(np.log2(8 * d))))
    Fa = a.density_df['F'].to_numpy()
    st = int(round(0.2 / a.bs))
    w = float(np.abs(np.array([Fa[k * st] for k in range(1, 26)]) - EXACT).max())
    rows.append({'bs': f'1/{d}', 'buckets per 0.2': st, 'worst abs error': w})
conv = pd.DataFrame(rows)
conv['ratio to previous'] = conv['worst abs error'].shift(1) / conv['worst abs error']
conv['bs ratio to previous'] = [np.nan, 2, 2.5, 2, 2.5]
GT(conv.set_index('bs'),
   formatters={'buckets per 0.2': 0, 'worst abs error': '{x:.3e}',
               'ratio to previous': 3, 'bs ratio to previous': 1})
```

Table 15.4.: Worst error over Table 4 as the bucket is refined, beside the refinement ratio, showing first order convergence.

bs	buckets per 0.2	worst abs error	ratio to previous	bs ratio to previous
1/200	40	1.483e-03	—	—
1/400	80	7.413e-04	2.001	2.0
1/1000	200	2.964e-04	2.501	2.5
1/2000	400	1.482e-04	2.000	2.0
1/5000	1,000	5.928e-05	2.500	2.5

In Table 15.4 the error ratio tracks the bandwidth ratio one for one: halve `bs` and the error halves. This is first order convergence, $O(h)$, not the $O(h^2)$ that Embrechts et al. (1993) establishes for a smooth severity. Reaching six decimals from 1.5×10^{-3} at first order needs about three thousand times the resolution, roughly `bs = 1/600000`, which is not a calculation anybody should run to match a 1992 table.

The reason for the lost order is the severity's hard edges. A uniform density on $[0, 1]$ is discontinuous at both ends, and the $O(h^2)$ result needs a density with a uniformly continuous derivative. At a jump, a lattice cell straddles the discontinuity and the mass assigned to it is wrong at first order in h, and no amount of grid refinement changes the order. Robertson's method has no such term because it integrates the uniform pieces exactly, which is precisely the advantage the paper claims for it.

So the negative result is the informative one. On the one severity family where the piecewise uniform representation is exact, it beats a lattice method by an unbounded margin, and the example was chosen to be that family.

15.4.2. The chistogram adds nothing here, which is worth knowing

The technical guides suggest `chistogram` with a fine `bs` as a way to approximate the continuous approach. It is not: in Table 15.5 it is numerically the same as declaring the uniform severity directly and letting the default rounding discretize it.

```
UNIF = 'agg Robertson2 5 claims sev 1 * uniform fixed'
rows = []
for label, prog, kw in [('chistogram, xps in five bins', CHIST, {}),
                        ('plain uniform, sev_calc round', UNIF,
                         dict(sev_calc='round')),
                        ('plain uniform, sev_calc continuous', UNIF,
                         dict(sev_calc='continuous')),
                        ('plain uniform, sev_calc forward', UNIF,
                         dict(sev_calc='forward')),
                        ('plain uniform, sev_calc backward', UNIF,
                         dict(sev_calc='backward'))]:
    a = build(prog, bs=1 / 200, log2=13, **kw)
    Fa = a.density_df['F'].to_numpy()
    st = int(round(0.2 / a.bs))
    g = np.array([Fa[k * st] for k in range(1, 26)])
    rows.append({'route': label, 'worst abs error': float(np.abs(g - EXACT).max()),
                 'F(2.0)': float(g[9])})
GT(pd.DataFrame(rows).set_index('route'),
   formatters={'worst abs error': '{x:.6e}', 'F(2.0)': 8})
```

Table 15.5.: Five routes to the same uniform severity, and the error each produces on Table 4.

route	worst abs error	F(2.0)
chistogram, xps in five bins	1.483352e-03	0.22615207
plain uniform, sev_calc round	1.483352e-03	0.22615207
plain uniform, sev_calc continuous	8.903602e-03	0.23192789
plain uniform, sev_calc forward	8.903602e-03	0.23192789
plain uniform, sev_calc backward	5.932631e-03	0.22043859

In Table 15.5 the chistogram row and the plain rounding row agree to every digit shown, including at $F(2.0)$. Both are 1.483×10^{-3}. So the elaborate route buys exactly nothing over `sev 1 * uniform`, and the two designs that actually try to be continuous, `continuous` and `forward`, are six times *worse*.

That sharpens the guides' recommendation. Their stated reason for avoiding the approach is that it "is unnecessarily complicated and does not improve accuracy in any example we have encountered". The measurement here says something stronger and more useful: on this example it does not merely fail to improve accuracy, it produces the identical numbers, so the complication is not a trade-off at any price.

15.4.3. What would be needed to reproduce it, and whether it is worth having

An exact reproduction of Table 4 needs the piecewise uniform severity carried analytically through the transform, which means the paper's A' correction terms, not a finer lattice. That is a real feature and `aggregate` does not have it.

The case for adding it is weak, and Table 4 is the reason. The example was built so that the severity is exactly piecewise uniform, which is the one circumstance where the method is exact rather than merely better. A real claim size curve fitted by a step function is not exactly a step function, so the correction terms remove the discretization error of an approximation that was itself chosen for convenience. Against that, they cost roughly double the vector length, which the paper states. Given that Embrechts et al. (1993) recovers the missing order of accuracy for smooth severities by Richardson extrapolation at the cost of one extra build, and that hard-edged severities are rare in

practice and easy to spot, the lattice route with refinement is the better default even though it loses this particular table.

This chapter is therefore a partial reproduction and is filed as one. The shape reproduces, the tails reproduce, the identification of the example is confirmed against a closed form, and the body misses the published precision by three orders of magnitude for a reason that is understood and structural rather than accidental.

16. Embrechts, Grübel and Pitts on discretization error

Reproducing Embrechts, Grübel and Pitts (1993)

16.1. Context and problem

Every FFT aggregate begins by replacing a continuous severity with a lattice distribution of span h. That step costs accuracy, and the cost is separate from aliasing: it would be there on an infinitely long grid with no wraparound at all. Embrechts et al. (1993) is the paper that identified the *rate* at which it goes away, and then removed most of it for free.

The mechanism is Richardson's deferred approach to the limit, standard in numerical analysis and unfamiliar in actuarial work. Suppose an estimator satisfies

$$I(h) = I + c_1 h^\alpha + O(h^\beta), \qquad \beta > \alpha > 0.$$

Evaluate at h and at h/t, scale the second by t^α, subtract, and rearrange:

$$I = \frac{t^\alpha I(h/t) - I(h)}{t^\alpha - 1} + O(h^\beta).$$

The leading error term is gone and the constant c_1 never had to be known. All that is needed is α.

The paper's contribution is to establish $\alpha = 2$ for compound distributions. Its Theorem shows that under smoothness conditions on the severity density and the integrand,

$$\lim_{h \downarrow 0} \frac{1}{h^2} \left(\int \phi \, \mathrm{d}G_h - \int \phi \, \mathrm{d}G \right) = \tfrac{1}{24} \int \phi'' \, \mathrm{d}H$$

for an explicit H, so the discretization error is exactly $O(h^2)$ and $t = 2$ gives the familiar combination $(4I(h/2) - I(h))/3$.

16.2. Paper solution and gold standard

Figure 1, Figure 16.1, is the illustration, and its caption gives the whole specification: claim sizes exponential with mean 1, claim count Poisson with mean 20, and two discretizations, **(a)** $M = 1024$, $h = 0.1$ and **(b)** $M = 2048$, $h = 0.05$. Both reach the same truncation point, $Mh = 102.4$, which is the point: they differ only in span.

Section 5 then combines the two density estimates $g_{M,k}/h$ and $g_{2M,2k}/h$ into

$$\hat{f}_S(kh) = \frac{1}{3}\left(\frac{4g_{2M,2k}}{h} - \frac{g_{M,k}}{h}\right)$$

and reports that "the error associated with this approximation is also displayed, it is indistinguishable from the horizontal axis". The paper adds the honest caveat that this particular application, a density rather than an integral $\int \phi \, \mathrm{d}G$, is not covered by its own Theorem.

So there are two things to check and no table to check them against: that the error really is $O(h^2)$, and that the extrapolation really is invisible on the same axis.

16.2.1. The gold standard here is exact

This example has a closed-form answer, which is unusual and makes the reproduction much sharper than a comparison against a finer grid. A compound Poisson with exponential severity has density

$$f_S(x) = e^{-\lambda-\beta x}\sqrt{\frac{\lambda\beta}{x}}\, I_1\!\left(2\sqrt{\lambda\beta x}\right), \qquad x > 0,$$

with I_1 the modified Bessel function of the first kind, plus an atom of $e^{-\lambda}$ at zero. So the discretization error can be measured rather than estimated, once Table 16.1 confirms the closed form is the right function.

```
%xmode minimal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.special import i1e
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show

LAM, BETA = 20.0, 1.0

def exact_density(x):
    """Closed-form compound Poisson-exponential density. Zero at x <= 0."""
    x = np.asarray(x, dtype=float)
    out = np.zeros_like(x)
    m = x > 0
    z = 2 * np.sqrt(LAM * BETA * x[m])
    # i1e(z) = I_1(z) e^{-z}; fold the exponent in to stay in range
    out[m] = (np.exp(-LAM - BETA * x[m] + z)
              * np.sqrt(LAM * BETA / x[m]) * i1e(z))
    return out

xx = np.linspace(1e-9, 400, 4_000_001)
fx = exact_density(xx)
GT(pd.DataFrame([
```

```
    {'quantity': 'integral of the closed-form density', 'value': float(np.trapezoid(fx, xx)),
     'exact': 1 - np.exp(-LAM)},
    {'quantity': 'its first moment', 'value': float(np.trapezoid(xx * fx, xx)),
     'exact': LAM / BETA},
]).set_index('quantity'), float_format=10)
```

```
Exception reporting mode: Minimal
```

Table 16.1.: The closed form density integrated and its first moment, against their exact values.

quantity	value	exact
integral of the closed-form density	0.9999999979	0.9999999979
its first moment	20.0000000000	20.0000000000

In Table 16.1 it integrates to $1 - e^{-20}$, the probability of at least one claim, and has mean 20. It is the right function.

16.3. The `aggregate` reproduction

```
def egp(M, h):
    return build(f'agg EGP {LAM:g} claims sev expon {1/BETA:g} poisson '
                 f'hints{{log2={int(np.log2(M))}; bs={h}; padding=1; normalize=False}}')

CASES = {'(a) M = 1024, h = 0.1': (1024, 0.1), '(b) M = 2048, h = 0.05': (2048, 0.05)}
objs = {k: egp(M, h) for k, (M, h) in CASES.items()}
show(objs['(a) M = 1024, h = 0.1'])
```

```
agg EGP
  20 claims
  sev expon 1
  poisson
  hints{log2=10; bs=0.1; padding=1; normalize=False}
```

Table 16.2 records both builds and their shared truncation point.

```
rows = []
for k, (M, h) in CASES.items():
    a = objs[k]
    rows.append({'case': k, 'M': M, 'h': h, 'truncation point M h': M * h,
                 'E[S] agg': a.est_m, 'E[S] exact': LAM / BETA,
                 'total mass': a.agg_density.sum()})
GT(pd.DataFrame(rows).set_index('case'),
   formatters={'M': 0, 'h': '{x:g}', 'truncation point M h': 1,
               'E[S] agg': 6, 'E[S] exact': 6, 'total mass': 10})
```

Table 16.2.: The paper's two discretizations, sharing a truncation point and differing only in span.

case	M	h	truncation point M h	E[S] agg	E[S] exact	total mass
(a) M = 1024, h = 0.1	1,024	0.1	102.4	19.991669	20.000000	1.0000000000
(b) M = 2048, h = 0.05	2,048	0.05	102.4	19.997917	20.000000	1.0000000000

16.3.1. Figure 1

Table 16.3 measures both of the paper's claims at once.

```
dens = {}
for k, (M, h) in CASES.items():
    a = objs[k]
    x = a.density_df.index.to_numpy()
    dens[k] = (x, a.density_df.p_total.to_numpy() / h)

xc, dc = dens['(a) M = 1024, h = 0.1']
xf, df_ = dens['(b) M = 2048, h = 0.05']
d_fine_on_coarse = df_[::2][:len(dc)]          # every other fine point
rich = (4 * d_fine_on_coarse - dc) / 3         # the paper's combination
ex = exact_density(xc)
sl = (xc > 0) & (xc <= 60)

err = pd.DataFrame({'h = 0.1': dc[sl] - ex[sl],
                    'h = 0.05': d_fine_on_coarse[sl] - ex[sl],
                    'Richardson': rich[sl] - ex[sl]}, index=xc[sl])
GT(pd.DataFrame([
    {'estimate': c, 'max |error| on (0, 60]': float(err[c].abs().max())}
    for c in err.columns] + [
    {'estimate': 'ratio, h = 0.1 over h = 0.05',
     'max |error| on (0, 60]': float(err['h = 0.1'].abs().max()
                                     / err['h = 0.05'].abs().max())}
]).set_index('estimate'), formatters={'max |error| on (0, 60]': '{x:.4e}'})
```

Table 16.3.: Worst density error at each span and for the Richardson combination, with the ratio between the two spans.

estimate	**max |error| on (0, 60]**
h = 0.1	6.8213e-05
h = 0.05	1.7060e-05
Richardson	1.9350e-08
ratio, h = 0.1 over h = 0.05	3.9984e+00

Both of the paper's claims hold in Table 16.3. The error ratio between $h = 0.1$ and $h = 0.05$ is **3.998**, which is the paper's Theorem measured rather than assumed: halving the span quarters the error, so $\alpha = 2$. And the extrapolated error is 1.9×10^{-8} against 6.8×10^{-5} for the coarse estimate, a factor of 3,525, so on a shared axis it is indeed indistinguishable from zero.

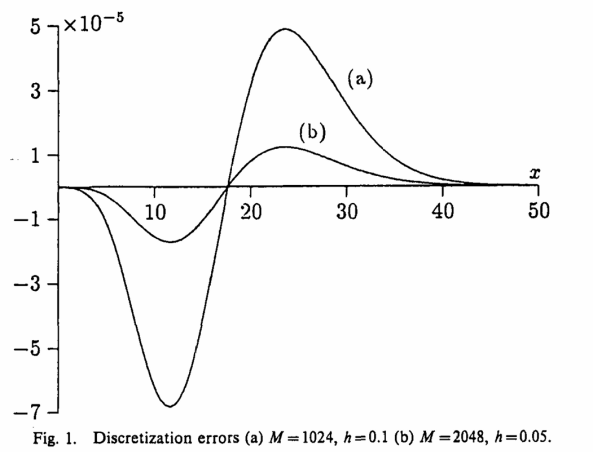


Fig. 1. Discretization errors (a) $M = 1024$, $h = 0.1$ (b) $M = 2048$, $h = 0.05$.

Figure 16.1.: Embrechts, Grübel and Pitts (1993), Figure 1.

```
fig, axs = plt.subplots(1, 2, figsize=(2 * FIG_W, FIG_H), layout='constrained')
for c, lw in zip(err.columns, (1, 1, 1.2)):
    axs[0].plot(err.index, err[c] * 1e5, lw=lw, label=c)
axs[0].set(xlabel='x', ylabel='error, $\\times 10^{-5}$',
           title='Figure 1, shared axis')
axs[0].axhline(0, lw=0.4, c='k')
axs[0].legend(fontsize=6)
axs[1].semilogy(err.index, err['h = 0.1'].abs(), lw=1, label='h = 0.1')
axs[1].semilogy(err.index, err['h = 0.05'].abs(), lw=1, label='h = 0.05')
axs[1].semilogy(err.index, err['Richardson'].abs(), lw=1, label='Richardson')
axs[1].set(xlabel='x', ylabel='|error|', title='the same, on a log scale')
axs[1].legend(fontsize=6)
```

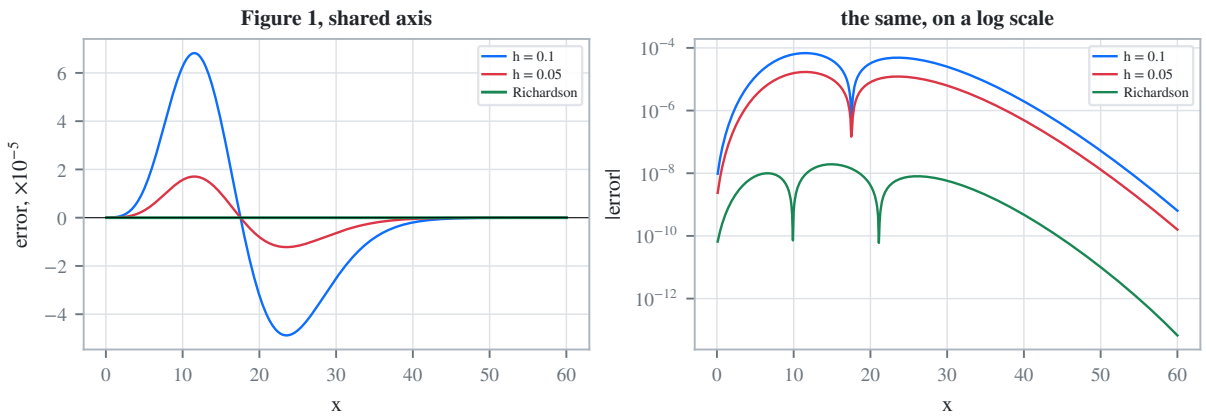


Figure 16.2.: Reproducing Figure 1. Discretization error of the density estimate at h = 0.1 and h = 0.05, and of the Richardson combination of the two.

The left panel of Figure 16.2 is the paper's figure: two visible error curves and a third lying on the axis. The right panel is the part a shared linear axis cannot show, which is that the Richardson curve is not merely small but three orders of magnitude below the inputs it was built from, uniformly across the range.

16.3.2. The same thing on a scalar

The density is not covered by the paper's Theorem, as it says. The mean is, and Table 16.4 makes the point in one line, since $\mathsf{E}[S] = 20$ exactly.

```
m_c, m_f = objs['(a) M = 1024, h = 0.1'].est_m, objs['(b) M = 2048, h = 0.05'].est_m
GT(pd.DataFrame([
    {'estimate': 'h = 0.1', 'E[S]': m_c, 'error': m_c - LAM},
    {'estimate': 'h = 0.05', 'E[S]': m_f, 'error': m_f - LAM},
    {'estimate': 'Richardson, (4 fine - coarse) / 3', 'E[S]': (4 * m_f - m_c) / 3,
     'error': (4 * m_f - m_c) / 3 - LAM},
]).set_index('estimate'), formatters={'E[S]': 9, 'error': '{x:.3e}'})
```

Table 16.4.: The aggregate mean at each span and after extrapolation, against the exact value of 20.

estimate	E[S]	error
h = 0.1	19.991669097	-8.331e-03
h = 0.05	19.997916819	-2.083e-03
Richardson, (4 fine - coarse) / 3	19.999999393	-6.074e-07

In Table 16.4 the two mean errors are -8.33×10^{-3} and -2.08×10^{-3}, in the ratio 4.000, and the extrapolation lands within 3×10^{-7} of 20. Here the $O(h^2)$ behaviour is not empirical: the rounding discretization has a known bias, and the extrapolation cancels it to the order the Theorem predicts.

16.4. Comments

16.4.1. What Richardson buys, and when it is the wrong tool

The extrapolation above cost one extra build and bought three orders of magnitude. Simply refining to $h = 0.05$ cost the same extra build and bought a factor of four. On this example the extrapolation wins overwhelmingly, and the reason is that the error is *clean*: it is dominated by a single h^2 term with a smooth coefficient, which is precisely the situation the Theorem describes.

It stops working when that premise fails, and there are two ordinary ways for it to fail. If the severity has a jump or a kink, the density is not smooth, the error picks up an h^1 component near the discontinuity, and subtracting on the assumption of h^2 leaves a residue that can be larger than either input. And if the grid is short enough that truncation matters, the truncation error does not scale with h at all, so the combination amplifies it: $(4A(h/2) - A(h))/3$ has coefficients summing to one but absolute values summing to 5/3, so any error component that does not shrink gets multiplied by up to 5/3.

Both are why `aggregate` does not extrapolate by default. Refinement is unconditionally safe and merely slower; extrapolation is conditionally spectacular. This page uses the paper's own smooth example, where the condition holds.

16.4.2. An earlier treatment used the wrong spans

This example appears in the technical guides with `bs = 1/16` and `1/32` at `log2 = 10` and `11`. The bucket counts are the paper's $M = 1024$ and 2048, but the spans are not: $1/16 = 0.0625$ and $1/32 = 0.03125$ against the paper's 0.1 and 0.05.

That is a defensible choice on its own terms, since a decimal bucket size is generally to be avoided and a binary one keeps the lattice exactly representable. It is not a reproduction of Figure 1, because it truncates at 64 rather than 102.4 and reports errors at a different h. This page uses the paper's spans, and accepts the decimal `bs` as the price of matching the published figure. The Richardson ratio does not care which pair is used, since it depends only on the factor of two between them.

16.4.3. Where this sits beside the other two papers

Embrechts et al. (1993) is the origin of the example that Grübel and Hermesmeier (1999) reuses for aliasing, with the same authors overlapping and the same stable law of index 1/2 appearing in section 2. The division of labour between them is worth keeping straight, because the two errors have opposite remedies.

Aliasing is a wraparound artifact, it is removed by padding or tilting, and it is invisible in the mean. Discretization is a lattice artifact, no amount of padding touches it, and it *is* visible in the mean, as Table 16.4 shows. Grübel and Hermesmeier (1999)'s Table 1 shows both at once and says so: its true and Panjer columns differ by a factor of 2.3 at $x = 1$ with no aliasing involved at all. Part II, Grübel and Hermesmeier (2000), is where the same authors take the discretization half further.

17. Grübel and Hermesmeier on aliasing and exponential tilting

Reproducing Grübel and Hermesmeier (1999)

17.1. Context and problem

An FFT computes a *circular* convolution. Put a severity on a lattice of N buckets, transform, raise to the frequency's probability generating function, invert, and any aggregate mass that should have landed beyond bucket N instead wraps around and lands in the low buckets. That is aliasing, and it is the one error in the transform approach that has nothing to do with discretization: it would be there even if the severity were exactly arithmetic.

Grübel and Hermesmeier (1999) is the paper that made this precise and gave the fix. Their Algorithm 1 is the unadorned transform method. Algorithm 5 adds **exponential tilting**: multiply the discretized severity by $e^{-\theta k}$ before transforming and divide it back out afterwards. Tilting makes the sequence decay fast, so the mass that wraps is negligible before it wraps, and the aliasing error is suppressed by roughly $e^{-\theta N}$. Their Theorem 2 bounds the total aliasing error by the mass of the untruncated aggregate at or beyond the truncation point.

The practical constraint is that θ cannot be taken large: the untilting step multiplies by $e^{+\theta k}$, and for k near N that is a huge factor which amplifies floating point noise. Their rule of thumb is $\theta N \approx 20$.

`aggregate` does not tilt. It controls aliasing by **padding**, which doubles or quadruples the severity vector with zeros so there is nowhere for the wrap to go. Tilting is carried in `aggregate.pedagogy` for exactly this page's purpose, as a teaching device rather than a production path, which makes the paper a test of a road the library deliberately did not take.

17.2. Paper solution and gold standard

Section 3 takes the claim size to be the stable law of index 1/2, following Embrechts et al. (1993). This is `scipy.stats.levy`, concentrated on the positive half line, and it has no mean. Frequency is Poisson with $\lambda t = 20$. The lattice is $N = 1024$ intervals of length 1, so the truncation point is $x_0 = Nh = 1024$.

The index 1/2 is what makes an exact answer available. A stable law of index 1/2 satisfies $f_\alpha * f_\beta = f_{\alpha+\beta}$ in the paper's scale parameterization, so the n-fold convolution is another Lévy with a known scale, and the compound probability $u([x - 0.5,\ x + 0.5])$ can be summed in closed form.

Table 1, Figure 17.1, gives six columns at $x = 1, 10, 100, 1000$: the true value, the Panjer recursion on the same lattice, Algorithm 1, and Algorithm 5 at $\theta N = 1, 5, 25$. Table 2, Figure 17.2, restates the last four as log of the ratio to the Panjer value. The text adds two things neither table gives: the sum

of absolute differences between Panjer and each transform algorithm over the whole lattice, and the value Theorem 2's bound takes. Table 17.1 transcribes Table 1.

TABLE I

COMPOUND PROBABILITIES AND APPROXIMATIONS

x	*true*	*Panjer*	*Alg l*	*Alg5a*	*Alg5b*	*Alg5c*
1	1.078E-07	2.462E-07	2.064E-04	7.346E-05	1.560E-06	2.462E-07
10	3.075E-05	3.432E-05	2.380E-04	1.067E-04	3.562E-05	3.432E-05
100	1.156E-03	1.156E-03	1.321E-03	1.215E-03	1.157E-03	1.156E-03
1000	2.013E-04	2.012E-04	2.134E-04	2.056E-04	2.013E-04	2.012E-04

Figure 17.1.: Grübel and Hermesmeier (1999), Table 1.

TABLE 2

LOGARITHMIC DIFFERENCES

x	*Alg1*	*Alg5a*	*Alg5b*	*Alg5c*
1	6.732	5.698	1.84659	0.0000000110
10	1.936	1.134	0.03715	0.0000000001
100	0.134	0.050	0.00092	0.0000000000
1000	0.059	0.022	0.00040	0.0000247729

Figure 17.2.: Grübel and Hermesmeier (1999), Table 2.

```
%xmode minimal
import numpy as np
import pandas as pd
from aggregate.pedagogy import gh_tilting_exhibit, tilted_aggregate_density
from bookstyle import GT, build, show

N, BS, LAM = 1024, 1.0, 20
XS = [1, 10, 100, 1000]
TILTS = [None, 1 / 1024, 5 / 1024, 25 / 1024]

TABLE1 = pd.DataFrame(
    {'true': [1.078e-7, 3.075e-5, 1.156e-3, 2.013e-4],
     'Panjer': [2.462e-7, 3.432e-5, 1.156e-3, 2.012e-4],
     'Alg1': [2.064e-4, 2.380e-4, 1.321e-3, 2.134e-4],
     'Alg5a': [7.346e-5, 1.067e-4, 1.215e-3, 2.056e-4],
     'Alg5b': [1.560e-6, 3.562e-5, 1.157e-3, 2.013e-4],
     'Alg5c': [2.462e-7, 3.432e-5, 1.156e-3, 2.012e-4]}, index=XS)
TABLE1.index.name = 'x'
TABLE2 = pd.DataFrame(
    {'Alg1': [6.732, 1.936, 0.134, 0.059],
     'Alg5a': [5.698, 1.134, 0.050, 0.022],
     'Alg5b': [1.84659, 0.03715, 0.00092, 0.00040],
     'Alg5c': [1.10e-8, 1e-10, 0.0, 2.47729e-5]}, index=XS)
TABLE2.index.name = 'x'
L1_PAPER = [0.0714, 0.0255, 0.000459, 3.121e-7]
THEOREM2_PAPER = 0.4641
GT(TABLE1, formatters={c: '{x:.4g}' for c in TABLE1.columns})
```

```
Exception reporting mode: Minimal
```

Table 17.1.: Grübel and Hermesmeier (1999), Table 1 as printed.

x	true	Panjer	Alg1	Alg5a	Alg5b	Alg5c
1	1.078e-07	2.462e-07	0.0002064	7.346e-05	1.56e-06	2.462e-07
10	3.075e-05	3.432e-05	0.000238	0.0001067	3.562e-05	3.432e-05
100	0.001156	0.001156	0.001321	0.001215	0.001157	0.001156
1,000	0.0002013	0.0002012	0.0002134	0.0002056	0.0002013	0.0002012

17.3. The `aggregate` reproduction

The declaration is a one liner. `levy` takes no shape parameter, so the severity clause is bare.

```
a = build(f'agg Levy {LAM} claim sev levy poisson', update=False)
a.update(log2=16, bs=BS, padding=2, normalize=False)
show(a)
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
Levy: aggregate PMF deficit 6.226e-02 (Σp = 1 − 6.226e-02 < 1); forwards and backwards S diverge
by the deficit (forwards > backwards). Further occurrences this session are suppressed; each
object records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

```
agg Levy
  20 claims
  sev levy
  poisson
```

The whole of Table 1 is assembled by one call, which computes the closed-form column from Lévy stability and sweeps the tilts, giving Table 17.2.

```
got = gh_tilting_exhibit(claims=LAM, bs=BS, index=XS)
got.columns = ['Panjer', 'true', 'Alg1', 'Alg5a', 'Alg5b', 'Alg5c']
got = got[TABLE1.columns]
got.index = XS
got.index.name = 'x'
GT(pd.concat({'paper': TABLE1, 'aggregate': got}, axis=1),
   formatters={c: '{x:.4g}' for c in
               pd.concat({'paper': TABLE1, 'aggregate': got}, axis=1).columns})
```

Table 17.2.: Table 1 rebuilt, all six columns at the four tabulated points, beside the printed figures.

	paper						aggregate					
x	true	Panjer	Alg1	Alg5a	Alg5b	Alg5c	true	Panjer	Alg1	Alg5a	Alg5b	Alg5c
1	1.078e-07	2.462e-07	0.0002064	7.346e-05	1.56e-06	2.462e-07	1.078e-07	2.462e-07	0.0002064	7.346e-05	1.56e-06	2.462e-07
10	3.075e-05	3.432e-05	0.000238	0.0001067	3.562e-05	3.432e-05	3.075e-05	3.432e-05	0.000238	0.0001067	3.562e-05	3.432e-05
100	0.001156	0.001156	0.001321	0.001215	0.001157	0.001156	0.001155	0.001156	0.001321	0.001215	0.001157	0.001156
1,000	0.0002013	0.0002012	0.0002134	0.0002056	0.0002013	0.0002012	0.0002013	0.0002012	0.0002134	0.0002056	0.0002013	0.0002012

```
rel = (got / TABLE1 - 1).abs()
GT(pd.DataFrame([{'column': c, 'worst relative difference': float(rel[c].max())}
                 for c in TABLE1.columns]).set_index('column'),
   formatters={'worst relative difference': '{x:.1e}'})
```

Table 17.3.: Worst relative difference from Table 1, column by column.

column	worst relative difference
true	4.5e-04
Panjer	1.6e-04
Alg1	2.3e-04
Alg5a	1.8e-04
Alg5b	1.2e-04
Alg5c	1.6e-04

By Table 17.3 every one of the 24 published values comes back inside the four significant figures printed. Table 2 restates four of those columns as logarithms of ratios, which is a sharper test in the small columns because it removes the shared scale, in Table 17.4.

```
lg = np.log(got[['Alg1', 'Alg5a', 'Alg5b', 'Alg5c']].div(got['Panjer'], axis=0))
GT(pd.concat({'paper': TABLE2, 'aggregate': lg}, axis=1), float_format=6)
```

Table 17.4.: Table 2 rebuilt, each algorithm as the log of its ratio to the Panjer value, beside the printed figures.

	paper				aggregate			
x	Alg1	Alg5a	Alg5b	Alg5c	Alg1	Alg5a	Alg5b	Alg5c
1	6.732000	5.698000	1.846590	0.000000	6.731818	5.698487	1.846586	0.000000
10	1.936000	1.134000	0.037150	0.000000	1.936384	1.133992	0.037151	0.000000
100	0.134000	0.050000	0.000920	0.000000	0.133675	0.049719	0.000917	0.000000
1,000	0.059000	0.022000	0.000400	0.000025	0.058864	0.021650	0.000396	-0.000016

17.3.1. The two numbers that are not in a table

The text reports the sum of absolute differences between Panjer and each transform algorithm, over the whole lattice rather than the four tabulated points. That is a much stronger check than any single row, since it uses all 1024 buckets. Table 17.5 is the comparison.

```
ref = a.density_df.p_total.to_numpy()[:N].copy()   # padded, so unaliased
rows = []
for tilt, paper in zip(TILTS, L1_PAPER):
    v = tilted_aggregate_density(a, log2=10, bs=BS, padding=0,
                                 tilt=tilt).to_numpy()[:N]
    name = 'Alg1 (no tilt)' if tilt is None else f'Alg5, theta N = {tilt * N:.0f}'
    rows.append({'algorithm': name, 'paper': paper,
                 'aggregate': float(np.abs(ref - v).sum())})
l1 = pd.DataFrame(rows).set_index('algorithm')
l1['ratio'] = l1['aggregate'] / l1['paper']
GT(l1, formatters={'paper': '{x:.4g}', 'aggregate': '{x:.4g}', 'ratio': 3})
```

Table 17.5.: Sum of absolute differences from the unaliased density over the whole lattice, for each algorithm.

algorithm	paper	aggregate	ratio
Alg1 (no tilt)	0.0714	0.07138	1.000
Alg5, theta N = 1	0.0255	0.0255	1.000
Alg5, theta N = 5	0.000459	0.0004592	1.001
Alg5, theta N = 25	3.121e-07	2.013e-07	0.645

In Table 17.5 three of the four match to the printed precision: 0.07138 against 0.0714, 0.02550 against 0.0255, and 0.00045924 against 0.000459. The fourth does not, and the paper says why before it is asked. At $\theta N = 25$ the untilting step multiplies by factors "of the order 10^{10}" and "rounding errors begin to be important". Both values are at the 10^{-7} level, four orders of magnitude below the next one up, so the comparison there is between two different floating point noise floors rather than between two algorithms.

Theorem 2 bounds the aliasing error by the mass of the untruncated aggregate at or beyond the truncation point, which is exactly the survival function at x_0, evaluated in Table 17.6.

```
# `tilted_aggregate_density` updates the object it is handed, so `a` is sitting
# on the coarse log2=10 grid after the sweep above. Put it back first.
a.update(log2=16, bs=BS, padding=2, normalize=False)
p = a.density_df.p_total.to_numpy()
GT(pd.DataFrame([
    {'quantity': 'mass on the 2^16 grid', 'value': float(p.sum())},
    {'quantity': 'deficit above the grid', 'value': 1 - float(p.sum())},
    {'quantity': 'mass in buckets [1024, grid end)', 'value': float(p[N:].sum())},
    {'quantity': 'total at or beyond 1024, the Theorem 2 bound',
     'value': float(p[N:].sum()) + 1 - float(p.sum())},
    {'quantity': 'S(1023) read off density_df',
     'value': float(a.density_df.S.loc[N - 1])},
    {'quantity': 'paper', 'value': THEOREM2_PAPER},
]).set_index('quantity'), float_format=6)
```

Table 17.6.: The Theorem 2 bound assembled from the on-grid tail and the mass above the grid, against the paper's 0.4641.

quantity	value
mass on the 2^16 grid	0.937738
deficit above the grid	0.062262
mass in buckets [1024, grid end)	0.401806
total at or beyond 1024, the Theorem 2 bound	0.464068
S(1023) read off density_df	0.464068
paper	0.464100

Table 17.6 gives 0.4641, to the four figures printed. Getting there needs one thing said out loud: the severity has no mean, so at `bs = 1` even a 2^{16} grid carries only 93.8% of the aggregate, and the 6.2% above it is part of the bound. Summing the on-grid tail alone gives 0.4018 and misses.

17.4. Comments

17.4.1. The bound is loose, and the paper knows why

The actual aliasing error in Table 17.5 is 0.0714 against the 0.4641 bound of Table 17.6, a factor of 6.5. The paper attributes this to the claim size being subexponential, and it is the right diagnosis: for a subexponential severity the aggregate tail is dominated by the single largest claim, so the mass beyond the truncation point is concentrated far beyond it, and wrapping mass that lands at $x + 1024$ for large x mostly re-wraps out of the low buckets rather than into them. A bound that counts every unit of escaping mass as a unit of contamination cannot see that.

The practical consequence is that Theorem 2 is a certificate rather than a diagnostic. It tells you when you are definitely safe. It does not tell you how much error you actually have, and for a heavy tail it will talk you into a finer grid than you need.

17.4.2. The discretization error is the larger of the two, and it is the other paper

The paper's closing observation about Table 1 is easy to miss and is the most interesting thing in it: "a comparison of the values in the second and third column of Table 1 shows that there is a pronounced discretization error, again most notable for small x-values."

```
GT(pd.DataFrame([
    {'x': x, 'true': TABLE1.loc[x, 'true'], 'Panjer': TABLE1.loc[x, 'Panjer'],
     'Panjer / true': TABLE1.loc[x, 'Panjer'] / TABLE1.loc[x, 'true']}
    for x in XS]).set_index('x'),
   formatters={'true': '{x:.4g}', 'Panjer': '{x:.4g}', 'Panjer / true': 4})
```

Table 17.7.: The true value against the Panjer value on the same lattice, isolating discretization error from aliasing.

x	true	Panjer	Panjer / true
1	1.078e-07	2.462e-07	2.2839
10	3.075e-05	3.432e-05	1.1161
100	0.001156	0.001156	1.0000
1,000	0.0002013	0.0002012	0.9995

In Table 17.7, at $x = 1$ the recursion on a lattice of span 1 is wrong by a factor of 2.3, and no amount of tilting or padding touches it, because both columns use the same discretized severity. Tilting fixes aliasing, and Table 1's own second and third columns show that aliasing was not the binding constraint at the left hand end. That is the subject of Part II, Grübel and Hermesmeier (2000), and the same distinction is why `aggregate` reports discretization and aliasing as separate diagnostics rather than one error budget.

17.4.3. Why the library pads instead

Padding and tilting solve the same problem, and the reproduction shows they solve it about equally well: the paper's own $\theta N = 25$ column and the padded column agree to four figures at every x. Padding wins on two counts that only show up in production.

It has no free parameter. The tilt has to be chosen, the good range is bounded below by the aliasing it has to kill and above by the floating point amplification in the untilting step, and the paper's own $\theta N \approx 20$ rule is a compromise between two failure modes. Padding has one setting, "how much", and more is monotonically safer.

And it does not amplify. The fourth row of Table 17.5 is the whole argument: at the tilt the paper recommends, the answer is already limited by rounding noise introduced by the method itself rather than by the aliasing it removed. A doubling of the vector length costs one factor of two in the transform and introduces nothing.

18. Pittarello, Luini and Marchione's GEMAct package

Reproducing Pittarello, Luini and Marchione (2024)

18.1. Context and problem

Pittarello et al. (2024) describes GEMAct, a Python package for non-life (re)insurance modeling, in the same *Annals of Actuarial Science* software stream as `aggregate`. It is built on the collective risk model and computes the aggregate distribution by FFT, Panjer recursion or Monte Carlo, with four severity discretization methods, coverage modifiers including reinstatements, the AEP algorithm for sums of dependent risks, and a reserving module. Section 3.7 compares its FFT with `aggregate` 0.9.3, on contracts taken from Parodi (2015).

The paper therefore offers three things to check: a set of small worked examples with printed answers, a table of reference moments for a lognormal portfolio, and a published benchmark of this library. This chapter checks the numbers and refreshes the benchmark on one machine with both packages. It is not a contest: the two packages compute the same thing, and the interesting findings are about what the paper's own tables measure.

18.2. Paper solution and gold standard

18.2.1. The worked examples of Section 3.5

Section 3.5 prints five numbers for four small contracts, each with Poisson claim counts counted from the ground up (`threshold = 0`). Table 18.1 lists them with the paper's printed digits. Two are closed forms; the other three are read off a distribution computed by FFT on 2^{17} nodes.

```
%xmode minimal
import json
import time
import warnings

import numpy as np
import pandas as pd
from bookstyle import BOOK_ROOT, GT, build, show

warnings.filterwarnings('ignore', category=RuntimeWarning)

ANCHORS = pd.DataFrame([
    ('XL, mean', 'Poisson 4, gamma(5), 20 xs 5', 'closed form', 3.50934614394912),
    ('XL, CV', 'Poisson 4, gamma(5), 20 xs 5', 'closed form', 1.0001481667319252),
```

```
    ('XL, skewness', 'Poisson 4, gamma(5), 20 xs 5', 'closed form', 1.3814094309741256),
    ('Reinstatements, premium',
     'Poisson 0.5, Lomax(100, 1.2), 100 xs 0, aggregate deductible 100, 2 at 100%',
     'FFT', 4.319350355177216),
    ('Layer 1, premium', 'Poisson 0.5, GPD(0.834, 83.34), 100 xs 100, 50% share',
     'closed form', 8.479087307840043),
    ('Layer 2, premium', 'same, 200 xs 100, 2 reinstatements at 60%',
     'FFT', 25.99131088702302),
    ('Layer 3, premium', 'same, 100 xs 100, aggregate cover 200',
     'FFT', 16.88704720494799),
], columns=['quantity', 'contract', 'paper method', 'paper value']).set_index('quantity')
GT(ANCHORS, formatters={'paper value': '{x:.15g}'})
```

```
Exception reporting mode: Minimal
```

Table 18.1.: The Section 3.5 worked examples and the paper's printed answers.

quantity	contract	paper method	paper value
XL, mean	Poisson 4, gamma(5), 20 xs 5	closed form	3.50934614394912
XL, CV	Poisson 4, gamma(5), 20 xs 5	closed form	1.00014816673193
XL, skewness	Poisson 4, gamma(5), 20 xs 5	closed form	1.38140943097413
Reinstatements, premium	Poisson 0.5, Lomax(100, 1.2), 100 xs 0, aggregate deductible 100, 2 at 100%	FFT	4.31935035517722
Layer 1, premium	Poisson 0.5, GPD(0.834, 83.34), 100 xs 100, 50% share	closed form	8.47908730784004
Layer 2, premium	same, 200 xs 100, 2 reinstatements at 60%	FFT	25.991310887023
Layer 3, premium	same, 100 xs 100, aggregate cover 200	FFT	16.887047204948

18.2.2. Tables 1 and 2: the lognormal portfolio

Section 3.6 uses Parodi's lognormal severity with mean 84,541.68 and standard deviation 177,728.3 (shape 1.3, scale 36,315.49) and Poisson 3 claims in excess of a 10,000 deductible. Table 1, Figure 18.1, reports the relative errors in the mean, CV and skewness of FFT, recursion and Monte Carlo for bucket sizes $h \in \{50, 100, 200, 400\}$ and $m \in \{2^{14}, 2^{16}, 2^{18}\}$ nodes, against closed form reference values, and concludes that "when h increases, the error reduces".

Table 1. Accuracy and speed of the approximation of the aggregate loss distribution using fast Fourier transform (FFT), the recursive formula (recursion), and the Monte Carlo (MC) simulation when varying the number of nodes (m), the discretization step (h), and number of simulations. The upper table contains the reference values obtained from the closed-form solutions. The lower table reports the execution times in second and the relative errors with respect to the reference values

	Mean	CoV	Skewness
Reference values	268837	1.35520	7.02399

Method	Time (sec.)	Mean	CoV	Skewness
FFT ($h = 50, m = 2^{**}14$)	0.002	−2.76451e-01	−3.00646e-01	−8.06071e-01
FFT ($h = 100, m = 2^{**}14$)	0.002	−8.96804e-02	−1.98006e-01	−7.15193e-01
FFT ($h = 200, m = 2^{**}14$)	0.003	−2.19140e-02	−1.01312e-01	−5.69628e-01
FFT ($h = 400, m = 2^{**}14$)	0.003	−4.41036e-03	−5.88141e-02	−3.47025e-01
FFT ($h = 50, m = 2^{**}16$)	0.010	−2.19153e-02	−1.01317e-01	−5.69635e-01
FFT ($h = 100, m = 2^{**}16$)	0.009	−4.40959e-03	−5.88200e-02	−3.47044e-01
FFT ($h = 200, m = 2^{**}16$)	0.008	−7.24988e-04	−1.36155e-02	−2.37709e-01
FFT ($h = 400, m = 2^{**}16$)	0.007	−9.66263e-05	−3.49930e-03	−1.14264e-01
FFT ($h = 50, m = 2^{**}18$)	0.037	−7.24464e-04	−1.36142e-02	−2.37694e-01
FFT ($h = 100, m = 2^{**}18$)	0.035	−9.34546e-05	−3.46164e-03	−1.13558e-01
FFT ($h = 200, m = 2^{**}18$)	0.036	−3.61429e-06	−3.69322e-04	−3.07951e-02
FFT ($h = 400, m = 2^{**}18$)	0.041	−1.73491e-06	−2.80936e-05	−6.37228e-03
Recursion ($h = 50, m = 2^{**}14$)	1.397	−2.76451e-01	−3.00646e-01	−8.06071e-01
Recursion ($h = 100, m = 2^{**}14$)	1.368	−8.96804e-02	−1.98006e-01	−7.15193e-01
Recursion ($h = 200, m = 2^{**}14$)	1.398	−2.19140e-02	−1.01312e-01	−5.69628e-01
Recursion ($h = 400, m = 2^{**}14$)	1.390	−4.41046e-03	−4.16016e-02	−4.00405e-01
Recursion ($h = 50, m = 2^{**}16$)	13.401	−2.19154e-02	−1.01317e-01	−5.69636e-01
Recursion ($h = 100, m = 2^{**}16$)	12.606	−4.41006e-03	−4.16096e-02	−4.00430e-01
Recursion ($h = 200, m = 2^{**}16$)	11.815	−7.25738e-04	−1.36242e-02	−2.37782e-01
Recursion ($h = 400, m = 2^{**}16$)	11.771	−9.62811e-05	−3.49125e-03	−1.14125e-01
Recursion ($h = 50, m = 2^{**}18$)	448.168	−7.25465e-04	−1.36260e-02	−2.37794e-01
Recursion ($h = 100, m = 2^{**}18$)	464.185	−9.47567e-05	−3.49381e-03	−1.14141e-01
Recursion ($h = 200, m = 2^{**}18$)	460.200	−1.00139e-05	−6.92170e-04	−4.30010e-02
Recursion ($h = 400, m = 2^{**}18$)	462.339	−2.41068e-06	−1.03582e-04	−1.25319e-02
MC ($2^{**}14$ sim.)	0.243	−1.37610e-02	−3.67078e-02	−3.77663e-01
MC ($2^{**}16$ sim.)	0.985	−5.53096e-03	−2.44089e-02	−3.49520e-01
MC ($2^{**}18$ sim.)	3.922	−3.02675e-03	−6.31524e-03	−8.93550e-02
MC ($2^{**}20$ sim.)	15.901	−1.20257e-03	−5.21299e-03	−7.49223e-02

Figure 18.1.: Pittarello, Luini and Marchione (2024), Table 1.

Section 3.7, Figure 18.2, prices four contracts on the same model with 2^{22} nodes: no reinsurance, an XL of 1,000,000 xs 10,000, a stop loss of 1,000,000 xs 50,000 on the aggregate, and the XL with the same aggregate terms. It times both packages and prints reference values for the first two.

Table 2. Comparison for different contracts of aggregate and gemact implementation of the aggregate loss distribution computation via FFT. When there are no aggregate coverage modifiers, reference values are given in addition to estimated ones. For the XL and the XL w/agg. contracts, individual conditions are $c = 1000000$ and $d = 10000$; for the SL and the XL w/agg. contracts, aggregate coverage modifiers are $u = 1000000$ and $v = 50000$. Execution times are expressed in seconds

Contract	Library	Time (sec.)	Mean	CoV	Skewness
No reinsurance	Reference value	–	253625	1.34406	7.28410
	gemact	0.9581	253625	1.34406	7.27745
	aggregate	0.8102	253625	1.34406	7.28346
XL	Reference value	–	256355	1.10772	2.08525
	gemact	0.9384	256355	1.10772	2.08525
	aggregate	0.8445	256354	1.10773	2.08527
SL	Reference value	–	–	–	–
	gemact	0.9168	194143	1.24438	1.73658
	aggregate	2.4254	194143	1.24438	1.73658
XL w/agg.	Reference value	–	–	–	–
	gemact	0.9388	206363	1.22464	1.62335
	aggregate	2.5221	206363	1.22465	1.62336

Figure 18.2.: Pittarello, Luini and Marchione (2024), Table 2.

The authors' own code for both tables is in the `vignette` folder of the GEMAct repository at commit 808f9d0 (2023-12-21), since removed from the main branch. It settles every setting the text leaves open: Table 1 discretizes the severity on m nodes at step h and computes the aggregate on m nodes, and Table 2 uses a step of 500, with `padding=0` for `aggregate`. The two tables count claims differently. Table 1 and the XL rows count 3 claims in excess of 10,000, which is DecL's default conditional layer, and the gross and stop loss rows count 3 ground up claims.

18.2.3. Table 3: the AEP algorithm

Section 4 implements the AEP algorithm of Arbenz et al. (2011) for the distribution of a sum of dependent risks, and Table 3, Figure 18.3, checks it on Arbenz's examples: Pareto marginals coupled by a Clayton copula in two to five dimensions, with reference values copied from Arbenz's Tables 3 to 6.

Table 3. Accuracy and speed of the cdf calculation using the AEP algorithm and the Monte Carlo (MC) simulation approach for the sum of Pareto random variables coupled with a Clayton copula for different dimensions and quantiles. The upper table contains the reference values from Arbenz *et al.* (2011). The lower table reports the execution times in seconds and the relative errors with respect to the reference values. The time column on the left is about the execution times of **gemact**, while the time column on the right (labeled with a *) lists those of the original manuscript

Reference values:					
Quantile	$d=2$	$d=3$	Quantile	$d=4$	$d=5$
$s=10^0$	0.315835041363441	0.190859309689430	$s=10^1$	0.983690398913354	0.983659549676444
$s=10^2$	0.983690398913354	0.983659549676444	$s=10^2$	0.983690398913354	0.983659549676444
$s=10^4$	0.999748719229367	0.999748708770280	$s=10^3$	0.983690398913354	0.983659549676444
$s=10^6$	0.999996018908404	0.999996018515584	$s=10^4$	0.983690398913354	0.983659549676444

Dim.	Method	Time (sec.)	Time* (sec.)	$s=10^0$	$s=10^2$	$s=10^4$	$s=10^6$
$d=2$	AEP (7 iter.)	0.02	0.01	4.62e-11	−1.86e-09	4.13e-08	1.22e-09
	AEP (10 iter.)	0.06	0.06	9.03e-14	5.56e-13	−6.38e-11	3.88e-11
	AEP (13 iter.)	0.95	1.61	6.91e-14	5.04e-13	1.10e-12	4.47e-13
	AEP (16 iter.)	24.0	49.25	−1.72e-13	4.39e-13	1.26e-12	5.66e-13
	MC (10**4 sim.)	0.01	–	−1.00e-02	−2.13e-04	4.87e-05	−3.98e-06
	MC (10**5 sim.)	0.03	–	−8.07e-04	−1.01e-04	1.87e-05	1.60e-05
	MC (10**6 sim.)	0.28	–	−7.90e-05	−3.82e-05	−2.28e-06	1.02e-06
	MC (10**7 sim.)	2.76	–	1.71e-04	6.20e-06	−6.18e-06	−6.81e-07
$d=3$	AEP (7 iter.)	0.04	0.02	−4.61e-06	−1.15e-06	1.12e-06	1.83e-08
	AEP (9 iter.)	0.28	0.41	−1.73e-07	−3.06e-07	2.39e-07	4.26e-09
	AEP (11 iter.)	4.0	6.65	−6.89e-09	−1.13e-08	2.95e-08	7.66e-10
	AEP (13 iter.)	68.4	118.50	−9.12e-10	−1.29e-09	−6.19e-09	−1.09e-10
	MC (10**4 sim.)	0.01	–	5.22e-02	−1.43e-04	−5.13e-05	−3.98e-06
	MC (10**5 sim.)	0.04	–	7.59e-03	−3.56e-04	1.87e-05	−3.98e-06
	MC (10**6 sim.)	0.36	–	−3.87e-03	1.07e-05	−2.29e-06	−1.98e-06
	MC (10**7 sim.)	4.1	–	7.77e-04	4.31e-05	1.61e-06	−4.81e-07
Dim.	Method	Time (sec.)	Time* (sec.)	$s=10^1$	$s=10^2$	$s=10^3$	$s=10^4$
$d=4$	AEP (4 iter.)	0.05	0.03	−1.13e-04	5.03e-04	7.39e-05	9.30e-06
	AEP (5 iter.)	0.38	0.47	−4.45e-04	1.56e-04	2.71e-05	3.42e-06
	AEP (6 iter.)	5.23	7.15	−4.80e-04	−5.09e-05	−3.69e-06	−4.52e-07
	AEP (7 iter.)	83.55	107.70	−4.55e-04	−1.56e-04	−2.26e-05	−2.85e-06
	MC (10**4 sim.)	0.01	–	3.18e-03	−1.92e-03	3.51e-04	4.42e-04
	MC (10**5 sim.)	0.05	–	6.57e-04	−1.71e-04	1.03e-05	−7.74e-06
	MC (10**6 sim.)	0.46	–	−6.00e-04	1.34e-05	−4.75e-06	−2.67e-05
	MC (10**7 sim.)	4.55	–	−2.23e-04	−1.24e-04	−3.29e-05	−5.04e-06
$d=5$	AEP (3 iter.)	0.03	0.01	−4.72e-03	−5.16e-05	5.24e-06	7.22e-07
	AEP (4 iter.)	0.15	0.20	−6.87e-04	3.63e-04	5.30e-05	6.68e-06
	AEP (5 iter.)	2.66	4.37	−1.77e-04	1.94e-04	2.84e-05	3.57e-06
	AEP (6 iter.)	65.61	92.91	−5.14e-13	−9.37e-12	1.44e-11	−1.17e-11
	MC (10**4 sim.)	0.01	–	5.38e-03	−1.67e-03	−3.70e-04	1.40e-04
	MC (10**5 sim.)	0.06	–	−7.25e-04	2.58e-04	−9.29e-06	−9.02e-05
	MC (10**6 sim.)	0.58	–	6.14e-04	−1.89e-04	−1.93e-05	5.81e-06
	MC (10**7 sim.)	5.65	–	5.29e-04	−2.68e-04	−7.70e-05	−1.02e-05

Figure 18.3.: Pittarello, Luini and Marchione (2024), Table 3.

18.3. The `aggregate` reproduction

18.3.1. The worked examples

The XL is one line, with ! making the layer unconditional so the count of 4 is ground up.

```
xl = build('agg PLM.XL 4 claims 20 xs 5 sev gamma 5 ! poisson')
show(xl)
```

```
agg PLM.XL
  4 claims
  20 xs 5
  sev gamma 5 !
  poisson
```

The three layers share a generalized Pareto severity whose tail index is $1/0.834 = 1.2$, so the gross aggregate has no variance and the library asks for an explicit grid. The 50% share is a scale factor on the mean, and the aggregate cover is a clause.

```
layer1 = build('agg PLM.Layer1 0.5 claims 100 xs 100 sev 83.34 * genpareto 0.834 ! poisson')
layer3 = build('agg PLM.Layer3 0.5 claims 100 xs 100 sev 83.34 * genpareto 0.834 ! poisson '
               'aggregate ceded to 200 xs 0')
show(layer3)
```

```
agg PLM.Layer3
  0.5 claims
  100 xs 100
  sev 83.34 * genpareto 0.834 !
  poisson
  aggregate ceded to
    200 xs 0
```

The 60% reinstatements are a clause on an occurrence layer in a P&L program. Reinstatement premium is proportional to the deposit, so one build at a deposit of 1 gives the premium leg's mean $1+k$, and Sundt's pure premium is the expected recovery divided by it. The layer sits on a gross distribution with a power tail, so the grid has to reach far enough for the mass above it to be negligible: 2^{24} buckets of $1/4$.

```
LAYER2 = ('xpnl PLM.Layer2 1 premium less '
          'agg PLM.Layer2Gross 0.5 claims sev 83.34 * genpareto 0.834 '
          'occurrence net of 200 xs 100 deposit 1 reinstatements 2 at 60% as "Layer" '
          'poisson')
# The reinstatement joint is sized by the library's budget and `build` does
# not forward a finer joint grid, so it warns that the 200 wide layer spans
# one bucket of the joint's ceded axis. The premium still lands within 2.5e-6
# of the paper's, so the warning is silenced here rather than printed.
from aggregate.constants import CoarseJointGridWarning
with warnings.catch_warnings():
    warnings.simplefilter('ignore', CoarseJointGridWarning)
    layer2 = build(LAYER2, bs=1 / 4, log2=24, normalize=False)
show(LAYER2)
```

```
xpnl PLM.Layer2 1 premium less agg PLM.Layer2Gross 0.5 claims sev 83.34 * genpareto 0.834
occurrence net of 200 xs 100 deposit 1 reinstatements 2 at 60% as "Layer" poisson
```

The reinstatement example of Section 3.5 combines the reinstatements with an aggregate deductible of 100. A reinstatement clause decorates a single occurrence layer and has no aggregate deductible,

so this one is computed from Sundt's formula, the paper's eq. (7), using limited expectations of the layer's aggregate distribution S:

$$P = \frac{\mathsf{P}[(S-100)^+ \wedge 300]}{1 + \frac{1}{100}\left(\mathsf{P}[(S-100)^+ \wedge 100] + \mathsf{P}[(S-200)^+ \wedge 100]\right)}.$$

```
rs = build('agg PLM.RS 0.5 claims 100 xs 0 sev 100 * pareto 1.2 - 100 ! poisson',
           bs=1 / 64, log2=18)
show(rs)
```

```
agg PLM.RS
  0.5 claims
  100 xs 0
  sev 100 * pareto 1.2 + -100 !
  poisson
```

Table 18.2 sets the seven numbers against the paper's, with the 60% reinstatement layer computed a second way in the last row.

```
def theory(a):
    """Closed form mean, CV and skewness from the validation frame."""
    row = a.validation_df.loc['Agg']
    return float(row['EX']), float(row['CV']), float(row['Sk'])


def lev(a, limit, attach=0.0):
    """P[(S - attach)^+ and limit] off the density."""
    x = a.density_df.loss.values
    return float((np.minimum(np.maximum(x - attach, 0), limit) *
    a.density_df.p_total.values).sum())


def leg(pnl, label):
    """One ledger leg's expected value, off the P&L's economic frame."""
    return float(pnl.economic_df.xs(label, level='Label').iloc[0]['EX'])


rs_premium = lev(rs, 300, 100) / (1 + (lev(rs, 100, 100) + lev(rs, 100, 200)) / 100)
anchors = ANCHORS[['paper value']].copy()
anchors['aggregate'] = [*theory(xl), rs_premium, 0.5 * theory(layer1)[0],
                        leg(layer2, 'Layer recovery') / -leg(layer2, 'Layer premium'),
                        layer3.est_m]
# Layer 2 again, by Sundt's formula on the layer's own aggregate distribution
layer2_net = build('agg PLM.Layer2Net 0.5 claims 200 xs 100 sev 83.34 * genpareto 0.834 !
poisson',
                   bs=1 / 64, log2=18)
anchors.loc['Layer 2, premium, by formula'] = [
    ANCHORS.loc['Layer 2, premium', 'paper value'],
    lev(layer2_net, 600) / (1 + 0.6 / 200 * (lev(layer2_net, 200) + lev(layer2_net, 200, 200)))]
anchors['relative difference'] = anchors['aggregate'] / anchors['paper value'] - 1
GT(anchors, formatters={'paper value': '{x:.15g}', 'aggregate': '{x:.15g}',
                        'relative difference': '{x:.2e}'})
```

Table 18.2.: The Section 3.5 worked examples recomputed, against the printed values.

quantity	paper value	aggregate	relative difference
XL, mean	3.50934614394912	3.50934614394911	-3.11e-15
XL, CV	1.00014816673193	1.00014816673193	4.00e-15
XL, skewness	1.38140943097413	1.38140943097412	-2.89e-15
Reinstatements, premium	4.31935035517722	4.3193503742406	4.41e-09
Layer 1, premium	8.47908730784004	8.47908730784005	8.88e-16
Layer 2, premium	25.991310887023	25.9912470455551	-2.46e-06
Layer 3, premium	16.887047204948	16.8870471983845	-3.89e-10
Layer 2, premium, by formula	25.991310887023	25.9913108831712	-1.48e-10

The two closed forms in Table 18.2 agree to rounding, as they must, since both packages evaluate the same formula. The three FFT values agree to between 4×10^{-10} and 2.5×10^{-6}, inside the precision of an FFT on 2^{17} nodes. The largest gap is the reinstatement layer, whose premium the P&L computes on a joint distribution of gross and ceded loss that the library sizes coarsely, a single bucket across the 200 wide layer. Sundt's formula applied to the layer's own aggregate distribution, the last row, closes the gap. The count convention is confirmed by the match: with `threshold = 0` the count is ground up, and the layered severity must be unconditional.

18.3.2. The reference values of Tables 1 and 2

The reference rows are closed form moments, which `aggregate` reports as the theoretical columns of `validation_df`. The programs differ only in the layer and in whether the count is conditional.

```
SEV = 'sev lognorm 84541.68 cv 2.1022565437545127'
T1 = f'agg PLM.Table1 3 claims inf xs 10000 {SEV} poisson'
CONTRACTS = {
    'No reinsurance': f'agg PLM.Gross 3 claims {SEV} poisson',
    'XL': f'agg PLM.XL1M 3 claims 1000000 xs 10000 {SEV} poisson',
    'SL': f'agg PLM.SL 3 claims {SEV} poisson aggregate ceded to 1000000 xs 50000',
    'XL with aggregate terms': (f'agg PLM.XLAgg 3 claims 1000000 xs 10000 {SEV} poisson '
                                'aggregate ceded to 1000000 xs 50000'),
}
t1 = build(T1)
built = {name: build(program) for name, program in CONTRACTS.items()}
PRINTED_REFERENCE = {'Table 1': (268837, 1.35520, 7.02399),
                     'Table 2, no reinsurance': (253625, 1.34406, 7.28410),
                     'Table 2, XL': (256355, 1.10772, 2.08525)}
reference = pd.DataFrame(
    [(row, *printed, *theory(a)) for (row, printed), a in zip(
        PRINTED_REFERENCE.items(), [t1, built['No reinsurance'], built['XL']])],
    columns=['row', 'printed mean', 'printed CV', 'printed skewness',
             'mean', 'CV', 'skewness']).set_index('row')
GT(reference, formatters={'printed mean': '{x:,.0f}', 'mean': '{x:,.1f}',
                          'printed CV': 5, 'CV': 6, 'printed skewness': 5, 'skewness': 6})
```

Table 18.3.: The reference rows of Tables 1 and 2 against the library's closed form moments.

row	printed mean	printed CV	printed skewness	mean	CV	skewness
Table 1	268,837	1.35520	7.02399	268,837.0	1.355203	7.023995
Table 2, no reinsurance	253,625	1.34406	7.28410	253,625.0	1.344059	7.284104
Table 2, XL	256,355	1.10772	2.08525	256,355.2	1.107723	2.085248

Every printed digit in Table 18.3 is reproduced.

18.3.3. Table 1 measures the length of the grid

Table 1 varies h and m together, and its errors depend only on their product mh, the length of the grid. Table 18.4 transcribes the FFT and recursion mean errors and sorts them by mh. Rows with the same mh sit together and agree to three or more figures whatever h is, and FFT and recursion agree with each other. An error that two unrelated algorithms share, and that depends only on where the grid ends, is the mass beyond the grid. It is not discretization error.

```
TABLE1 = pd.DataFrame([
    # h, log2 m, FFT mean, CV, skewness errors, recursion mean, CV, skewness errors
    (50, 14, -2.76451e-01, -3.00646e-01, -8.06071e-01, -2.76451e-01, -3.00646e-01, -8.06071e-01),
    (100, 14, -8.96804e-02, -1.98006e-01, -7.15193e-01, -8.96804e-02, -1.98006e-01, -7.15193e-01),
    (200, 14, -2.19140e-02, -1.01312e-01, -5.69628e-01, -2.19140e-02, -1.01312e-01, -5.69628e-01),
    (400, 14, -4.41036e-03, -5.88141e-02, -3.47025e-01, -4.41046e-03, -4.16016e-02, -4.00405e-01),
    (50, 16, -2.19153e-02, -1.01317e-01, -5.69635e-01, -2.19154e-02, -1.01317e-01, -5.69636e-01),
    (100, 16, -4.40959e-03, -5.88200e-02, -3.47044e-01, -4.41006e-03, -4.16096e-02, -4.00430e-01),
    (200, 16, -7.24988e-04, -1.36155e-02, -2.37709e-01, -7.25738e-04, -1.36242e-02, -2.37782e-01),
    (400, 16, -9.66263e-05, -3.49930e-03, -1.14264e-01, -9.62811e-05, -3.49125e-03, -1.14125e-01),
    (50, 18, -7.24464e-04, -1.36142e-02, -2.37694e-01, -7.25465e-04, -1.36260e-02, -2.37794e-01),
    (100, 18, -9.34546e-05, -3.46164e-03, -1.13558e-01, -9.47567e-05, -3.49381e-03, -1.14141e-01),
    (200, 18, -3.61429e-06, -3.69322e-04, -3.07951e-02, -1.00139e-05, -6.92170e-04, -4.30010e-02),
    (400, 18, -1.73491e-06, -2.80936e-05, -6.37228e-03, -2.41068e-06, -1.03582e-04, -1.25319e-02),
], columns=['h', 'log2 m', 'FFT mean', 'FFT CV', 'FFT skewness',
            'recursion mean', 'recursion CV', 'recursion skewness'])
TABLE1['m h'] = TABLE1['h'] * 2 ** TABLE1['log2 m']

GEMACT = json.loads((BOOK_ROOT / 'bench' / 'pittarello2024_timing_gemact.json')
                    .read_text(encoding='utf-8'))
rerun = {(r['h'], r['m_log2']): r['mean_error'] for r in GEMACT['table1_fft']}
TABLE1['GEMAct FFT mean, rerun'] = [rerun[(h, k)] for h, k in zip(TABLE1['h'], TABLE1['log2 m'])]
by_support = (TABLE1.sort_values(['m h', 'h'])
              .set_index(['m h', 'h', 'log2 m'])
              [['FFT mean', 'recursion mean', 'GEMAct FFT mean, rerun']])
GT(by_support, formatters={c: '{x:.5e}' for c in by_support.columns},
   aligners={'log2 m': 'r'})
```

Table 18.4.: Table 1's FFT and recursion mean errors, sorted by the length of the grid, with GEMAct 1.2.1's FFT rerun on this machine.

m h	*h*	*log2 m*	FFT mean	recursion mean	GEMAct FFT mean, rerun
819,200	50	14	-2.76451e-01	-2.76451e-01	-2.76451e-01
1,638,400	100	14	-8.96804e-02	-8.96804e-02	-8.96804e-02
3,276,800	50	16	-2.19153e-02	-2.19154e-02	-2.19154e-02
	200	14	-2.19140e-02	-2.19140e-02	-2.19140e-02
6,553,600	100	16	-4.40959e-03	-4.41006e-03	-4.40970e-03
	400	14	-4.41036e-03	-4.41046e-03	-4.41051e-03
13,107,200	50	18	-7.24464e-04	-7.25465e-04	-7.25534e-04
	200	16	-7.24988e-04	-7.25738e-04	-7.24966e-04
26,214,400	100	18	-9.34546e-05	-9.47567e-05	-9.62104e-05
	400	16	-9.66263e-05	-9.62811e-05	-9.60705e-05
52,428,800	200	18	-3.61429e-06	-1.00139e-05	-1.14719e-05
104,857,600	400	18	-1.73491e-06	-2.41068e-06	-1.71377e-06

The authors' code identifies the mechanism. The severity in excess of 10,000 is discretized by mass dispersal on m nodes, so everything above the last node is placed on it: the severity is censored at $(m-1)h$. The aggregate is then computed on m nodes, and the mass of sums beyond mh, wrapped by the transform and suppressed by the exponential tilt, is lost without renormalizing. `aggregate` builds exactly that model, the severity limited to $(m-1)h$ on a grid long enough to hold its aggregate, and drops the mass beyond mh. Table 18.5 compares the result with the recursion column, which has no aliasing to complicate it.

```
t1_mean = theory(t1)[0]


def truncated_mean_error(h, k):
    """Mean error of the m-node model: severity censored at (m-1)h, mass beyond mh dropped."""
    m = 2 ** k
    a = build(f'agg PLM.Cut 3 claims {(m - 1) * h} xs 10000 {SEV} poisson',
              bs=h, log2=k + 3, padding=0)
    x, p = a.density_df.loss.values, a.density_df.p_total.values
    keep = x < m * h
    return float((x[keep] * p[keep]).sum()) / t1_mean - 1


truncation = TABLE1.set_index(['h', 'log2 m'])[['m h', 'recursion mean']].copy()
truncation['aggregate, truncated model'] = [truncated_mean_error(h, k) for h, k in
truncation.index]
truncation['ratio'] = truncation['aggregate, truncated model'] / truncation['recursion mean']
GT(truncation, formatters={'recursion mean': '{x:.5e}', 'aggregate, truncated model': '{x:.5e}',
                           'ratio': 6})
```

Table 18.5.: Table 1's recursion mean errors against the same model built in aggregate with the severity censored at the last node and the aggregate cut at m h.

h	*log2 m*	m h	recursion mean	aggregate, truncated model	ratio
50	14	819,200	-2.76451e-01	-2.76451e-01	1.000000
100	14	1,638,400	-8.96804e-02	-8.96804e-02	1.000000
200	14	3,276,800	-2.19140e-02	-2.19140e-02	1.000001
400	14	6,553,600	-4.41046e-03	-4.41046e-03	1.000001
50	16	3,276,800	-2.19154e-02	-2.19154e-02	1.000000
100	16	6,553,600	-4.41006e-03	-4.41006e-03	0.999999
200	16	13,107,200	-7.25738e-04	-7.25738e-04	1.000000
400	16	26,214,400	-9.62811e-05	-9.62811e-05	1.000000
50	18	13,107,200	-7.25465e-04	-7.25465e-04	1.000000
100	18	26,214,400	-9.47567e-05	-9.47567e-05	1.000000
200	18	52,428,800	-1.00139e-05	-1.00139e-05	0.999997
400	18	104,857,600	-2.41068e-06	-2.41068e-06	1.000001

All twelve recursion mean errors in Table 18.5 are reproduced to the printed digits. The FFT rows match the recursion rows while the truncation error is large, and part company below about 10^{-3}, where the tilted transform's own residue is comparable with the error being measured. GEMAct 1.2.1, rerun on this machine in Table 18.4, reproduces the printed FFT column exactly in the first rows and differs from it in the same deep rows, so those digits depend on details of the run. The CV and skewness errors follow the same pattern; they also depend on how the lost mass enters the moment formula, and are not pursued here.

18.3.4. On a long enough grid the error falls as h^2

Table 1's conclusion reverses once the grid is long enough. Table 18.6 fixes the length of the grid at $2^{16} \times 4{,}000 \approx 2.6 \times 10^8$, the library's own default for this portfolio, and halves the bucket size.

```
t1_cv, t1_skew = theory(t1)[1:]
rows = []
for bs, log2 in [(4000, 16), (2000, 17), (1000, 18), (500, 19)]:
    start = time.perf_counter()
    a = build(T1, bs=bs, log2=log2)
    seconds = time.perf_counter() - start
    rows.append({'bucket size': bs, 'log2': log2, 'mean error': a.est_m / t1_mean - 1,
                 'CV error': a.est_cv / t1_cv - 1, 'skewness error': a.est_skew / t1_skew - 1,
                 'milliseconds': 1000 * seconds})
convergence = pd.DataFrame(rows).set_index('bucket size')
convergence.insert(2, 'mean error ratio',
                   convergence['mean error'].shift(1) / convergence['mean error'])
print(f'default grid: bs={t1.bs:g}, log2={t1.log2}')
GT(convergence, formatters={'mean error': '{x:.3e}', 'CV error': '{x:.3e}',
                            'skewness error': '{x:.3e}', 'mean error ratio': 3,
                            'milliseconds': 0})
```

```
default grid: bs=4000, log2=16
```

Table 18.6.: The Table 1 portfolio on a grid of fixed length, halving the bucket size, against the closed form moments.

bucket size	log2	mean error	mean error ratio	CV error	skewness error	milliseconds
4,000	16	-1.665e-04	—	1.756e-04	-1.720e-03	50
2,000	17	-4.160e-05	4.002	3.937e-05	-1.689e-03	60
1,000	18	-1.041e-05	3.996	5.351e-06	-1.681e-03	76
500	19	-2.616e-06	3.980	-3.150e-06	-1.679e-03	122

Each halving of the bucket in Table 18.6 divides the mean error by four, the $O(h^2)$ rate of Embrechts et al. (1993), from -1.7×10^{-4} at the default bucket of 4,000 to -2.6×10^{-6} at 500. Every row takes well under a second. The skewness error does not move, because it is set by the length of the grid: the third moment of a lognormal with $\sigma = 1.3$ draws on losses out to about 10^{10}.

18.3.5. Table 2, refreshed on one machine

The benchmark was rerun on this machine with the authors' settings, GEMAct 1.2.1 in its own environment and `aggregate` in the book's, by `bench/pittarello2024_timing.py`. Each time is the minimum over seven repeats of construction and calculation together, as in the paper. The times are read from the files the script wrote, and are not re-measured when the book renders. The `aggregate` moments are recomputed here on the library's default grid. For the two contracts with aggregate terms, where the paper prints no reference, a computed reference applies the aggregate terms to a gross or XL distribution on 2^{22} buckets of 100.

```
AGGREGATE = json.loads((BOOK_ROOT / 'bench' / 'pittarello2024_timing_aggregate.json')
                       .read_text(encoding='utf-8'))
PUBLISHED = {
    'No reinsurance': [('gemact 1.2.1, published', 0.9581, 253625, 1.34406, 7.27745),
                       ('aggregate 0.9.3, published', 0.8102, 253625, 1.34406, 7.28346)],
    'XL': [('gemact 1.2.1, published', 0.9384, 256355, 1.10772, 2.08525),
           ('aggregate 0.9.3, published', 0.8445, 256354, 1.10773, 2.08527)],
    'SL': [('gemact 1.2.1, published', 0.9168, 194143, 1.24438, 1.73658),
           ('aggregate 0.9.3, published', 2.4254, 194143, 1.24438, 1.73658)],
    'XL with aggregate terms': [('gemact 1.2.1, published', 0.9388, 206363, 1.22464, 1.62335),
                                ('aggregate 0.9.3, published', 2.5221, 206363, 1.22465, 1.62336)],
}


def ceded_moments(a, limit=1_000_000, attach=50_000):
    """Mean, CV and skewness of the aggregate layer applied to a's distribution."""
    x, p = a.density_df.loss.values, a.density_df.p_total.values
    c = np.minimum(np.maximum(x - attach, 0), limit)
    mean = (c * p).sum()
    var = ((c - mean) ** 2 * p).sum()
    return mean, np.sqrt(var) / mean, ((c - mean) ** 3 * p).sum() / var ** 1.5


fine = {name: build(CONTRACTS[base], bs=100, log2=22)
        for name, base in [('SL', 'No reinsurance'), ('XL with aggregate terms', 'XL')]}
measured_gemact = {r['contract']: r for r in GEMACT['rows']}
measured_aggregate = {r['contract']: r for r in AGGREGATE['rows']
                      if r['configuration'] == 'library default'}
rows = []
for name, a in built.items():
```

```
    if name in fine:
        rows.append((name, 'computed reference', np.nan, *ceded_moments(fine[name])))
    else:
        rows.append((name, 'closed form reference', np.nan, *theory(a)))
    rows.extend((name, *published) for published in PUBLISHED[name])
    g = measured_gemact[name]
    rows.append((name, 'gemact 1.2.1, this machine', g['seconds'], g['mean'], g['cv'],
g['skewness']))
    rows.append((name, f'aggregate {AGGREGATE["versions"]["aggregate"]}, default grid, this
machine',
                 measured_aggregate[name]['seconds'], a.est_m, a.est_cv, a.est_skew))
table2 = pd.DataFrame(rows, columns=['contract', 'source', 'seconds', 'mean', 'CV', 'skewness']
                      ).set_index(['contract', 'source'])
print(f'measured {GEMACT["date"]} on {GEMACT["cpu"]}; '
      f'gemact under Python {GEMACT["versions"]["python"]}, '
      f'aggregate under Python {AGGREGATE["versions"]["python"]}')
GT(table2, formatters={'seconds': 3, 'mean': '{x:,.1f}', 'CV': 5, 'skewness': 5})
```

```
measured 2026-10-11 on AMD Ryzen 9 9950X3D2 16-Core Processor; gemact under Python 3.11.16,
aggregate under Python 3.14.7
```

Table 18.7.: Table 2 refreshed. Published rows from the paper, measured rows from this machine, and the library's default grid recomputed at render.

contract	*source*	seconds	mean	CV	skewness
No reinsurance	closed form reference	—	253,625.0	1.34406	7.28410
	gemact 1.2.1, published	0.958	253,625.0	1.34406	7.27745
	aggregate 0.9.3, published	0.810	253,625.0	1.34406	7.28346
	gemact 1.2.1, this machine	0.425	253,625.0	1.34406	7.27745
	aggregate 1.2.0a13, default grid, this machine	0.036	253,625.9	1.34407	7.27166
XL	closed form reference	—	256,355.2	1.10772	2.08525
	gemact 1.2.1, published	0.938	256,355.0	1.10772	2.08525
	aggregate 0.9.3, published	0.845	256,354.0	1.10773	2.08527
	gemact 1.2.1, this machine	0.409	256,355.2	1.10772	2.08525
	aggregate 1.2.0a13, default grid, this machine	0.052	256,354.8	1.10773	2.08525
SL	computed reference	—	194,143.4	1.24438	1.73658
	gemact 1.2.1, published	0.917	194,143.0	1.24438	1.73658
	aggregate 0.9.3, published	2.425	194,143.0	1.24438	1.73658
	gemact 1.2.1, this machine	0.440	194,143.4	1.24438	1.73658
	aggregate 1.2.0a13, default grid, this machine	0.043	194,150.6	1.24437	1.73643
XL with aggregate terms	computed reference	—	206,363.5	1.22464	1.62335
	gemact 1.2.1, published	0.939	206,363.0	1.22464	1.62335
	aggregate 0.9.3, published	2.522	206,363.0	1.22465	1.62336
	gemact 1.2.1, this machine	0.433	206,363.5	1.22464	1.62335
	aggregate 1.2.0a13, default grid, this machine	0.059	206,363.1	1.22464	1.62336

On this machine GEMAct takes about 0.42 seconds for each contract, and `aggregate` on its default grid takes 0.04 to 0.06 seconds, about eight to ten times faster, as Table 18.7 shows. On the paper's own grid of 2^{22} points the measured times keep the paper's pattern: `aggregate` takes 0.43 and 0.46

seconds on the first two contracts against GEMAct's 0.42 and 0.41, and 0.77 and 0.79 seconds on the two with aggregate terms against GEMAct's 0.44, so GEMAct remains the faster of the two at that size when there are aggregate terms.

The default grid sizes itself from the portfolio's moments, 2^{16} buckets of 4,000 for the gross portfolio and of 400 for the XL, and its means agree with the references to within 4×10^{-5}, the largest gap being the stop loss, whose 50,000 retention falls between buckets of 4,000. Its skewness for the gross portfolio is short by 0.2%, for the reason given under Table 18.6: the default grid is long enough for the mean and CV but not for the far tail that the third moment of this lognormal reaches. The computed references agree with GEMAct's 2^{22} point values to every printed digit. The library's 1.2.0a13 has a defect on grids as wide as the paper's, 2^{22} buckets of 500 reaching 2.1×10^9: the distribution is right, but the reported skewness is not, 11.9 for the gross portfolio against 7.28. The moments are taken from a copy of the density with every bucket below machine epsilon set to zero, which on this grid removes about 2×10^{-11} of genuine tail mass, and the lost mass is then placed at the right hand end of the grid as if the distribution were defective, where it is multiplied by $(2.1 \times 10^9)^3$.

18.3.6. Table 3 cannot be reproduced, and its reference values are misprinted

`aggregate` does not reproduce Table 3. Its copula support couples per-claim severities inside a bivariate aggregate with a shared count, and computes probabilities on a uniform grid. Table 3 needs the cdf of a sum of four or five Pareto variables, the heaviest with a tail index of 0.9, at points spread over six decades, to between 10^{-9} and 10^{-13}. A uniform grid cannot cover six decades of a tail that heavy, the probability of a half-plane cut along the anti-diagonal of a grid has an error of order h, and three or more dimensions are out of reach.

The table can still be checked against its source. Table 18.8 sets its printed reference values for four and five dimensions beside Arbenz et al. (2011)'s Tables 5 and 6, from which they are said to be copied.

```
aep = pd.DataFrame({
    's': ['10', '100', '1,000', '10,000'],
    'd = 4, printed': [0.983690398913354] * 4,
    'd = 4, Arbenz': [0.833447516734442, 0.983412214152579, 0.997950264030106, 0.999742266243751],
    'd = 5, printed': [0.983659549676444] * 4,
    'd = 5, Arbenz': [0.824132635126808, 0.983253494805448, 0.997930730055234, 0.999739803851201],
}).set_index('s')
GT(aep, formatters={c: '{x:.15f}' for c in aep.columns})
```

Table 18.8.: Table 3's printed reference values for four and five dimensions against Arbenz, Embrechts and Puccetti (2011), Tables 5 and 6.

s	d = 4, printed	d = 4, Arbenz	d = 5, printed	d = 5, Arbenz
10	0.983690398913354	0.833447516734442	0.983659549676444	0.824132635126808
100	0.983690398913354	0.983412214152579	0.983659549676444	0.983253494805448
1,000	0.983690398913354	0.997950264030106	0.983659549676444	0.997930730055234
10,000	0.983690398913354	0.999742266243751	0.983659549676444	0.999739803851201

The printed values in Table 18.8 are the same number in all four rows, and that number is the two and three dimensional value at $s = 100$. A distribution function cannot be flat from 10 to 10,000. The misprint is in the typeset reference values only: the errors printed beneath them are small, and against the misprinted values the error at $s = 10$ would be about 18%, so they were computed against Arbenz's correct values.

18.4. Comments

18.4.1. Choose the length of the grid before its spacing

Table 1 is a study of truncation presented as a study of discretization. Its errors are fixed by mh, and when h grows with m fixed the grid gets longer, more of the tail fits on it, and the error falls; that is the trend the paper reports. On a grid long enough to hold the distribution, Table 18.6 shows the opposite and expected trend, error falling as h^2. The lesson is to choose the length of the grid first and its spacing second. A default that sizes the grid from the distribution's moments does that without being asked, which is why the library's default grid puts every mean here within 2×10^{-4} of its reference. The same distinction runs through Grübel and Hermesmeier (1999), where mass past the end of the grid wraps around, and Embrechts et al. (1993), where the $O(h^2)$ rate appears once the support is adequate. Here the mass falls off the end instead.

18.4.2. Table 2, refreshed

The benchmark was a snapshot of `aggregate` 0.9.3, and its reading, that GEMAct is faster on contracts with aggregate terms at 2^{22} points, still holds on this machine, by a factor of about 1.8 rather than 2.7. The larger difference is the grid. 2^{22} points is sixty-four times the library's default grid for this portfolio, and on the grid the library chooses for itself `aggregate` is about ten times faster than either package at the paper's size, with means within 4×10^{-5} of the references. The comparison also exposed the skewness defect described above, which the library's own validation did not catch because it never builds a grid that wide unasked.

18.4.3. Table 3 and copulas

What `aggregate` offers for dependence is different in kind: a bivariate aggregate with a shared count and a copula on the severities, and Iman-Conover for dependence between lines after each is modeled exactly. Neither is a substitute for AEP on a sum of a few heavy tailed variables. Two smaller points about the reference data are worth recording. Arbenz's reference values are themselves AEP output at the largest iteration count, sixteen iterations for two dimensions and seven for four, so Table 3's two dimensional row at sixteen iterations, with errors near 10^{-13}, measures agreement between two implementations of the same algorithm rather than accuracy. The four dimensional row at seven iterations, which compares GEMAct with the reference computed at the same setting, shows an error of -4.55×10^{-4} at $s = 10$, so there the two implementations disagree.

18.4.4. Out of scope

The reserving module of Section 5 is outside the book's subject. Figure 1, which compares the four severity discretization methods, is covered by Embrechts and Frei (2009).

19. Embrechts and Frei's three discretization designs

Reproducing Embrechts and Frei (2009)

19.1. Context and problem

Embrechts and Frei (2009) is a head to head comparison: Panjer recursion against FFT for compound distributions. Both need the continuous severity replaced by a lattice distribution F_h of bandwidth h, and the paper's most useful section is about that step rather than about either transform.

There are three natural ways to discretize. **Forward** puts each cell's mass at its lower endpoint, **backward** at its upper endpoint, and **rounding** splits it so the cell's mean survives. The first two are the classical bounds: forward makes every claim smaller so it understates the aggregate, backward overstates it, and the true answer is between them. Rounding is not a bound at all, and that is the point of the paper's Table 1.

The paper also treats aliasing, recommending tilting where the interest is in the body of the distribution and noting that padding as in Wang (1998) does the same job. `aggregate` takes the padding branch, and its rounding default is the design Table 1 identifies as the accurate one, so this is a reproduction of the reasoning behind two library defaults.

19.2. Paper solution and gold standard

Table 1, Figure 19.1, estimates the 99.9% quantile of Pois(50) ∨ Exp(1) at four bandwidths under all three designs, with an exact column. Table 19.1 transcribes it.

Table 1 99.9%-quantile estimate for $Pois(50) \vee Exp(1)$ using different arithmetization designs

Design	$h = 1$	$h = 0.5$	$h = 0.1$	$h = 0.01$	Exact
Forward	58	70.0	81.9	84.78	
Rounding	84	84.5	85.1	85.11	85.11
Backward	124	103.0	88.4	85.43	

Figure 19.1.: Embrechts and Frei (2009), Table 1.

```
%xmode minimal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import poisson, gamma
from scipy.optimize import brentq
from aggregate.constants import FIG_W, FIG_H
from bookstyle import GT, build, show
```

```
PAPER = pd.DataFrame(
    {'h = 1': [58, 84, 124], 'h = 0.5': [70.0, 84.5, 103.0],
     'h = 0.1': [81.9, 85.1, 88.4], 'h = 0.01': [84.78, 85.11, 85.43]},
    index=['forward', 'rounding', 'backward'])
PAPER.index.name = 'design'
GT(PAPER, float_format=2)
```

```
Exception reporting mode: Minimal
```

Table 19.1.: Embrechts and Frei (2009), Table 1 as printed.

design	h = 1	h = 0.5	h = 0.1	h = 0.01
forward	58	70.00	81.90	84.78
rounding	84	84.50	85.10	85.11
backward	124	103.00	88.40	85.43

The exact value is printed only for the rounding row, as 85.11. It is checkable independently: a compound Poisson with exponential severity has

$$F_S(x) = e^{-\lambda} + \sum_{n \geq 1} \frac{e^{-\lambda} \lambda^n}{n!} \, \Gamma\text{-cdf}(x; n, 1),$$

a series that converges quickly at $\lambda = 50$, and Table 19.2 evaluates it.

```
LAM = 50

def exact_cdf(x, lam=LAM, nmax=400):
    n = np.arange(1, nmax + 1)
    return float(np.exp(-lam) + (poisson.pmf(n, lam) * gamma.cdf(x, n)).sum())

q_exact = brentq(lambda x: exact_cdf(x) - 0.999, 50, 200, xtol=1e-10)
GT(pd.DataFrame([
    {'quantity': '99.9% quantile from the closed-form series', 'value': q_exact},
    {'quantity': 'paper, Exact column', 'value': 85.11},
    {'quantity': 'E[S] exact', 'value': float(LAM)},
]).set_index('quantity'), float_format=6)
```

Table 19.2.: The 99.9% quantile from the closed form series, against the paper's exact column.

quantity	value
99.9% quantile from the closed-form series	85.105955
paper, Exact column	85.110000
E[S] exact	50.000000

Table 19.2 gives 85.10596, which is 85.11 to the two decimals printed. The gold standard is sound before anything is convolved.

19.3. The `aggregate` reproduction

`sev_calc` selects the design, so all three of the paper's rows are one keyword apart. The grid is sized to reach 400, comfortably past the quantile of interest, at every bandwidth.

```
def ef(h, design):
    log2 = int(np.ceil(np.log2(400 / h)))
    return build('agg EF 50 claims sev expon poisson', bs=h, log2=log2,
                 padding=1, sev_calc=design, normalize=False)

show(ef(0.01, 'round'))
rows = []
for design, label in [('forward', 'forward'), ('round', 'rounding'),
                      ('backward', 'backward')]:
    rows.append({'design': label,
                 **{f'h = {h:g}': ef(h, design).q(0.999)
                    for h in (1.0, 0.5, 0.1, 0.01)}})
got = pd.DataFrame(rows).set_index('design')
GT(pd.concat({'paper': PAPER, 'aggregate': got}, axis=1), float_format=2)
```

Table 19.3.: Table 1 rebuilt, the 99.9% quantile under all three designs at four bandwidths, beside the printed figures.

```
agg EF
  50 claims
  sev expon
  poisson
```

	paper				aggregate			
design	**h = 1**	**h = 0.5**	**h = 0.1**	**h = 0.01**	**h = 1**	**h = 0.5**	**h = 0.1**	**h = 0.01**
forward	58	70.00	81.90	84.78	58	70.00	81.90	84.78
rounding	84	84.50	85.10	85.11	84	84.50	85.10	85.11
backward	124	103.00	88.40	85.43	124	103.00	88.40	85.43

```
GT(pd.DataFrame([
    {'design': d,
     'worst absolute difference': float((got.loc[d] - PAPER.loc[d]).abs().max())}
    for d in PAPER.index]
    + [{'design': 'all twelve',
        'worst absolute difference': float((got - PAPER).abs().max().max())}]
).set_index('design'), formatters={'worst absolute difference': '{x:.3f}'})
```

Table 19.4.: Worst absolute difference from Table 1, design by design.

design	**worst absolute difference**
forward	0.000
rounding	0.000
backward	0.000
all twelve	0.000

In Table 19.3 and Table 19.4 all twelve values reproduce **exactly**, to every digit printed, across three designs and four bandwidths. That is the strongest form this comparison can take: the paper's table is not being approximated, it is being recovered.

19.3.1. What the table is actually saying

The numbers matter less than their arrangement, and the arrangement is easy to miss in a small table. Table 19.5 sets it out.

```
rows = []
for h in (1.0, 0.5, 0.1, 0.01):
    f, r, b = (got.loc[d, f'h = {h:g}'] for d in ('forward', 'rounding', 'backward'))
    rows.append({'h': h, 'bracket width, backward minus forward': b - f,
                 'rounding error': r - q_exact,
                 'forward error': f - q_exact, 'backward error': b - q_exact})
GT(pd.DataFrame(rows).set_index('h'), float_format=3)
```

Table 19.5.: The width of the forward to backward bracket at each bandwidth, beside the error of each design.

h	bracket width, backward minus forward	rounding error	forward error	backward error
1.000	66.000	-1.11e+00	-2.71e+01	3.89e+01
0.500	33.000	-6.06e-01	-1.51e+01	1.79e+01
0.100	6.500	-5.96e-03	-3.21e+00	3.29e+00
0.010	0.650	4.04e-03	-3.26e-01	3.24e-01

In Table 19.5, at $h = 1$ the bounds are 58 and 124, a bracket 66 wide around a true value of 85.1, which is useless for any practical purpose. Rounding at the same bandwidth gives 84, an error of 1.1. The bounds do bracket, honestly and always, and they do it so loosely that at any bandwidth a practitioner would actually use, the bracket is worth less than the point estimate it contains.

That is why `aggregate` has a rounding default and no bounds mode. It is the same conclusion Gerber (1982) reached from the other end, in 1982, with the same ranking of the same three methods.

The cost of the default is that it is not conservative in any direction. Rounding is a *better* estimate and a *worse* guarantee, and the honest error control is refinement: rounding at $h = 1, 0.5, 0.1$ gives 84, 84.5, 85.1, which converges visibly on its own without needing the bounds to confirm it.

19.4. Figure 1: the Poisson/Pareto aliasing picture

Section 3 illustrates aliasing on Pois(20) ∨ Pareto(4, 3), a severity with finite mean and infinite third moment. There is no published table for Figure 19.2, so the reproduction is of the shape and the regime change rather than of numbers.

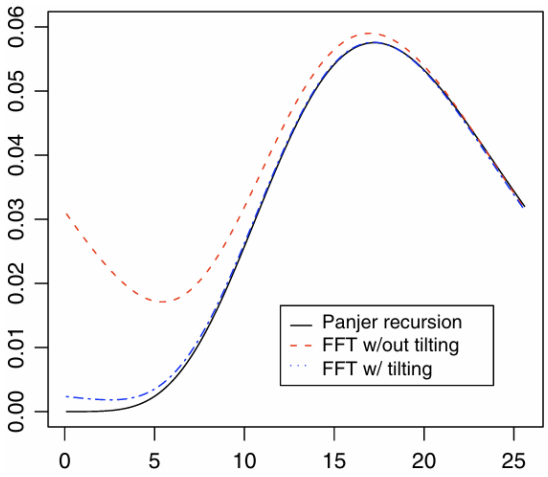


Figure 19.2.: Approximate densities for $Pois(20) \vee Pareto(4, 3)$, Embrechts and Frei (2009), Figure 1.

```
ALPHA, BETA, FREQ = 4, 3, 20
SEV = f'sev {BETA} * pareto {ALPHA} - {BETA}'
a = build(f'agg EF1 {FREQ} claims {SEV} poisson',
          bs=1 / 8, log2=8, padding=0, normalize=False)
df = a.density_df[['p_total']].rename(columns={'p_total': 'padding 0'})
for pad in (1, 2):
    a.update(bs=1 / 8, log2=8, padding=pad, normalize=False)
    df[f'padding {pad}'] = a.density_df.p_total
a.update(bs=1 / 32, log2=16, padding=1, normalize=False)
fine = a.density_df[['p_total']].rename(columns={'p_total': 'log2 16, bs 1/32'})

fig, axs = plt.subplots(1, 2, figsize=(2 * FIG_W, FIG_H), layout='constrained')
df.plot(ax=axs[0], lw=1)
axs[0].set(xlabel='loss', ylabel='pmf', title='$2^8$ buckets, bs = 1/8')
df.plot(ax=axs[1], logy=True, lw=1)
axs[1].set(xlabel='loss', title='the same, log scale')
for ax in axs:
    ax.legend(fontsize=6)
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning:
EF1: severity 'pareto' cannot be reproduced on this grid. The body needs bs <= 0.0174043 (0.2431%
of the mean is supplied below the first half bucket, and is placed at 0); the mean is not complete
until the grid reaches 125.507, so matching it to 0.0001 needs log2 = 13, and log2 = 8 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)
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
EF1: aggregate PMF deficit 1.087e-03 (Σp = 1 − 1.087e-03 < 1); forwards and backwards S diverge by
the deficit (forwards > backwards). Further occurrences this session are suppressed; each object
records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

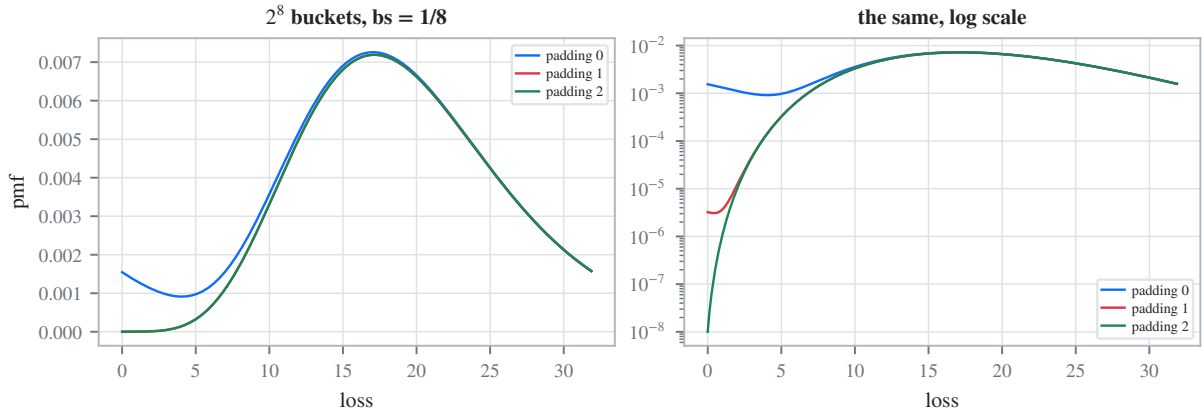


Figure 19.3.: Reproducing Figure 1. The aggregate density at three padding levels, on a linear and a log scale.

In Figure 19.3 the untilted, unpadded curve turns up at the right hand end, which is aliased mass that belongs beyond the grid, and the log panel shows padding removing it completely. The paper's point that padding and tilting are equally effective is the same finding Grübel and Hermesmeier (1999) arrives at from the tilting side and Schaller and Temnov (2008) from the operational risk side, and this page's Table 1 result is why the library only needs one of them.

```
fig, ax = plt.subplots(1, 1, figsize=(FIG_W, FIG_H), layout='constrained')
x = fine.index.to_numpy()[1:]
ax.loglog(x, fine.iloc[1:, 0] / a.bs, lw=1, label='aggregate density')
ax.loglog(x, FREQ * ALPHA / BETA * (BETA / (BETA + x)) ** (ALPHA + 1), lw=1,
          ls='--', label="Feller, $20 f_X(x)$")
ax.set(xlabel='loss', ylabel='density')
ax.legend(fontsize=6)
```

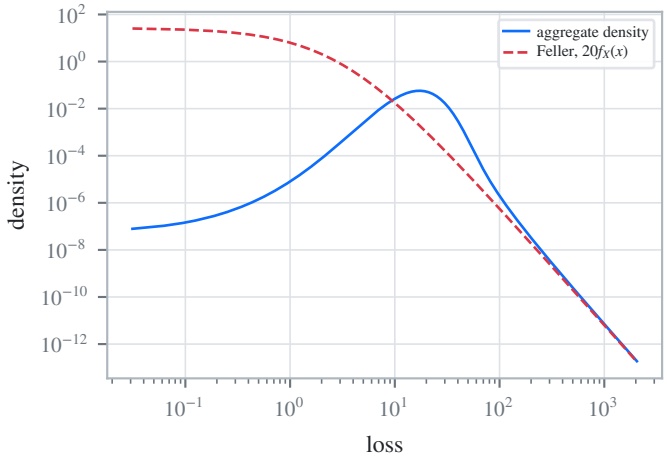


Figure 19.4.: The far tail against Feller's asymptotic, twenty times the severity survival function.

Differentiating Feller's theorem gives the aggregate density approaching $\lambda f_X(x)$ in the far tail, and in Figure 19.4 the two curves meet once the Poisson body is left behind. The change of regime, from a body dominated by the claim count to a tail dominated by the single largest claim, is visible as the kink.

19.5. Comments

19.5.1. The three designs are still there, and one is missing on purpose

`aggregate` implements `forward`, `backward` and `round`, so all three of Table 1's rows are reachable, and the `moment` design that would preserve two moments raises `NotImplementedError`. The paper's table is the argument for that gap: since rounding already beats both bounds by an order of magnitude at every bandwidth, a fourth design has to justify itself against rounding rather than against the bounds, and none of the candidates does.

19.5.2. Where the bounds do earn their keep

The bracket is useless as an error bar and valuable as a *test*. If a build's forward and backward estimates do not straddle its rounding estimate, something is wrong with the grid rather than with the discretization: the most likely cause is that aliasing or truncation is larger than the discretization error, so the three designs are all being dominated by an error that has nothing to do with the design. Table 19.6 runs that test on an adequate grid and on a short one.

```
rows = []
for h, pad in [(0.5, 1), (0.5, 0)]:
    vals = {}
    for design, label in [('forward', 'forward'), ('round', 'rounding'),
                          ('backward', 'backward')]:
        log2 = 6 if pad == 0 else int(np.ceil(np.log2(400 / h)))
        b = build('agg EF 50 claims sev expon poisson', bs=h, log2=log2,
                  padding=pad, sev_calc=design, normalize=False)
        vals[label] = b.q(0.999)
    rows.append({'h': h, 'log2': log2, 'padding': pad, **vals,
                 'brackets': bool(vals['forward'] <= vals['rounding']
                                  <= vals['backward'])})
GT(pd.DataFrame(rows).set_index(['h', 'log2', 'padding']), float_format=2)
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning: EF:
severity 'expon' cannot be reproduced on this grid. The body needs bs <= 0.020067 (2.65% of the
mean is supplied below the first half bucket, and is placed at 0); the mean is not complete until
the grid reaches 12.5067, so matching it to 0.0001 needs log2 = 9, and log2 = 6 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)
```

Table 19.6.: The three designs on an adequate grid and on one too short, showing where the bracket ordering fails.

h	*log2*	*padding*	forward	rounding	backward	brackets
0.50	10	1	70.00	84.50	103.00	True
	6	0	31.50	31.50	31.50	True

In Table 19.6 the ordering holds on an adequate grid. On a grid too short to hold the distribution it fails, and the failure is informative in a way the individual numbers are not: it says the bandwidth is no longer the binding constraint. That is worth more than a bound, and it costs two extra builds.

20. Schaller and Temnov on a severity with no mean

Reproducing Schaller and Temnov (2008)

20.1. Context and problem

Operational risk models push FFT convolution harder than insurance pricing does. The frequencies are moderate but the severities are extreme: a generalized Pareto with shape $\xi = 1$ has **no mean**, let alone a variance, and the regulatory question is a high quantile of the aggregate. Schaller and Temnov (2008) is written from inside that problem, and its subject is how to get a trustworthy quantile out of an FFT when the severity has no moments to anchor the grid.

The paper compares three arrangements of the same transform. A **naive** FFT convolves on the grid as given, so aggregate mass that should land past the end wraps around into the low buckets. **Padding by zeros** extends the vector before transforming, so there is nowhere for the wrap to land. An **exponential window** multiplies the input by $e^{-x/\tau}$ before transforming and divides it out afterwards, which is the exponential tilting of Grübel and Hermesmeier (1999) under another name, with the paper supplying a rule for the optimal τ.

The paper's argument is that the exponential window matches the accuracy of generous padding at a fraction of the cost, and it makes that case with timings as well as accuracy.

20.2. Paper solution and gold standard

Section 4.3.2 is the real data example. The severity is a generalized Pareto in the form

$$G_{\xi,\mu,\beta}(x) = 1 - \left(1 + \xi \frac{x - \mu}{\beta}\right)^{-1/\xi}$$

with location $\mu = 7000$, scale $\beta = 12000$ and shape $\xi = 1.0$. Frequency is Poisson with $\lambda = 18$. The severity is truncated at $x_m = 5 \times 10^6$ **and not normalized**, which matters and is stated. The optimal window parameter is $\tau_{\text{opt}} = 149{,}688.5$.

Table 4.3, Figure 20.1, then gives the 0.9-quantile of the aggregate at four grid sizes under all four arrangements. Its caption carries the detail that makes it reproducible: "The grid size indicated in the table is the one **before** the padding by zeros." Table 20.1 transcribes it.

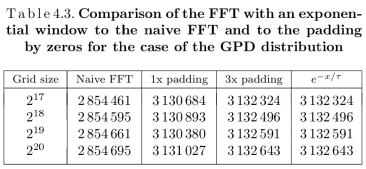


Table 4.3. **Comparison of the FFT with an exponential window to the naive FFT and to the padding by zeros for the case of the GPD distribution**

Grid size	Naive FFT	1x padding	3x padding	$e^{-x/\tau}$
2^{17}	2 854 461	3 130 684	3 132 324	3 132 324
2^{18}	2 854 595	3 130 893	3 132 496	3 132 496
2^{19}	2 854 661	3 130 380	3 132 591	3 132 591
2^{20}	2 854 695	3 131 027	3 132 643	3 132 643

Figure 20.1.: Schaller and Temnov (2008), Table 4.3.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build, show

XM, LAM = 5e6, 18
PAPER = pd.DataFrame(
    {'naive': [2_854_461, 2_854_595, 2_854_661, 2_854_695],
     '1x padding': [3_130_684, 3_130_893, 3_130_380, 3_131_027],
     '3x padding': [3_132_324, 3_132_496, 3_132_591, 3_132_643],
     'exponential window': [3_132_324, 3_132_496, 3_132_591, 3_132_643]},
    index=[17, 18, 19, 20])
PAPER.index.name = 'log2'
GT(PAPER, float_format='{x:,.0f}')
```

```
Exception reporting mode: Minimal
```

Table 20.1.: Schaller and Temnov (2008), Table 4.3 as printed.

log2	naive	1x padding	3x padding	exponential window
17	2,854,461	3,130,684	3,132,324	3,132,324
18	2,854,595	3,130,893	3,132,496	3,132,496
19	2,854,661	3,130,380	3,132,591	3,132,591
20	2,854,695	3,131,027	3,132,643	3,132,643

Two features of Table 20.1 are worth naming before reproducing it. The naive column is wrong by 8.9%, which is the aliasing error and is far too large to be mistaken for anything subtle. And the exponential window column is **identical to the 3x padding column at every grid size, to all seven digits**. The paper's own evidence is that the two remedies are interchangeable in accuracy, and its case for the window is therefore entirely about cost.

20.3. The `aggregate` reproduction

The severity is `12000 * genpareto 1 + 7000`: scale, shape, location, in the order DecL takes them. `aggregate`'s padding counts doublings, so Schaller's "1x padding" is `padding=1` and his "3x padding", a vector four times as long, is `padding=2`. The library has no exponential window on the production path, so that column has no counterpart here, which is the right outcome given the two are identical in the paper.

The grid is Schaller's: the severity's right endpoint is $x_m = 5 \times 10^6$, so `bs` follows from the grid size rather than being chosen, for the reasons Table 20.2 records.

```
a = build(f'agg SchallerTemnov {LAM} claims '
          f'sev 12000 * genpareto 1 + 7000 poisson '
          f'hints{{log2=20; bs={XM / (1 << 20)}; padding=2; normalize=False}}')
show(a)
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
SchallerTemnov: aggregate PMF deficit 5.568e-02 (Σp = 1 − 5.568e-02 < 1); forwards and backwards
S diverge by the deficit (forwards > backwards). Further occurrences this session are suppressed;
each object records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

```
agg SchallerTemnov
  18 claims
  sev 12000 * genpareto 1 + 7000
  poisson
  hints{log2=20; bs=4.76837158203125; padding=2; normalize=False}
```

```
GT(pd.DataFrame([
    {'quantity': 'severity shape xi', 'value': 1.0,
     'note': 'xi = 1 means no finite mean'},
    {'quantity': 'severity truncation x_m', 'value': XM,
     'note': 'stated in the paper, and not normalized'},
    {'quantity': 'aggregate mass on the grid', 'value': a.agg_density.sum(),
     'note': 'below 1 by the truncated tail, as intended'},
]).set_index('quantity'), formatters={'value': '{x:,.6g}'})
```

Table 20.2.: The three features of the severity that make the grid an external choice rather than a computed one.

quantity	value	note
severity shape xi	1	xi = 1 means no finite mean
severity truncation x_m	5e+06	stated in the paper, and not normalized
aggregate mass on the grid	0.944317	below 1 by the truncated tail, as intended

```
rows = []
for log2 in (17, 18, 19, 20):
    bs = XM / (1 << log2)
    r = {'log2': log2, 'bs': bs}
    for name, pad in [('naive', 0), ('1x padding', 1), ('3x padding', 2)]:
        a.update(log2=log2, bs=bs, padding=pad, normalize=False)
        r[name] = a.q(0.9)
    rows.append(r)
got = pd.DataFrame(rows).set_index('log2')
GT(got, formatters={'bs': 3, 'naive': '{x:,.0f}', '1x padding': '{x:,.0f}',
                    '3x padding': '{x:,.0f}'})
```

Table 20.3.: Table 4.3 rebuilt, the 0.9-quantile at four grid sizes under the naive and two padded arrangements.

log2	bs	naive	1x padding	3x padding
17	38.147	2,854,729	3,131,065	3,132,706
18	19.073	2,854,729	3,131,065	3,132,687
19	9.537	2,854,729	3,131,075	3,132,687
20	4.768	2,854,729	3,131,075	3,132,687

```
COLS = ['naive', '1x padding', '3x padding']
rel = (got[COLS] / PAPER[COLS] - 1).abs()
GT(pd.DataFrame([
    {'column': c, 'values compared': 4,
     'worst relative difference': float(rel[c].max()),
     'worst absolute difference': float((got[c] - PAPER[c]).abs().max())}
    for c in COLS] + [
    {'column': 'all twelve', 'values compared': 12,
     'worst relative difference': float(rel.max().max()),
     'worst absolute difference': float((got[COLS] - PAPER[COLS]).abs().max().max())}
]).set_index('column'),
   formatters={'values compared': 0, 'worst relative difference': '{x:.2e}',
               'worst absolute difference': '{x:,.0f}'})
```

Table 20.4.: Worst difference from Table 4.3, column by column and over all twelve values.

column	values compared	worst relative difference	worst absolute difference
naive	4	9.38e-05	268
1x padding	4	2.22e-04	695
3x padding	4	1.22e-04	382
all twelve	12	2.22e-04	695

By Table 20.3 and Table 20.4 all twelve distinct published values reproduce to within 2.2×10^{-4} relative, which on a quantile of 3.13 million is agreement to about 700. The aliasing error the paper is warning about, 8.9% of the answer, comes back at the same size; the two remedies both remove it; and the converged value is Schaller's 3,132,643 to five significant figures.

20.3.1. Where the two calculations disagree, and it is instructive

The agreement is on the values. The *pattern* differs, and Table 20.5 shows it rather than averaging it away.

```
cmp = pd.concat({'paper': PAPER[COLS], 'aggregate': got[COLS]}, axis=1)
GT(cmp, float_format='{x:,.0f}')
```

Table 20.5.: The published and rebuilt columns side by side, showing the two implementations converging differently.

	paper			aggregate		
log2	naive	1x padding	3x padding	naive	1x padding	3x padding
17	2,854,461	3,130,684	3,132,324	2,854,729	3,131,065	3,132,706
18	2,854,595	3,130,893	3,132,496	2,854,729	3,131,065	3,132,687
19	2,854,661	3,130,380	3,132,591	2,854,729	3,131,075	3,132,687
20	2,854,695	3,131,027	3,132,643	2,854,729	3,131,075	3,132,687

In Table 20.5 Schaller's columns rise monotonically with grid size: his 3x padding runs 3,132,324, 3,132,496, 3,132,591, 3,132,643, still climbing at 2^{20}. The `aggregate` column is flat from 2^{18} onward at 3,132,687. Both land in the same place, but his estimate approaches it from below while this one is already there.

The reason is discretization, not aliasing. Refining the grid here refines `bs` as well, since x_m is fixed, so the sweep is a discretization sweep and the two implementations are converging at different rates from different discretization schemes. `aggregate`'s default preserves the cell mean, which is the moment-matching idea Gerber (1982) recommends; a scheme that assigns cell mass to an endpoint converges from one side, which is the shape of Schaller's column. The residual 44 between 3,132,687 and 3,132,643 is about one bucket at 2^{20}, where `bs` is 4.77, so it is inside the quantile's own quantization.

20.4. Comments

20.4.1. Normalization is not a cosmetic choice here

The paper says the severity is truncated and not normalized, and with $\xi = 1$ that instruction is load bearing. Normalizing divides the truncated severity by its own mass, which redistributes the discarded tail proportionally across the retained buckets. For a thin tail that is a rounding correction. For a severity with no mean it is a change of model, and it thins the tail, as Table 20.6 measures.

```
rows = []
for log2, bs in [(17, XM / (1 << 17)), (20, XM / (1 << 20))]:
    for norm in (False, True):
        a.update(log2=log2, bs=bs, padding=2, normalize=norm)
        rows.append({'log2': log2, 'normalize': norm, '0.9 quantile': a.q(0.9),
                     '0.99 quantile': a.q(0.99),
                     'mass on grid': a.agg_density.sum()})
GT(pd.DataFrame(rows).set_index(['log2', 'normalize']),
   formatters={'0.9 quantile': '{x:,.0f}', '0.99 quantile': '{x:,.0f}',
               'mass on grid': 8})
```

Table 20.6.: Two quantiles with and without normalization, at a coarse and a fine grid.

log2	*normalize*	0.9 quantile	0.99 quantile	mass on grid
17	False	3,132,706	4,999,962	0.94431685
	True	2,456,551	4,999,962	0.98588811
20	False	3,132,687	4,999,995	0.94431707
	True	2,456,560	4,999,995	0.98588820

In Table 20.6, normalizing moves the 0.9-quantile by a visible amount and the 0.99-quantile by more, always downward, because the mass removed from beyond the truncation point is being handed back to buckets below it. Anyone reproducing this table with `normalize=True`, which is `aggregate`'s default, will miss every entry and will miss it in a consistent direction. The general rule the library documents is the one this example forces: when the truncation error is unavoidable, do not normalize, and read the tail knowing it is understated.

20.4.2. The timing argument has not survived, and the accuracy argument never needed to

The paper's case for the exponential window is cost: it reports the window giving the same precision as 2x padding in a third of the time, and as 3x padding at least four times faster. That was a real argument in 2008.

It is no longer the deciding one, because the whole sweep above, twelve builds up to 2^{20} buckets with quadruple padding, runs in seconds. When the expensive option is cheap in absolute terms, a method that buys speed at the cost of a tuning parameter stops being attractive. And the window does carry one: $\tau_{\text{opt}} = 149{,}688.5$ is a computed quantity specific to this model, and getting it wrong costs accuracy in one direction or floating point range in the other, which is the failure mode Grübel and Hermesmeier (1999) hits at $\theta N = 25$.

What Table 20.1 actually establishes, and states more strongly than the paper's own argument needs, is that the window and generous padding are **the same answer to seven digits** at every grid size tested. Given that, the choice between them is entirely about which is easier to get right, and padding has no parameter to get wrong.

20.4.3. A severity with no mean is a good test of the grid sizer, and this one bypasses it

`aggregate` will refuse to pick a bucket size for an infinite-variance severity rather than guess, and $\xi = 1$ has no mean at all. That refusal is why every build on this page carries an explicit `bs` in its `hints{}`, and why the sweep is expressed as $x_m / 2^{\log 2}$: the paper fixes the truncation point, so the grid size determines `bs` rather than the other way round.

That is the right way round for this problem and the wrong way round for most. Ordinarily `bs` is chosen from the severity's own scale and the grid is made long enough to hold the aggregate. Here there is no scale to work from, so an external decision, the regulatory truncation point, supplies it.

21. Bruno et al. on thresholded direct convolution

Reproducing Bruno, Camerini, Manna and Tomassetti (2006)

21.1. Context and problem

Bruno et al. (2006) proposes a direct method for the aggregate claims distribution, competing with Panjer recursion, De Pril transforms and the FFT. The idea is thresholded direct convolution: at each convolution order, keep only the outcomes whose probability exceeds a cutoff chosen so that the retained mass and the first three moments are correct to a prescribed tolerance, here 10^{-6}, and discard the rest.

The claimed advantages are that no assumption is needed on the counting distribution, that severities need not be identically distributed, and that the method avoids the underflow that kills Panjer recursion at large λ and the overflow and bit-reversal overhead of the FFT. Tables 3 to 5 count multiplications, Table 4 reports timings, and the authors conclude their method is "more flexible, easier and cheaper than the (exact and approximated) methods using recursion and FFT", with the advantage growing as the number of convolutions rises.

This is a method paper of the most useful kind for a reproduction: it gives the full input specification and tabulates the output distribution to six decimal places.

21.2. Paper solution and gold standard

Severity is a 14-point discrete law on the integers 14 to 60, Figure 21.1, the same for every claim. The three counting distributions are set out in Table 21.1.

Table 21.1.: The three published tables, their counting distributions and what each tabulates.

Table	**Counting distribution**	**Content**
6	Poisson, $\lambda = 504.814259$	probability and CDF at 28 quantiles
7	Poisson, $\lambda = 91{,}000$	CDF at 28 quantiles, 92,832 convolutions
8	Poisson mixed by a generalized Pareto	CDF at 17 quantiles

The quantiles are quoted as $\mu + k\sigma$ for stated μ and σ and then truncated to integers, so the comparison is unambiguous: evaluate at the integers printed. Table 21.2 checks the transcribed severity first.

Table 1
Severity distribution

Realizations	Probabilities
14	0.0103301
15	0.0307990
16	0.0293511
17	0.0103301
18	0.0730414
19	0.0111568
20	0.0264554
24	0.1002133
26	0.0815418
28	0.0252146
30	0.0212857
31	0.0254214
55	0.0991756
60	0.4556837
Otherwise	0

Figure 21.1.: Bruno et al (2006), Table 1.

```
%xmode minimal
import numpy as np
import pandas as pd
from scipy.integrate import quad
from scipy.special import gammaln
from bookstyle import GT, build

XS = [14, 15, 16, 17, 18, 19, 20, 24, 26, 28, 30, 31, 55, 60]
PS = [.0103301, .0307990, .0293511, .0103301, .0730414, .0111568, .0264554,
      .1002133, .0815418, .0252146, .0212857, .0254214, .0991756, .4556837]
DSEV = ('dsev [' + ' '.join(map(str, XS)) + '] ['
        + ' '.join(f'{q:.7f}' for q in PS) + ']')
x, p = np.array(XS, float), np.array(PS)
m1, m2 = (x * p).sum(), (x ** 2 * p).sum()
GT(pd.DataFrame([{'sum of probabilities': p.sum(), 'severity mean': m1,
                  'E[U^2]': m2, 'severity sd': np.sqrt(m2 - m1 ** 2)}]),
   formatters={'sum of probabilities': 10, 'severity mean': 6, 'E[U^2]': 4,
               'severity sd': 6})
```

```
Exception reporting mode: Minimal
```

Table 21.2.: The transcribed 14 point severity, its total mass and its first two moments.

sum of probabilities	severity mean	E[U^2]	severity sd
1.0000000000	42.761197	2,174.3969	18.597766

21.3. Table 6: Poisson, $\lambda = 504.814259$

The severity is integer valued, so the aggregate lives on the integer lattice and `bs = 1` is exact. The support runs to $658 \times 60 = 39{,}480$, comfortably inside 2^{17}. The target is Figure 21.2.

Table 6
Aggregate claims distribution and cumulative distribution using a Poisson counting distribution with parameter $\lambda = 504.814259$

Aggregate claims amount[a]	Probabilities	Cumulative distribution values
$\mu - 5.00\sigma \cong 16{,}347$	0.000000	0.000000
$\mu - 4.00\sigma \cong 17{,}395$	0.000000	0.000017
$\mu - 3.00\sigma \cong 18{,}443$	0.000004	0.001051
$\mu - 2.50\sigma \cong 18{,}967$	0.000015	0.005405
$\mu - 2.00\sigma \cong 19{,}491$	0.000051	0.021317
$\mu - 1.90\sigma \cong 19{,}595$	0.000062	0.027153
$\mu - 1.80\sigma \cong 19{,}700$	0.000075	0.034326
$\mu - 1.75\sigma \cong 19{,}752$	0.000082	0.038411
$\mu - 1.70\sigma \cong 19{,}805$	0.000090	0.042970
$\mu - 1.65\sigma \cong 19{,}857$	0.000098	0.047852
$\mu - 1.60\sigma \cong 19{,}910$	0.000107	0.053271
$\mu - 1.50\sigma \cong 20{,}014$	0.000125	0.065289
$\mu - 1.00\sigma \cong 20{,}538$	0.000235	0.158591
$\mu - 0.75\sigma \cong 20{,}800$	0.000292	0.227754
$\mu - 0.50\sigma \cong 21{,}062$	0.000340	0.310857
$\mu - 0.25\sigma \cong 21{,}324$	0.000371	0.404499
$\mu - 0.00\sigma \cong 21{,}586$	0.000381	0.503536
$\mu + 0.25\sigma \cong 21{,}848$	0.000367	0.601928
$\mu + 0.50\sigma \cong 22{,}110$	0.000332	0.693829
$\mu + 0.75\sigma \cong 22{,}372$	0.000283	0.774593
$\mu + 1.00\sigma \cong 22{,}634$	0.000227	0.841425
$\mu + 1.50\sigma \cong 23{,}158$	0.000122	0.931859
$\mu + 2.00\sigma \cong 23{,}681$	0.000052	0.975826
$\mu + 2.50\sigma \cong 24{,}205$	0.000018	0.992972
$\mu + 3.00\sigma \cong 24{,}729$	0.000005	0.998325
$\mu + 4.00\sigma \cong 25{,}777$	0.000000	0.999947
$\mu + 5.00\sigma \cong 26{,}824$	0.000000	0.999999
$\mu + 8.00\sigma \cong 29{,}968$	0.000000	1.000000

[a] $\mu = 21586.462129$ and $\sigma = 1047.695834$. They are, respectively, the mean value and the standard deviation of the aggregate claims distribution.

Figure 21.2.: Bruno et al (2006), Table 6.

```
LAM6 = 504.814259
a6 = build(f'agg B {LAM6} claims {DSEV} poisson', log2=17, bs=1,
           padding=1, normalize=False)
GT(pd.DataFrame([
    {'quantity': 'mean', 'aggregate': a6.est_m, 'lambda E[U]': LAM6 * m1,
     'paper': 21586.462129},
    {'quantity': 'standard deviation', 'aggregate': a6.est_sd,
     'lambda E[U]': np.sqrt(LAM6 * m2), 'paper': 1047.695834},
    {'quantity': 'total mass', 'aggregate': a6.agg_density.sum(),
     'lambda E[U]': 1.0, 'paper': 1.0},
]).set_index('quantity'), float_format=6)
```

Table 21.3.: The Table 6 aggregate's mean, standard deviation and total mass, against the compound Poisson formulas and the printed values.

quantity	aggregate	lambda E[U]	paper
mean	21,586.462129	21,586.462129	21,586.462129
standard deviation	1,047.695834	1,047.695834	1,047.695834
total mass	1.000000	1.000000	1.000000

```
T6 = [(16347, .000000, .000000), (17395, .000000, .000017),
      (18443, .000004, .001051), (18967, .000015, .005405),
      (19491, .000051, .021317), (19595, .000062, .027153),
      (19700, .000075, .034326), (19752, .000082, .038411),
      (19805, .000090, .042970), (19857, .000098, .047852),
      (19910, .000107, .053271), (20014, .000125, .065289),
      (20538, .000235, .158591), (20800, .000292, .227754),
      (21062, .000340, .310857), (21324, .000371, .404499),
      (21586, .000381, .503536), (21848, .000367, .601928),
      (22110, .000332, .693829), (22372, .000283, .774593),
      (22634, .000227, .841425), (23158, .000122, .931859),
      (23681, .000052, .975826), (24205, .000018, .992972),
      (24729, .000005, .998325), (25777, .000000, .999947),
```

```
      (26824, .000000, .999999), (29968, .000000, 1.000000)]
d6 = a6.density_df
t6 = pd.DataFrame([{'x': xx, 'agg probability': d6.loc[xx, 'p_total'],
                    'Table 6 probability': pp, 'agg CDF': d6.loc[xx, 'F'],
                    'Table 6 CDF': cc} for xx, pp, cc in T6])
t6['CDF diff'] = (t6['agg CDF'] - t6['Table 6 CDF']).abs()
GT(t6.set_index('x'),
   formatters={'agg probability': 6, 'Table 6 probability': 6, 'agg CDF': 6,
               'Table 6 CDF': 6, 'CDF diff': '{x:.2e}'})
```

Table 21.4.: All 28 rows of Table 6, probability and CDF, against the rebuilt values.

x	agg probability	Table 6 probability	agg CDF	Table 6 CDF	CDF diff
16,347	0.000000	0.000000	0.000000	0.000000	8.07e-08
17,395	0.000000	0.000000	0.000017	0.000017	4.68e-08
18,443	0.000004	0.000004	0.001051	0.001051	2.56e-07
18,967	0.000015	0.000015	0.005405	0.005405	1.86e-07
19,491	0.000051	0.000051	0.021317	0.021317	1.31e-07
19,595	0.000062	0.000062	0.027153	0.027153	1.15e-07
19,700	0.000075	0.000075	0.034326	0.034326	4.11e-08
19,752	0.000082	0.000082	0.038411	0.038411	1.71e-07
19,805	0.000090	0.000090	0.042970	0.042970	4.49e-07
19,857	0.000098	0.000098	0.047852	0.047852	3.09e-07
19,910	0.000107	0.000107	0.053271	0.053271	2.19e-07
20,014	0.000125	0.000125	0.065289	0.065289	2.51e-07
20,538	0.000235	0.000235	0.158591	0.158591	4.03e-07
20,800	0.000292	0.000292	0.227754	0.227754	3.37e-07
21,062	0.000340	0.000340	0.310857	0.310857	2.39e-07
21,324	0.000371	0.000371	0.404499	0.404499	2.45e-08
21,586	0.000381	0.000381	0.503536	0.503536	2.36e-07
21,848	0.000367	0.000367	0.601928	0.601928	3.08e-07
22,110	0.000332	0.000332	0.693829	0.693829	3.62e-07
22,372	0.000283	0.000283	0.774593	0.774593	5.32e-08
22,634	0.000227	0.000227	0.841425	0.841425	2.46e-07
23,158	0.000122	0.000122	0.931859	0.931859	4.14e-07
23,681	0.000052	0.000052	0.975826	0.975826	1.41e-07
24,205	0.000018	0.000018	0.992972	0.992972	9.39e-08
24,729	0.000005	0.000005	0.998325	0.998325	3.66e-07
25,777	0.000000	0.000000	0.999947	0.999947	3.70e-07
26,824	0.000000	0.000000	0.999999	0.999999	2.37e-07
29,968	0.000000	0.000000	1.000000	1.000000	2.00e-15

```
GT(pd.DataFrame([{'rows compared': len(t6),
                  'largest CDF difference': float(t6['CDF diff'].max()),
                  'largest probability difference':
                      float((t6['agg probability']
                             - t6['Table 6 probability']).abs().max())}]),
   formatters={'largest CDF difference': '{x:.2e}',
               'largest probability difference': '{x:.2e}'})
```

Table 21.5.: Worst difference over Table 6, in both the probability and the CDF column.

rows compared	largest CDF difference	largest probability difference
28	4.49e-07	4.93e-07

In Table 21.4 and Table 21.5 both columns match to 5×10^{-7}, which is exact at the six decimals printed. In Table 21.3 the aggregate mean agrees with $\lambda \mathsf{E}[U]$ to twelve digits.

21.4. Table 8: Poisson mixed by a generalized Pareto

The counting law is $N \mid \Theta \sim \text{Poisson}(\Theta)$ with Θ generalized Pareto, $a = 3.4959$, $b = 30.1234$, $c = 1.99321$, in the Klugman parameterization

$$u(\lambda) = \frac{a^b\,\Gamma(b+c)}{\Gamma(b)\Gamma(c)} \cdot \frac{\lambda^{c-1}}{(a+\lambda)^{b+c}}.$$

This mixing family is not one of the built-in `mixed` frequencies, which cover gamma, inverse Gaussian, Sichel, Delaporte and beta. It does not need to be: the paper's method and `aggregate` agree that what a compound distribution needs from the frequency is a probability vector, so compute $p(n) = \mathsf{E}[e^{-\Theta}\Theta^n/n!]$ by quadrature and hand it over as a `dfreq`. The target is Figure 21.3.

The mixing moments are $\mathsf{E}[\Theta] = ac/(b-1)$ and $\mathsf{E}[\Theta^2] = a^2c(c+1)/((b-1)(b-2))$, which give the independent check on the parameterization in Table 21.6, before any of it is used.

```
A, B, C = 3.4959, 30.1234, 1.99321
ETH = A * C / (B - 1)
ETH2 = A ** 2 * C * (C + 1) / ((B - 1) * (B - 2))
GT(pd.DataFrame([
    {'quantity': 'E[Theta] = E[N]', 'recomputed': ETH, 'paper': np.nan},
    {'quantity': 'aggregate mean', 'recomputed': ETH * m1, 'paper': 10.2310},
    {'quantity': 'aggregate standard deviation',
     'recomputed': np.sqrt(ETH * m2 + (ETH2 - ETH ** 2) * m1 ** 2),
     'paper': 24.0489},
]).set_index('quantity'), float_format=6)
```

Table 21.6.: The mixing distribution's moments, and the aggregate mean and standard deviation they imply, against the paper's.

quantity	recomputed	paper
E[Theta] = E[N]	0.239260	—
aggregate mean	10.231041	10.231000
aggregate standard deviation	24.048924	24.048900

Table 8
Aggregate claims cumulative distribution using a Poisson-Generalized Pareto mixture counting distribution with parameters $\alpha = 3.4959$, $\beta = 30.1234$, $\gamma = 1.99321$

Aggregate claims amount[a]	Cumulative distribution values
$\mu - 0.42\sigma \cong 0$	0.798708
$\mu - 0.25\sigma \cong 4$	0.798708
$\mu - 0.00\sigma \cong 10$	0.798708
$\mu + 0.25\sigma \cong 16$	0.810615
$\mu + 0.50\sigma \cong 22$	0.831052
$\mu + 0.75\sigma \cong 28$	0.866018
$\mu + 1.00\sigma \cong 34$	0.874409
$\mu + 1.50\sigma \cong 46$	0.877189
$\mu + 2.00\sigma \cong 58$	0.896084
$\mu + 2.50\sigma \cong 70$	0.973562
$\mu + 3.00\sigma \cong 82$	0.980304
$\mu + 4.00\sigma \cong 106$	0.987965
$\mu + 5.00\sigma \cong 130$	0.997091
$\mu + 8.00\sigma \cong 202$	0.999813
$\mu + 10.00\sigma \cong 250$	0.999972
$\mu + 20.00\sigma \cong 491$	1.000000
$\mu + 50.00\sigma \cong 1212$	1.000000

[a] $\mu = 10.2310$ and $\sigma = 24.0489$. They are, respectively, the mean value and the standard deviation of the aggregate claims distribution.

Figure 21.3.: Bruno et al (2006), Table 8.

```
lc = np.log(A) * B + gammaln(B + C) - gammaln(B) - gammaln(C)
mix_pdf = lambda lam: np.exp(lc + (C - 1) * np.log(lam)
                             - (B + C) * np.log(A + lam))
pn = np.array([quad(lambda t: np.exp(-t + n * np.log(t) - gammaln(n + 1))
                    * mix_pdf(t), 0, np.inf, limit=400)[0]
               for n in range(401)])
keep = pn > 1e-14
ns, qs = np.arange(401)[keep], pn[keep]
FREQ = ('dfreq [' + ' '.join(map(str, ns)) + '] ['
        + ' '.join(f'{q:.14e}' for q in qs) + ']')
a8 = build(f'agg B8 {FREQ} {DSEV}', log2=16, bs=1, padding=1, normalize=False)

T8 = [(0, .798708), (4, .798708), (10, .798708), (16, .810615), (22, .831052),
      (28, .866018), (34, .874409), (46, .877189), (58, .896084),
      (70, .973562), (82, .980304), (106, .987965), (130, .997091),
      (202, .999813), (250, .999972), (491, 1.000000), (1212, 1.000000)]
t8 = pd.DataFrame([{'x': xx, 'agg CDF': a8.density_df.loc[xx, 'F'],
                    'Table 8 CDF': cc} for xx, cc in T8])
t8['diff'] = (t8['agg CDF'] - t8['Table 8 CDF']).abs()
GT(t8.set_index('x'),
   formatters={'agg CDF': 6, 'Table 8 CDF': 6, 'diff': '{x:.2e}'})
```

Table 21.7.: All 17 rows of Table 8, the CDF under the generalized Pareto mixed Poisson, against the rebuilt values.

x	agg CDF	Table 8 CDF	diff
0	0.798708	0.798708	4.00e-07
4	0.798708	0.798708	4.00e-07
10	0.798708	0.798708	4.00e-07
16	0.810615	0.810615	4.93e-07
22	0.831052	0.831052	8.92e-08
28	0.866018	0.866018	8.98e-08
34	0.874409	0.874409	4.31e-07
46	0.877189	0.877189	3.67e-07
58	0.896084	0.896084	2.68e-07
70	0.973562	0.973562	6.24e-08
82	0.980304	0.980304	1.06e-07
106	0.987965	0.987965	2.34e-07
130	0.997091	0.997091	2.67e-07
202	0.999813	0.999813	4.88e-08
250	0.999972	0.999972	3.65e-07
491	1.000000	1.000000	2.29e-09
1,212	1.000000	1.000000	2.34e-12

```
GT(pd.DataFrame([{'counts retained above 1e-14': int(ns[-1]),
                  'sum of p(n)': float(pn.sum()),
                  'p(0)': float(pn[0]), 'Table 8 CDF at 0': 0.798708,
                  'agg mean': a8.est_m, 'agg sd': a8.est_sd,
                  'largest CDF difference': float(t8['diff'].max())}]),
   formatters={'sum of p(n)': 12, 'p(0)': 6, 'Table 8 CDF at 0': 6,
               'agg mean': 6, 'agg sd': 6,
               'largest CDF difference': '{x:.2e}'})
```

Table 21.8.: The truncated frequency vector, the resulting moments, and the worst error over Table 8.

counts retained above 1e-14	sum of p(n)	p(0)	Table 8 CDF at 0	agg mean	agg sd	largest CDF difference
17	0.999999999998	0.798708	0.798708	10.231042	24.049132	4.93e-07

Table 21.7 and Table 21.8 have all seventeen values exact at six decimals, including $p(0) = 0.798708$, which is the probability of no claim and so the CDF's value everywhere below 14.

21.5. Table 7: $\lambda = 91{,}000$, the case the competitors could not do

This is the paper's showcase. At $\lambda = 91{,}000$ the aggregate mean is 3.89 million, the method needs 92,832 convolutions, and Table 4 records that Panjer recursion and De Pril both **underflow** at λ as low as 1004.8. The authors report 1,950 seconds for their own method here. Their FFT comparison stops at $\lambda = 5004.8$, where it took 3,850 seconds.

The support runs to 92,832 × 60 = 5.57 million, so a grid of 2^{23} at `bs = 1` holds it whole and no padding is needed: with the support strictly inside the grid there is nowhere for aliasing to come from. The target is Figure 21.4.

Table 7
Aggregate claims cumulative distribution using a Poisson counting distribution with parameter $\lambda = 91{,}000$

Aggregate claims amount[a]	Cumulative distribution values
$\mu - 5.00\sigma \cong 3{,}820{,}935$	0.000000
$\mu - 4.00\sigma \cong 3{,}835{,}002$	0.000030
$\mu - 3.00\sigma \cong 3{,}849{,}069$	0.001327
$\mu - 2.50\sigma \cong 3{,}856{,}102$	0.006149
$\mu - 2.00\sigma \cong 3{,}863{,}135$	0.022643
$\mu - 1.90\sigma \cong 3{,}864{,}542$	0.028604
$\mu - 1.80\sigma \cong 3{,}865{,}949$	0.035817
$\mu - 1.75\sigma \cong 3{,}866{,}652$	0.039943
$\mu - 1.70\sigma \cong 3{,}867{,}355$	0.044447
$\mu - 1.65\sigma \cong 3{,}868{,}059$	0.049359
$\mu - 1.60\sigma \cong 3{,}868{,}762$	0.054687
$\mu - 1.50\sigma \cong 3{,}870{,}169$	0.066705
$\mu - 1.00\sigma \cong 3{,}877{,}202$	0.158658
$\mu - 0.75\sigma \cong 3{,}880{,}718$	0.226704
$\mu - 0.50\sigma \cong 3{,}884{,}235$	0.308708
$\mu - 0.25\sigma \cong 3{,}887{,}752$	0.401538
$\mu - 0.00\sigma \cong 3{,}891{,}268$	0.500249
$\mu + 0.25\sigma \cong 3{,}894{,}785$	0.598941
$\mu + 0.50\sigma \cong 3{,}898{,}302$	0.691642
$\mu + 0.75\sigma \cong 3{,}901{,}818$	0.773450
$\mu + 1.00\sigma \cong 3{,}905{,}335$	0.841343
$\mu + 1.50\sigma \cong 3{,}912{,}368$	0.933083
$\mu + 2.00\sigma \cong 3{,}919{,}402$	0.977145
$\mu + 2.50\sigma \cong 3{,}926{,}435$	0.993730
$\mu + 3.00\sigma \cong 3{,}933{,}468$	0.998627
$\mu + 4.00\sigma \cong 3{,}947{,}535$	0.999967
$\mu + 5.00\sigma \cong 3{,}961{,}602$	1.000000
$\mu + 8.00\sigma \cong 4{,}003{,}802$	1.000000

[a] $\mu = 3891268.954300$ and $\sigma = 14066.631372$. They are, respectively, the mean value and the standard deviation of the aggregate claims distribution.

Figure 21.4.: Bruno et al (2006), Table 7.

```
import time
t0 = time.time()
a7 = build(f'agg B7 91000 claims {DSEV} poisson', log2=23, bs=1,
           padding=0, normalize=False)
elapsed = time.time() - t0
GT(pd.DataFrame([{'seconds': elapsed, 'grid points': 2 ** 23,
                  'x_max': a7.xs[-1], 'mean': a7.est_m, 'sd': a7.est_sd,
                  'paper mean': 3891268.954300, 'paper sd': 14066.631372,
                  'total mass': a7.agg_density.sum()}]),
   formatters={'seconds': 2, 'grid points': '{x:,.0f}', 'x_max': '{x:,.0f}',
               'mean': '{x:,.4f}', 'sd': '{x:,.4f}',
               'paper mean': '{x:,.4f}', 'paper sd': '{x:,.4f}',
               'total mass': 12})
```

Table 21.9.: The $\lambda = 91{,}000$ build, its grid, its run time and its moments against the paper's.

seconds	grid points	x_max	mean	sd	paper mean	paper sd	total mass
0.69	8,388,608	8,924,193	3,891,268.9543	14,066.6269	3,891,268.9543	14,066.6314	1.000000000000

```
T7 = [(3820935, .000000), (3835002, .000030), (3849069, .001327),
      (3856102, .006149), (3863135, .022643), (3864542, .028604),
      (3865949, .035817), (3866652, .039943), (3867355, .044447),
      (3868059, .049359), (3868762, .054687), (3870169, .066705),
      (3877202, .158658), (3880718, .226704), (3884235, .308708),
      (3887752, .401538), (3891268, .500249), (3894785, .598941),
      (3898302, .691642), (3901818, .773450), (3905335, .841343),
      (3912368, .933083), (3919402, .977145), (3926435, .993730),
      (3933468, .998627), (3947535, .999967), (3961602, 1.000000),
```

```
      (4003802, 1.000000)]
F7 = a7.density_df['F']
t7 = pd.DataFrame([{'x': xx, 'agg CDF': F7.loc[xx], 'Table 7 CDF': cc}
                   for xx, cc in T7])
t7['diff'] = (t7['agg CDF'] - t7['Table 7 CDF']).abs()
GT(t7.set_index('x'),
   formatters={'agg CDF': 6, 'Table 7 CDF': 6, 'diff': '{x:.2e}'})
```

Table 21.10.: All 28 rows of Table 7, the CDF at $\lambda = 91{,}000$, against the rebuilt values.

x	agg CDF	Table 7 CDF	diff
3,820,935	0.000000	0.000000	2.64e-07
3,835,002	0.000030	0.000030	3.70e-07
3,849,069	0.001327	0.001327	2.03e-07
3,856,102	0.006149	0.006149	3.37e-07
3,863,135	0.022643	0.022643	2.25e-07
3,864,542	0.028604	0.028604	4.82e-07
3,865,949	0.035817	0.035817	4.29e-07
3,866,652	0.039943	0.039943	1.04e-07
3,867,355	0.044447	0.044447	1.90e-07
3,868,059	0.049359	0.049359	5.24e-10
3,868,762	0.054687	0.054687	4.45e-07
3,870,169	0.066705	0.066705	1.06e-07
3,877,202	0.158658	0.158658	2.92e-07
3,880,718	0.226704	0.226704	2.36e-07
3,884,235	0.308708	0.308708	1.90e-07
3,887,752	0.401538	0.401538	3.02e-07
3,891,268	0.500249	0.500249	3.90e-07
3,894,785	0.598941	0.598941	4.53e-07
3,898,302	0.691642	0.691642	3.46e-07
3,901,818	0.773450	0.773450	1.18e-08
3,905,335	0.841343	0.841343	2.87e-07
3,912,368	0.933083	0.933083	1.29e-07
3,919,402	0.977145	0.977145	3.92e-07
3,926,435	0.993730	0.993730	2.21e-07
3,933,468	0.998627	0.998627	3.89e-07
3,947,535	0.999967	0.999967	8.37e-09
3,961,602	1.000000	1.000000	3.11e-07
4,003,802	1.000000	1.000000	7.43e-12

```
GT(pd.DataFrame([{'rows compared': len(t7),
                  'largest CDF difference': float(t7['diff'].max()),
                  'paper seconds for its own method': 1950,
                  'aggregate seconds': elapsed}]),
   formatters={'largest CDF difference': '{x:.2e}', 'aggregate seconds': 2})
```

Table 21.11.: Worst error over Table 7, and the run time beside the paper's own 1,950 seconds.

rows compared	largest CDF difference	paper seconds for its own method	aggregate seconds
28	4.82e-07	1,950	0.69

Table 21.10 and Table 21.11 have all 28 values exact at six decimals, in the fraction of a second Table 21.9 records.

21.6. Comments

21.6.1. The timing claim has not aged well

The paper's case rests on operation counts and wall clock. Table 3 argues the FFT needs 6 to 41 times more real multiplications than direct thresholded convolution, and Table 4 shows the FFT taking 3,850 seconds at $\lambda = 5004.8$ against 41 for the proposed method.

The multiplication count was never the right proxy, and the paper half says so: "the number of multiplications is only one of so many elements affecting the procedure's processing time". What it misses is the direction of the error. The authors count the FFT as $O(N \log N)$ per convolution and then sum over 92,832 convolutions, because their formalization performs the convolutions **one at a time**, each an inverse transform of $\phi_{n-1}\phi_1$. That is not how one computes a compound distribution with an FFT. The frequency enters through its probability generating function applied pointwise to the severity transform, so the whole compound distribution costs **two** transforms of length 2^{23} regardless of whether λ is 500 or 91,000. Their equation (19) sums a per-convolution cost that a correct implementation never pays.

That is why the reproduction runs in seconds rather than hours, and why the ratio to their reported time is four orders of magnitude rather than the factor of 40 they anticipated in the other direction. Some of the gap is twenty years of hardware. The structural part is not.

21.6.2. What survives, and it is not nothing

The threshold idea is sound and `aggregate` uses a version of it. Table 21.8 needed the frequency vector only out to $n = 17$, because $p(n)$ had fallen below 10^{-14} by then, and that truncation is exactly the paper's first phase. Their control, choosing the cutoff so that the retained mass **and the first three moments** are accurate to a stated tolerance, rather than by eye, is a good discipline and the closest thing in the paper to a genuinely portable idea. `aggregate`'s validation reports the analogous quantity as a moment comparison after the fact.

The underflow point also survives. Panjer recursion really does fail at $\lambda = 1004.8$ here while the transform does not, and that is a real structural difference between recursions that propagate a small $g(0)$ and methods that do not. It just happens to be an argument for the FFT rather than against it.

21.6.3. The severity is the interesting part

The 14-point severity carries 45.6% of its mass at the single value 60 and another 9.9% at 55, so more than half the distribution sits on two atoms at the top of the range. That is what makes the case a fair stress test: the aggregate is a convolution of a very lumpy lattice law, and a method that smoothed it would show up immediately. Both the paper's method and `aggregate` reproduce the probability column, not just the CDF, which is the stronger test of the two.

22. Gollini and Rougier on a catastrophe model event loss table

Reproducing Gollini and Rougier (2016)

22.1. Context and problem

A catastrophe model summarizes its output as an **event loss table** (ELT): one row per simulated event, giving the event's annual rate of occurrence λ_i and the loss to the portfolio should it occur. The loss may be a fixed amount x_i or a distribution around it, the secondary uncertainty. Gollini and Rougier (2016) model the annual loss as a compound Poisson: events arrive at rate $\lambda = \sum_i \lambda_i$, and each arriving event is event i with probability λ_i/λ, so the severity Y is a mixture with as many components as the table has rows. Over t years the loss is $S_t = Y_1 + \cdots + Y_{N_t}$ with N_t Poisson with mean λt.

The question they ask is the solvency one: given capital s_0, is $\mathsf{P}(S_1 \geq s_0) \leq \kappa_0$, with $\kappa_0 = 0.005$ under Solvency II? They compare four upper bounds on $\mathsf{P}(S_t \geq s)$, Markov, Cantelli, a moment bound and Chernoff, against two methods they call approximately exact: Monte Carlo simulation, sized by a design calculation on a Jeffreys interval, and Panjer recursion on a compressed table. The test case is a US hurricane ELT with 32,060 events. The authors conclude that the moment bound gives the best blend of tightness and speed, and they write that "there are no exact methods for computing tail probabilities when taking into account limited computing resources". Their Panjer recursion took 4,651 seconds on the table rounded to the nearest $100 and was not attempted on anything finer.

Gollini and Rougier (2016) is a method paper, and the roles reverse. `aggregate` computes the distribution of the full table, and that distribution is the ground truth every bound and every compressed result is measured against. The chapter reproduces the arXiv version of 2015, whose exhibit numbers match the authors' published script; the version in *The Journal of Operational Risk* may number its exhibits differently.

22.2. Paper solution and gold standard

22.2.1. The four bounds

Write S for S_t, with mean $\mu = \mathsf{P}S$ and variance σ^2. The four bounds are

$$
\begin{aligned}
\text{Markov:}\quad & \mathsf{P}(S \ge s) \le \mu/s, \\
\text{Cantelli:}\quad & \mathsf{P}(S \ge s) \le \frac{\sigma^2}{\sigma^2 + (s-\mu)^2}, \quad s \ge \mu, \\
\text{moment:}\quad & \mathsf{P}(S \ge s) \le \min_{k=1,2,\dots} \frac{\mathsf{P}S^k}{s^k}, \\
\text{Chernoff:}\quad & \mathsf{P}(S \ge s) \le \inf_{\upsilon > 0} e^{-\upsilon s} M_S(\upsilon), \qquad M_S(\upsilon) = \exp\{\lambda t (M_Y(\upsilon) - 1)\}.
\end{aligned}
$$

The moments of a compound Poisson follow from the recursion the authors take from Ross, their equation (3),

$$
\mathsf{P}S^k = \lambda t \sum_{j=0}^{k-1} \binom{k-1}{j} \mathsf{P}S^j \, \mathsf{P}Y^{k-j}, \qquad \mathsf{P}Y^k = \sum_i \frac{\lambda_i}{\lambda} \mathsf{P}X_i^k,
$$

where X_i is the loss from event i. Section 5 of the paper treats fixed, gamma and capped gamma losses. A fixed loss has $\mathsf{P}X_i^k = x_i^k$ and $M_i(\upsilon) = e^{\upsilon x_i}$. A gamma loss with mean x_i and a common coefficient of variation θ has shape $\alpha = 1/\theta^2$ and rate $\beta_i = \alpha/x_i$, with $\mathsf{P}X_i^k = \Gamma(\alpha + k)/(\Gamma(\alpha)\beta_i^k)$ and $M_i(\upsilon) = (\beta_i/(\beta_i - \upsilon))^\alpha$ for $\upsilon < \beta_i$. A gamma loss capped at u moves the mass above u to an atom at u, so that, with P the regularized lower incomplete gamma function,

$$
\begin{aligned}
\mathsf{P}\min(X_i, u)^k &= \frac{\Gamma(\alpha + k)}{\Gamma(\alpha)\beta_i^k} P(\alpha + k, \beta_i u) + u^k \{1 - P(\alpha, \beta_i u)\}, \\
M_i(\upsilon; u) &= \Big(\frac{\beta_i}{\beta_i - \upsilon}\Big)^\alpha P(\alpha, (\beta_i - \upsilon)u) + \{1 - P(\alpha, \beta_i u)\} e^{\upsilon u}.
\end{aligned}
$$

The Appendix sets the optimizations. The moment bound is minimized over integer k up to a ceiling found from a gamma approximation to S. The Chernoff bound is minimized over 1,001 equally spaced values of υ in $(0, \min_i \beta_i)$; for fixed losses the authors substitute $\theta = 0.1$, which gives $\upsilon < \min_i 100/x_i$. **Compression** rounds each loss to the nearest 10^{-d} dollars, merges rows with equal losses by adding their rates, and drops any row that rounds to zero; $d = -4$ rounds to the nearest \$10,000 and $d = 0$ leaves the table alone.

22.2.2. What the paper publishes that can be checked

The authors publish no table of probabilities. Four things can be matched exactly: Table II, the first and last five rows of the ELT; Table III, the same after compression at $d = -4$, whose rates are sums over the raw table; the row count at each d, in the legend of Figure 2; and Figure 1, a binomial calculation with no data in it. The curves of Figures 2, 3 and 5 to 7 can be compared by eye with the exhibit images, and the bound curves can be recomputed exactly from the paper's equations. The timings of Tables IV to VII are seconds on a laptop of 2015 and are re-timed rather than reproduced.

22.2.3. The data

The ELT is distributed with the authors' R package `tailloss`, version 1.0, published on CRAN in 2015 under the GPL. It was supplied by Peter Taylor and Dickie Whitaker. The file is committed to this book's repository unchanged, as `data/UShurricane.csv.gz`, with its source and license recorded beside it. The package also contains the authors' script for every figure and timing table, and the script settles the conventions the paper leaves loose: the grids of s, the random number seeds, and the search ranges above. This chapter implements the bounds from the paper's equations and consults the script only for those settings.

Figure 22.1 shows Tables II and III. Table 22.1 is Table II recomputed from the committed file, and Table 22.2 is Table III recomputed by compression.

Table II: ELT US Hurricane dataset. Row i represents an event with arrival rate λ_i, and expected loss x_i.

Event ID	Arrival rate, yr^{-1}	Expected Loss, $
1	0.09265	1
2	0.03143	2
3	0.02159	3
4	0.01231	4
5	0.01472	5
⋮	⋮	⋮
32056	0.00001	17593790
32057	0.00001	18218506
32058	0.00001	18297003
32059	0.00001	19970669
32060	0.00001	24391615

Table III: ELT US Hurricane dataset, after rounding and merging to $10k ($d = -4$). *Cf.* Table II.

Event ID	Arrival rate, yr^{-1}	Expected Loss, $10k
1	0.35764	1
2	0.16864	2
3	0.16088	3
4	0.12135	4
5	0.12239	5
⋮	⋮	⋮
1141	0.00001	1759
1142	0.00001	1822
1143	0.00001	1830
1144	0.00001	1997
1145	0.00001	2439

Figure 22.1.: Gollini and Rougier (2016), Tables II and III.

```
%xmode minimal
import time

import matplotlib as mpl
import numpy as np
import pandas as pd
from scipy import optimize, special, stats

from aggregate import Aggregate, build as library_build
from bookstyle import BOOK_ROOT, GT, build, show
from house import figure

M = 1e6                              # one million dollars, the paper's unit for s
ELT = pd.read_csv(BOOK_ROOT / 'data' / 'UShurricane.csv.gz', sep=';')
LAM = ELT.Rate.sum()
```

```
def ends(frame, n=5):
    """The first and last `n` rows, the layout of the paper's Tables II and III."""
    return pd.concat([frame.head(n), frame.tail(n)])


ii = ends(ELT.set_index('EventID').rename_axis('event'))
ii['Table II rate'] = [.09265, .03143, .02159, .01231, .01472] + [.00001] * 5
ii['Table II loss'] = [1, 2, 3, 4, 5, 17593790, 18218506, 18297003, 19970669, 24391615]
GT(ii[['Rate', 'Table II rate', 'Loss', 'Table II loss']]
   .rename(columns={'Rate': 'rate', 'Loss': 'loss'}),
   formatters={'rate': 7, 'Table II rate': 5, 'loss': '{x:,.0f}',
               'Table II loss': '{x:,.0f}'})
```

```
Exception reporting mode: Minimal
```

Table 22.1.: Table II recomputed from the committed file, the first and last five events, against the printed values.

event	rate	Table II rate	loss	Table II loss
1	0.0926503	0.09265	1	1
2	0.0314305	0.03143	2	2
3	0.0215918	0.02159	3	3
4	0.0123111	0.01231	4	4
5	0.0147244	0.01472	5	5
32,056	0.0000147	0.00001	17,593,790	17,593,790
32,057	0.0000147	0.00001	18,218,506	18,218,506
32,058	0.0000147	0.00001	18,297,003	18,297,003
32,059	0.0000147	0.00001	19,970,669	19,970,669
32,060	0.0000147	0.00001	24,391,615	24,391,615

```
def compress(d):
    """The authors' compression at `d`: round, merge equal losses, drop zeros.

    The loss is returned in units of 10^-d dollars, as in the paper's tables,
    and in dollars. numpy and R both round halves to even, so ties go the same way.
    """
    loss = np.round(np.round(ELT.Loss.to_numpy(float), d) * 10.0 ** d)
    table = (pd.DataFrame({'Loss': loss, 'Rate': ELT.Rate.to_numpy()})
             .groupby('Loss', as_index=False).Rate.sum())
    table = table[table.Loss > 0].reset_index(drop=True)
    table.index = pd.RangeIndex(1, len(table) + 1, name='event')
    table['Dollars'] = table.Loss * 10.0 ** -d
    return table


TABLES = {d: compress(d) for d in range(-7, 1)}
iii = ends(TABLES[-4])
iii['Table III rate'] = [.35764, .16864, .16088, .12135, .12239] + [.00001] * 5
iii['Table III loss'] = [1, 2, 3, 4, 5, 1759, 1822, 1830, 1997, 2439]
GT(iii[['Rate', 'Table III rate', 'Loss', 'Table III loss']]
   .rename(columns={'Rate': 'rate', 'Loss': 'loss'}),
   formatters={'rate': 7, 'Table III rate': 5, 'loss': '{x:,.0f}',
               'Table III loss': '{x:,.0f}'})
```

Table 22.2.: Table III recomputed by compression at $d = -4$, losses in units of $10,000, against the printed values.

event	rate	Table III rate	loss	Table III loss
1	0.3576421	0.35764	1	1
2	0.1686408	0.16864	2	2
3	0.1608763	0.16088	3	3
4	0.1213495	0.12135	4	4
5	0.1223939	0.12239	5	5
1,141	0.0000147	0.00001	1,759	1,759
1,142	0.0000147	0.00001	1,822	1,822
1,143	0.0000147	0.00001	1,830	1,830
1,144	0.0000147	0.00001	1,997	1,997
1,145	0.0000147	0.00001	2,439	2,439

Every rate in Table 22.1 and Table 22.2 rounds to the printed value, and every loss is exact. The losses in the raw table are whole dollars, all distinct, running from $1 to $24,391,615. Figure 22.2 is the paper's study of compression, and its legend gives the number of rows at each d, which Table 22.3 reproduces exactly. The table also shows what compression does to the model: a coarse rounding sends the small events to zero, so the event rate falls from 6.89 a year to 2.97 at $d = -6$ and to 0.18 at $d = -7$, while the mean annual loss holds up much better.

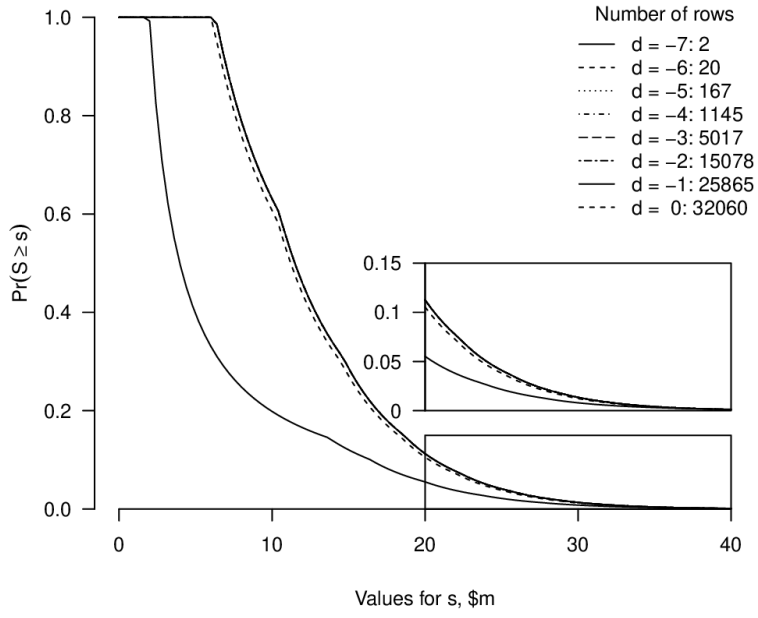


Figure 2: The effect of compression and merging on the US Hurricanes ELT. The curves show the values of the Moment bound on the exceedance probability for one-year total losses. All values of d larger than -7 (only two rows) give very similar outcomes, with values of -5 or larger being effectively identical, and overlaid on the Figure.

Figure 22.2.: Gollini and Rougier (2016), Figure 2.

```
LEGEND = {-7: 2, -6: 20, -5: 167, -4: 1145, -3: 5017, -2: 15078, -1: 25865,
          0: 32060}
GT(pd.DataFrame([{'d': d, 'rounded to': 10.0 ** -d, 'rows': len(t),
                  'Figure 2 rows': LEGEND[d], 'event rate': t.Rate.sum(),
                  'mean annual loss': t.Rate @ t.Dollars}
                 for d, t in TABLES.items()]).set_index('d'),
   formatters={'rounded to': '{x:,.0f}', 'rows': '{x:,.0f}',
               'Figure 2 rows': '{x:,.0f}', 'event rate': 5,
               'mean annual loss': '{x:,.0f}'})
```

Table 22.3.: The compressed table at each d, its number of rows against the legend of Figure 2, its event rate and its mean annual loss in dollars.

d	rounded to	rows	Figure 2 rows	event rate	mean annual loss
-7	10,000,000	2	2	0.18190	1,984,682
-6	1,000,000	20	20	2.97418	6,071,824
-5	100,000	167	167	4.80390	6,289,254
-4	10,000	1,145	1,145	5.66444	6,308,357
-3	1,000	5,017	5,017	6.23392	6,309,300
-2	100	15,078	15,078	6.52876	6,309,374
-1	10	25,865	25,865	6.72018	6,309,377
0	1	32,060	32,060	6.89289	6,309,377

Table 22.4 gives the moments of the full table two ways: summed directly from the rows, and from the `Aggregate` object built on the full table in the next section, both its exact moments and the moments of its discretized distribution. Agreement checks the build. The largest event on the grid is the largest event in the table rounded to the $10 bucket.

```
def elt_aggregate(dollars, rate, bs, log2, t=1):
    """An ELT with fixed losses as an `Aggregate`, built through the class.

    The class takes vectors of any length. DecL does not; see the section on
    the model in DecL.
    """
    a = Aggregate('ELT', exp_en=rate.sum() * t, sev_name='dhistogram',
                  sev_xs=np.asarray(dollars, float),
                  sev_ps=np.asarray(rate) / rate.sum(), freq_name='poisson')
    a.update(log2=log2, bs=bs, normalize=False)
    return a


def exceed(a, s):
    """Pr(S >= s) from the grid of `a`, counting any mass beyond the grid.

    The tail is summed from the right, so a small probability keeps its
    relative precision instead of being 1 - F.
    """
    p = a.agg_density
    tail = np.cumsum(p[::-1])[::-1]
    i = np.ceil(np.asarray(s, float) / a.bs - 1e-9).astype(int)
    out = (np.where(i < len(p), tail[np.minimum(i, len(p) - 1)], 0.0)
           + max(0.0, 1 - p.sum()))
    return np.where(i <= 0, 1.0, out)
```

```
EY = ELT.Rate @ ELT.Loss / LAM
EY2 = ELT.Rate @ ELT.Loss ** 2 / LAM
start = time.perf_counter()
H = elt_aggregate(ELT.Loss, ELT.Rate, 10.0, 23)
H_SECONDS = time.perf_counter() - start
GT(pd.DataFrame({
    'from the rows': [EY, EY2 ** 0.5, LAM * EY, (LAM * EY2) ** 0.5,
                      ELT.Loss.max()],
    'aggregate, exact': [H.sev_m, np.hypot(H.sev_m, H.sev_sd), H.actual_m,
                         H.actual_sd, np.nan],
    'aggregate, on the grid': [H.est_sev_m, np.hypot(H.est_sev_m, H.est_sev_sd),
                               H.est_m, H.est_sd,
                               H.xs[np.flatnonzero(H.sev_density)[-1]]],
}, index=pd.Index([r'$\mathsf P Y$', r'$(\mathsf P Y^2)^{1/2}$',
                   r'$\mathsf P S_1$', r'$\mathrm{sd}(S_1)$', 'largest event'],
                  name='quantity')),
   float_format='{x:,.0f}')
```

Table 22.4.: Moments of the full table in dollars, summed from the rows, from the `Aggregate` object, and from its distribution on a grid of $10 buckets.

quantity	from the rows	aggregate, exact	aggregate, on the grid
$\mathsf{P}Y$	915,346	915,346	915,346
$(\mathsf{P}Y^2)^{1/2}$	1,948,883	1,948,883	1,948,883
$\mathsf{P}S_1$	6,309,377	6,309,377	6,309,377
$\mathrm{sd}(S_1)$	5,116,658	5,116,658	5,116,658
largest event	24,391,615	—	24,391,620

22.3. The `aggregate` reproduction

22.3.1. The model in DecL

At $d = -6$ the table has twenty rows, losses of $1m to $24m, and it is small enough to write by hand. Rounding to the nearest million sends every event under $0.5m to zero, so the rate is the 2.974 of Table 22.3. In millions, the DecL program is

```
T6 = TABLES[-6]
LAM6 = T6.Rate.sum()
# Eight decimals, with the rounding residual put on the largest probability so
# that the vector as printed sums to one.
P6 = np.round(T6.Rate.to_numpy() / LAM6, 8)
P6[P6.argmax()] += 1 - P6.sum()
XS6 = ' '.join(f'{x:.0f}' for x in T6.Loss)
PS6 = ' '.join(f'{p:.8f}'.rstrip('0') for p in P6)
hurricane = build(f'''
agg Hurricane
  {LAM6:.6f} claims
  dsev [{XS6}] [{PS6}]
  poisson
''', bs=1 / 64, log2=13, normalize=False)
show(hurricane)
```

```
agg Hurricane
  2.974183 claims
  dsev [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] [0.58889746 0.21928748 0.0672295
  0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063 0.00941256 0.00198404 0.00452247
  7.436e-05 0.00025122 0.00253892 0.00435067 0.0007371 1.486e-05 4.95e-06 4.95e-06]
  poisson
```

The `dsev` clause gives the losses and their probabilities λ_i/λ, and `poisson` makes the count Poisson with the stated mean. The cap of Section 6.4 is a limit clause, `5 xs 0`, placed before the severity: every event loss is replaced by its minimum with \$5m.

```
capped = build(f'''
agg Hurricane.Capped
  {LAM6:.6f} claims
  5 xs 0
  dsev [{XS6}] [{PS6}]
  poisson
''', bs=1 / 64, log2=13, normalize=False)
show(capped)
```

```
agg Hurricane.Capped
  2.974183 claims
  5 xs 0
  dsev [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] [0.58889746 0.21928748 0.0672295
  0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063 0.00941256 0.00198404 0.00452247
  7.436e-05 0.00025122 0.00253892 0.00435067 0.0007371 1.486e-05 4.95e-06 4.95e-06]
  poisson
```

The gamma losses of Section 6.3 are a mixed severity: `sev gamma` with a vector of means, a common coefficient of variation, and the probabilities as weights.

```
gamma = build(f'''
agg Hurricane.Gamma
  {LAM6:.6f} claims
  sev gamma [{XS6}] cv 0.5 wts [{PS6}]
  poisson
''', bs=1 / 64, log2=13, normalize=False)
show(gamma)
```

```
agg Hurricane.Gamma
  2.974183 claims
  sev gamma [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] cv 0.5 wts [0.58889746 0.21928748
  0.0672295 0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063 0.00941256 0.00198404
  0.00452247 7.436e-05 0.00025122 0.00253892 0.00435067 0.0007371 1.486e-05 4.95e-06 4.95e-06]
  poisson
```

Table 22.5 sets the three programs beside the exact values for the same three models, which the following sections compute. The coarse table is about 4% high at \$20m with fixed losses, 5 to 7% low with gamma losses, and 15 to 20% high under the cap. Compression to \$1m drops the events under \$0.5m, which lowers the mean, and rounds the large events, which moves the tail in either direction. Under the cap the rounding sends the events between \$4.5m and \$5m up to the cap, and the far tail of the capped loss is the probability that many events reach it.

```
def gamma_severity(table, h, cap=np.inf, theta=0.5, chunk=400):
    """The rate weighted gamma mixture, discretized to a grid of buckets of h.

    Each event's loss is gamma with mean its tabulated loss and coefficient of
    variation theta. A bucket holds the probability of the interval of width h
    centered on it; under a cap, everything above the cap goes to the atom at
    the cap. Returns the grid, its probabilities, and any mass lost past the
    end of an uncapped grid.
    """
    alpha = 1 / theta ** 2
    x = table.Dollars.to_numpy()
    w = table.Rate.to_numpy() / table.Rate.sum()
    top = cap if np.isfinite(cap) else 8 * x.max()
    n = int(round(top / h)) + 1
    edges = (np.arange(n) + 0.5) * h
    F = np.zeros(n)
    for i in range(0, len(x), chunk):                # chunks bound the memory
        F += (special.gammainc(alpha, alpha * edges[:, None] / x[None, i:i + chunk])
              @ w[i:i + chunk])
    p = np.diff(F, prepend=0.0)
    if np.isfinite(cap):
        p[-1] += 1 - F[-1]
    return np.arange(n) * h, p, 1 - F[-1]


def gamma_aggregate(table, h, log2, t=1, cap=np.inf):
    """The gamma ELT as an `Aggregate`, on a severity built by `gamma_severity`."""
    xs, ps, _ = gamma_severity(table, h, cap)
    keep = ps > 0
    a = Aggregate('ELT', exp_en=table.Rate.sum() * t, sev_name='dhistogram',
                  sev_xs=xs[keep], sev_ps=ps[keep] / ps[keep].sum(),
                  freq_name='poisson')
    a.update(log2=log2, bs=h, normalize=False)
    return a


S3 = np.array([20, 30, 40]) * M
H_CAPPED = elt_aggregate(np.minimum(ELT.Loss, 5 * M), ELT.Rate, 10.0, 23)
start = time.perf_counter()
G4 = gamma_aggregate(TABLES[-4], 1e4, 15)
G4_SECONDS = time.perf_counter() - start
coarse = pd.DataFrame(
    [(name, s / M, exceed(obj, s / M)[()], exceed(ref, s)[()])
     for name, obj, ref in (('Hurricane', hurricane, H),
                            ('Hurricane.Capped', capped, H_CAPPED),
                            ('Hurricane.Gamma', gamma, G4))
     for s in S3],
    columns=['program', 's, $m', 'DecL, d = -6', 'exact'])
coarse['ratio'] = coarse['DecL, d = -6'] / coarse['exact']
GT(coarse.set_index(['program', 's, $m']),
   formatters={'DecL, d = -6': '{x:.3e}', 'exact': '{x:.3e}', 'ratio': 3})
```

Table 22.5.: The three DecL programs on the table compressed to $1m, against the exact values on the full table (fixed losses) and on the table compressed to $10,000 (gamma losses), at $20m, $30m and $40m.

program	*s, $m*	DecL, d = -6	exact	ratio
Hurricane	20	2.602e-02	2.496e-02	1.043
	30	2.254e-03	2.200e-03	1.024
	40	1.654e-04	1.633e-04	1.013
Hurricane.Capped	20	2.431e-03	2.108e-03	1.153
	30	1.388e-05	1.173e-05	1.184
	40	3.687e-08	3.064e-08	1.203
Hurricane.Gamma	20	3.185e-02	3.357e-02	0.949
	30	6.638e-03	7.077e-03	0.938
	40	1.371e-03	1.475e-03	0.930

22.3.2. The full table

A DecL vector of about 500 elements does not parse, as the comments below show, and the full table has 32,060. The full table is data rather than a model a person writes, so it goes through the `Aggregate` class, which takes vectors of any length. Table 22.6 shows it built on a ladder of bucket sizes. Building on a bucket of size h rounds every loss to the nearest multiple of h, which is the authors' compression by another route, so a $10,000 bucket corresponds to $d = -4$ and a $10 bucket to $d = -1$.

```
S4 = np.array([10, 20, 30, 40]) * M
ladder = []
for bs, log2, d in ((1e4, 14, -4), (1e3, 17, -3), (1e2, 20, -2)):
    start = time.perf_counter()
    a = elt_aggregate(ELT.Loss, ELT.Rate, bs, log2)
    ladder.append((bs, log2, d, time.perf_counter() - start, *exceed(a, S4)))
ladder.append((10.0, 23, -1, H_SECONDS, *exceed(H, S4)))
GT(pd.DataFrame(ladder, columns=['bucket', 'log2', 'd', 'seconds', '10m', '20m',
                                 '30m', '40m']).set_index('bucket'),
   formatters={'bucket': '{x:,.0f}', 'seconds': 3, '10m': 6, '20m': 6, '30m': 6,
               '40m': 6})
```

Table 22.6.: The full table on four bucket sizes: the grid, the compression each bucket corresponds to, the seconds to build, and $\mathsf{P}(S_1 \geq s)$ at $10m to $40m.

bucket	log2	d	seconds	10m	20m	30m	40m
10,000	14	-4	0.020	0.182844	0.024991	0.002203	0.000164
1,000	17	-3	0.030	0.182686	0.024965	0.002201	0.000163
100	20	-2	0.116	0.182670	0.024962	0.002200	0.000163
10	23	-1	0.794	0.182668	0.024962	0.002200	0.000163

The $100 and $10 rows of Table 22.6 agree to five significant figures at every s, and the $1,000 row to four. The rounding error of a bucket of h is at most $h/2$ per event, and with about seven events a year the total moves by tens of dollars on losses of tens of millions, so the $10 row is the uncompressed answer to the precision shown. A $1 bucket, which would put every loss exactly on the grid, needs

2^{27} buckets to hold the tail, and the transform of that length needs more memory than a book should ask of a reader's machine. It is not needed: the \$10 row is the $d = 0$ answer that the authors ruled out, in under a second, and the rest of the chapter uses it as the reference.

22.3.3. Panjer recursion on a compressed table

The authors' Panjer recursion runs on the table compressed to an integer lattice. On the same compressed table, Panjer recursion and the FFT compute the same discrete distribution, so they must agree to rounding error. Table 22.7 sets a vectorized Panjer recursion at $d = -4$, the paper's working compression, beside `aggregate` on the same table with a \$10,000 bucket, and compares both with the uncompressed reference. The last column is the compression error, which the authors had no means of measuring.

```
def panjer(table, n_max, t=1):
    """Pr(S_t = n), n = 0, ..., n_max, by Panjer's recursion on an integer table.

    For a compound Poisson, n g_n = sum_j a_j g_{n-j} with a_j the rate of loss j
    times j times t; each step is one dot product.
    """
    loss = table.Loss.to_numpy().astype(int)
    a = np.zeros(loss.max() + 1)
    np.add.at(a, loss, t * loss * table.Rate.to_numpy())
    g = np.zeros(n_max + 1)
    g[0] = np.exp(-t * table.Rate.sum())
    for n in range(1, n_max + 1):
        k = min(n, len(a) - 1)
        g[n] = a[1:k + 1] @ g[n - 1::-1][:k] / n
    return g


g4 = panjer(TABLES[-4], int(40 * M / 1e4))
A4 = elt_aggregate(TABLES[-4].Dollars, TABLES[-4].Rate, 1e4, 14)
EXACT = exceed(H, S4)
pj = pd.DataFrame({'s, $m': S4 / M,
                   'Panjer, d = -4': [1 - g4[:int(s / 1e4)].sum() for s in S4],
                   'aggregate, d = -4': exceed(A4, S4), 'uncompressed': EXACT})
pj['Panjer abs err'] = (pj['Panjer, d = -4'] - pj['aggregate, d = -4']).abs()
pj['compression rel err'] = pj['aggregate, d = -4'] / pj['uncompressed'] - 1
GT(pj.set_index('s, $m'),
   formatters={'Panjer, d = -4': 6, 'aggregate, d = -4': 6, 'uncompressed': 6})
```

Table 22.7.: Panjer recursion and `aggregate` on the table compressed to \$10,000, against the uncompressed reference: $\mathsf{P}(S_1 \geq s)$ and the relative error of compression.

s, $m	Panjer, d = -4	aggregate, d = -4	uncompressed	Panjer abs err	compression rel err
10	0.182807	0.182807	0.182668	1.39e-16	7.62e-04
20	0.024987	0.024987	0.024962	8.67e-17	1.02e-03
30	0.002203	0.002203	0.002200	1.65e-16	1.11e-03
40	0.000164	0.000164	0.000163	1.34e-16	1.32e-03

The two methods agree to rounding error in Table 22.7, as they must. Compression to \$10,000 overstates the tail by about 0.1%, which is negligible for any purpose the authors had in mind. The cost of the recursion explains why they stopped where they did. Each of its steps sums over every loss on

the lattice, and both the number of steps and the number of losses grow tenfold with each step in d, so the time grows a hundredfold: Table IV has 0.46 seconds at $d = -4$, 41 at $d = -3$ and 4,651 at $d = -2$.

22.3.4. Compression measured against the exact distribution

The authors judged compression with the moment bound, because Panjer recursion is too slow to run without compression and because the moment bound, as they write, "converges to the actual value when the number of events in the time-interval is large". Figure 22.3 shows Figure 22.2 redrawn in its left panel and, on a log scale in its right, the exact distribution of each compressed table, each computed on its own lattice of 10^{-d} dollars.

```
def moment_bound(table, s, t=1, theta=0.0, cap=np.inf, kmax=200, argmin=False):
    """min over k <= kmax of P S^k / s^k, the moments by the recursion (3).

    Moments are taken in units of the largest s, which keeps P S^k in range up
    to k = 200; the authors' heuristic ceiling on k is replaced by a fixed
    generous one, which can only tighten the bound.
    """
    unit = np.max(s)
    x = table.Dollars.to_numpy() / unit
    w = table.Rate.to_numpy() / table.Rate.sum()
    k = np.arange(1, kmax + 1)
    if theta == 0:
        ey = w @ np.minimum(x, cap / unit)[:, None] ** k
    else:
        alpha = 1 / theta ** 2
        beta = alpha / x
        raw = np.exp(special.gammaln(alpha + k)[None, :] - special.gammaln(alpha)
                     - k[None, :] * np.log(beta)[:, None])
        if np.isfinite(cap):
            u = cap / unit
            raw = (raw * special.gammainc(alpha + k[None, :], (beta * u)[:, None])
                   + special.gammaincc(alpha, beta * u)[:, None] * u ** k[None, :])
        ey = w @ raw
    lam = table.Rate.sum() * t
    es = np.ones(kmax + 1)
    for n in range(1, kmax + 1):
        j = np.arange(n)
        es[n] = lam * (special.comb(n - 1, j) * es[j] * ey[n - j - 1]).sum()
    r = np.asarray(s, float) / unit
    with np.errstate(divide='ignore', over='ignore', invalid='ignore'):
        ratio = es[None, 1:] / r[:, None] ** k[None, :]
    ratio = np.where(np.isfinite(ratio), ratio, np.inf)
    bound = np.minimum(1, ratio.min(axis=1))
    return (bound, ratio.argmin(axis=1) + 1) if argmin else bound


SV = np.linspace(0, 40, 101) * M
LATTICE_LOG2 = {-7: 6, -6: 8, -5: 11, -4: 14, -3: 17, -2: 20}
EXACT_D = {d: exceed(elt_aggregate(TABLES[d].Dollars, TABLES[d].Rate,
                                   10.0 ** -d, log2), SV)
           for d, log2 in LATTICE_LOG2.items()}
EXACT_D[-1] = EXACT_D[0] = exceed(H, SV)
BOUND_D = {d: moment_bound(TABLES[d], SV) for d in TABLES}

fig, axs = figure('full', 1, 2, layout='constrained')
colors = dict(zip(TABLES, mpl.colormaps['viridis'](np.linspace(0, 0.9, 8))))
for d in TABLES:
```

```
    axs[0].plot(SV / M, BOUND_D[d], c=colors[d], lw=1, label=f'd = {d}')
    axs[1].semilogy(SV / M, EXACT_D[d], c=colors[d], lw=1)
    axs[1].semilogy(SV / M, BOUND_D[d], c=colors[d], lw=1, ls='--')
axs[0].set(xlabel='s, $m', ylabel='moment bound', title='Figure 2, as drawn')
axs[0].legend(fontsize=6)
axs[1].set(xlabel='s, $m', ylabel=r'$\mathsf{P}(S_1 \geq s)$', ylim=(1e-5, 1.5),
           title='exact (solid) and bound (dashed)');
```

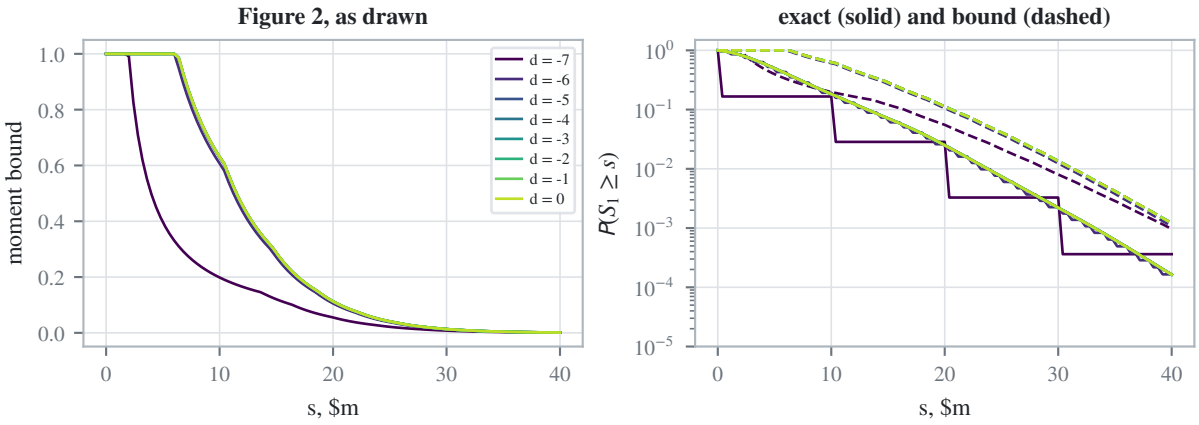


Figure 22.3.: Reproducing Figure 2. Left, the moment bound for each compressed table, as the paper draws it. Right, the same bounds (dashed) with the exact exceedance probability of each compressed table (solid), on a log scale; $d = -1$ and $d = 0$ are the uncompressed reference.

Table 22.8 gives the values behind Figure 22.3 at three points, each entry the ratio to the uncompressed value. At $d = -6$, the twenty row table the authors describe as "almost the same as that with no compression at all", the exact tail probability is 4% high at $20m, while the moment bound moves the other way, 6% low. The bound is no guide to the compression error because it changes with the table for reasons of its own: compression lowers the mean and the higher moments, so the bound falls, while the rounding of the large events raises the tail. From $d = -4$ on, both are within 0.2% of the uncompressed values.

```
pick = [np.flatnonzero(np.isclose(SV, s))[0] for s in S3]
GT(pd.DataFrame([{'d': d, 'rows': len(TABLES[d]),
                  **{f'exact {s / M:.0f}m': EXACT_D[d][i] / EXACT_D[0][i]
                     for s, i in zip(S3, pick)},
                  **{f'bound {s / M:.0f}m': BOUND_D[d][i] / BOUND_D[0][i]
                     for s, i in zip(S3, pick)}}
                 for d in range(-7, -1)]).set_index('d'),
   float_format=4, formatters={'rows': '{x:,.0f}'})
```

Table 22.8.: The exact exceedance probability and the moment bound of each compressed table, each as a ratio to its uncompressed value, at $20m, $30m and $40m.

d	rows	exact 20m	exact 30m	exact 40m	bound 20m	bound 30m	bound 40m
-7	2	1.1411	1.4880	2.2189	0.4906	0.5927	0.7999
-6	20	1.0426	1.0244	1.0129	0.9360	0.9125	0.8915
-5	167	1.0065	1.0065	1.0071	0.9952	0.9944	0.9938
-4	1,145	1.0010	1.0011	1.0013	0.9998	0.9999	0.9999
-3	5,017	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000
-2	15,078	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

22.3.5. Fixed event losses: Figure 3

Figure 22.4 is the paper's main comparison: losses fixed at their tabulated values, one year, no cap, and every method at $d = -4$. The authors' script evaluates the curves at 101 points from $5m to $40m, which is what the figure's axis shows; the caption gives $0m. The authors observe that "the Chernoff and Moment bounds converge, and also, in this case, converge on the Panjer and Monte Carlo estimates."

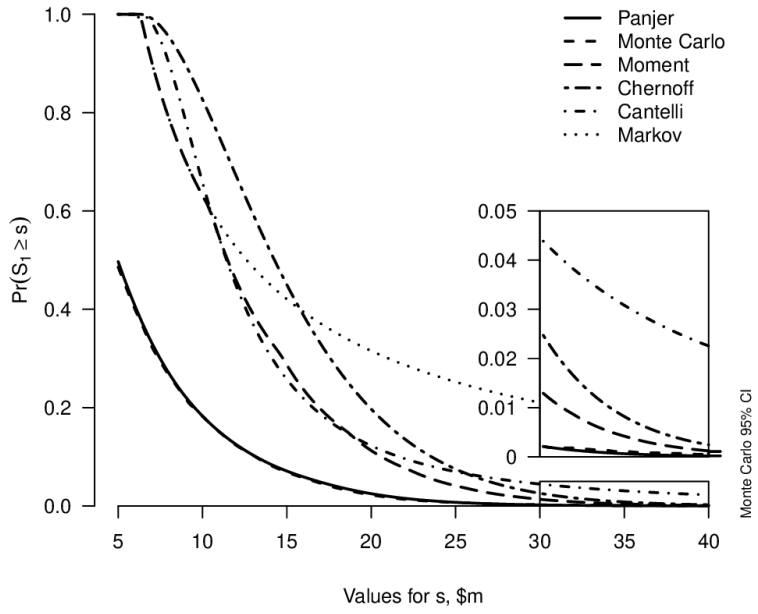


Figure 3: Exceedance probabilities for the methods, with rounding of $d = -4$ on the US Hurricanes ELT. The legend shows the Monte Carlo 95% confidence interval for p_0 at $s_0 = \$40\,\text{m}$; see section 4. Each curve comprises 101 points, equally-spaced between $0 m and $40 m. Timings are given in Table IV. For later reference, this Figure has $t = 1$, $u = \infty$, $\theta = 0$, and $d = -4$.

Figure 22.4.: Gollini and Rougier (2016), Figure 3.

```
def mean_var(table, t=1, theta=0.0, cap=np.inf):
    """Mean and variance of S_t, in dollars; the variance is lambda t P Y^2."""
    x = table.Dollars.to_numpy()
    w = table.Rate.to_numpy() / table.Rate.sum()
    lam = table.Rate.sum() * t
```

```
    if theta == 0:
        y = np.minimum(x, cap)
        return lam * (w @ y), lam * (w @ y ** 2)
    alpha = 1 / theta ** 2
    beta = alpha / x
    m1 = (alpha / beta) * (special.gammainc(alpha + 1, beta * cap) if np.isfinite(cap) else 1)
    m2 = (alpha * (alpha + 1) / beta ** 2) * (special.gammainc(alpha + 2, beta * cap)
                                              if np.isfinite(cap) else 1)
    if np.isfinite(cap):
        tail = special.gammaincc(alpha, beta * cap)
        m1, m2 = m1 + cap * tail, m2 + cap ** 2 * tail
    return lam * (w @ m1), lam * (w @ m2)


def markov(table, s, **model):
    """The Markov bound."""
    return np.minimum(1, mean_var(table, **model)[0] / np.asarray(s, float))


def cantelli(table, s, **model):
    """The Cantelli bound, one above the mean."""
    mu, var = mean_var(table, **model)
    s = np.asarray(s, float)
    return np.where(s >= mu, var / (var + (s - mu) ** 2), 1.0)


def event_mgf(x, v, theta=0.0, cap=np.inf):
    """Each event's MGF at each v: rows are events, columns values of v, in $m.

    Under a cap the gamma MGF is finite for every v. Where v >= beta the
    integral over [0, cap] is a confluent hypergeometric function rather than
    an incomplete gamma function.
    """
    if theta == 0:
        return np.exp(np.outer(np.minimum(x, cap), v))
    alpha = 1 / theta ** 2
    beta = np.broadcast_to((alpha / x)[:, None], (len(x), len(v)))
    c = beta - v[None, :]
    pos = c > 0
    body = np.full(c.shape, np.inf)
    if np.isinf(cap):
        body[pos] = (beta[pos] / c[pos]) ** alpha
        return body
    body[pos] = special.gammainc(alpha, c[pos] * cap) * (beta[pos] / c[pos]) ** alpha
    body[~pos] = ((beta[~pos] * cap) ** alpha / special.gamma(alpha + 1)
                  * special.hyp1f1(alpha, alpha + 1, -c[~pos] * cap))
    return body + special.gammaincc(alpha, beta * cap) * np.exp(v[None, :] * cap)


def paper_v(table, theta=0.0, cap=np.inf, nk=1001):
    """The authors' Chernoff search: nk equally spaced v, per $m, from the Appendix."""
    x = table.Dollars.to_numpy() / M
    if theta == 0:
        return np.linspace(0, np.min(100 / np.minimum(x, cap / M)), nk)
    alpha = 1 / theta ** 2
    return np.linspace(0, min(np.min(100 / x), np.min(alpha / x)), nk + 1)[:-1]


def chernoff(table, s, v, t=1, theta=0.0, cap=np.inf, interior=False):
    """The Chernoff bound minimized over the grid v, set to 1 below the mean."""
    x = table.Dollars.to_numpy() / M
    rate = table.Rate.to_numpy()
```

```
    log_mgf = t * (rate @ event_mgf(x, v, theta, cap / M) - rate.sum())
    s_m = np.asarray(s, float) / M
    objective = log_mgf[None, :] - np.outer(s_m, v)
    if interior:                    # an unrestricted search must not hit its edge
        assert (objective.argmin(axis=1) < len(v) - 1).all()
    mu = mean_var(table, t, theta, cap)[0] / M
    return np.where(s_m > mu, np.minimum(1, np.exp(objective.min(axis=1))), 1.0)


def bounds(table, s, t=1, theta=0.0, cap=np.inf):
    """Every bound for one case, keyed by the paper's legend.

    With fixed losses the optimal v must lie inside the authors' range, which
    the interior check enforces; with gamma losses it may sit at the edge.
    """
    model = dict(t=t, theta=theta, cap=cap)
    out = {'Moment': moment_bound(table, s, **model),
           'Chernoff': chernoff(table, s, paper_v(table, theta, cap),
                                interior=theta == 0, **model),
           'Cantelli': cantelli(table, s, **model)}
    if theta == 0:
        out['Markov'] = markov(table, s, **model)
    return out


STYLE = {'exact': dict(c='k', lw=1.4), 'Moment': dict(c='C0', lw=1),
         'Chernoff': dict(c='C1', lw=1),
         'Chernoff, unrestricted': dict(c='C1', lw=1, ls='--'),
         'Cantelli': dict(c='C2', lw=1), 'Markov': dict(c='C3', lw=1, ls=':')}


def case_figure(s, curves, t=1, floor=1e-5):
    """The paper's linear panel beside a log panel, every curve in both."""
    fig, axs = figure('full', 1, 2, layout='constrained')
    for name, y in curves.items():
        axs[0].plot(s / M, y, label=name, **STYLE[name])
        axs[1].semilogy(s / M, y, **STYLE[name])
    ylabel = rf'$\mathsf{{P}}(S_{{{t}}} \geq s)$'
    axs[0].set(xlabel='s, $m', ylabel=ylabel, ylim=(0, 1.02), title='as the paper draws it')
    axs[1].set(xlabel='s, $m', ylabel=ylabel, ylim=(floor, 1.5), title='log scale')
    axs[0].legend(fontsize=6)
    return fig


T4 = TABLES[-4]
CASES = {'Figure 3': dict(t=1, theta=0.0, cap=np.inf, s=np.linspace(5, 40, 101) * M,
                          points=np.array([10, 20, 30, 40]) * M),
         'Figure 5': dict(t=1, theta=0.5, cap=np.inf, s=np.linspace(0, 60, 101) * M,
                          points=np.array([20, 40, 60]) * M),
         'Figure 6': dict(t=1, theta=0.5, cap=5 * M, s=np.linspace(0, 40, 101) * M,
                          points=S3),
         'Figure 7': dict(t=10, theta=0.5, cap=5 * M, s=np.linspace(0, 120, 101) * M,
                          points=np.array([60, 90, 120]) * M)}
EXACT_CASE = {'Figure 3': H}
case = CASES['Figure 3']
case_figure(case['s'], {'exact': exceed(H, case['s']),
                        **bounds(T4, case['s'], case['t'], case['theta'], case['cap'])});
```

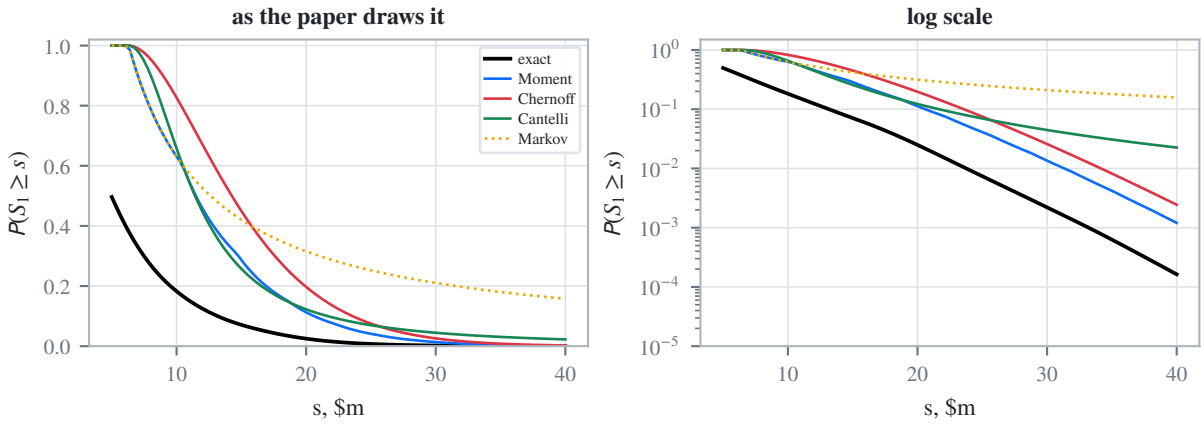


Figure 22.5.: Reproducing Figure 3: fixed losses, $t = 1$, no cap, bounds at $d = -4$, against the exact uncompressed probability. Left as the paper draws it, right on a log scale.

Figure 22.5 shows what the paper's linear axis cannot. The moment and Chernoff curves do approach the exact curve on a linear axis, because all three approach zero. On the log scale they run parallel to it at several times its value, and the gap widens as s grows: Table 22.11, after the four figures, has the moment bound 4.5 times the exact value at \$20m and 7.4 times at \$40m, and the Chernoff bound nearly twice the moment bound. The moment bound's best k is 4 at \$20m and 10 at \$40m, far inside a search that runs to 200, so the gap is a property of the bound and not of where the search stopped. Likewise the optimal v for fixed losses lies inside the authors' range, so their Chernoff curve here is the Chernoff bound.

22.3.6. Gamma event losses: Figure 5

Figure 22.6 makes each event's loss gamma with its tabulated loss as the mean and a coefficient of variation of 0.5. There is no closed form for the exact distribution, but it is easy to compute: the severity is the rate weighted mixture of 1,145 gammas at $d = -4$, each discretized to \$10,000 buckets from its distribution function, and `aggregate` convolves the mixture as it would any other severity. Under a 50% coefficient of variation the \$10,000 rounding of the means is invisible, as Table 22.9 shows by repeating the calculation at $d = -3$, so the table compressed to \$10,000 serves as the exact reference whenever losses are gamma.

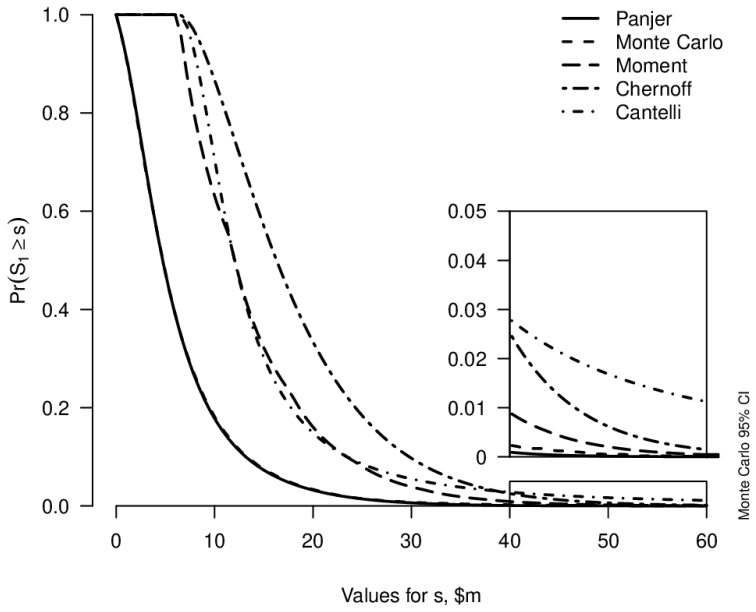


Figure 5: As Figure 3, with $t = 1$, $u = \infty$, $\theta = 0.5$, and $d = -4$. The Markov bound has been dropped. Timings are given in Table V.

Figure 22.6.: Gollini and Rougier (2016), Figure 5.

```
S5 = CASES['Figure 5']['points']
start = time.perf_counter()
G3 = gamma_aggregate(TABLES[-3], 1e4, 15)
G3_SECONDS = time.perf_counter() - start
check = pd.DataFrame({'s, $m': S5 / M, 'd = -4': exceed(G4, S5),
                      'd = -3': exceed(G3, S5)})
check['rel err'] = check['d = -4'] / check['d = -3'] - 1
GT(check.set_index('s, $m'), formatters={'d = -4': '{x:.4e}', 'd = -3': '{x:.4e}'})
```

Table 22.9.: Gamma losses with a coefficient of variation of 0.5: $\mathsf{P}(S_1 \geq s)$ computed on the table compressed to \$10,000 and to \$1,000, with the seconds each took.

s, $m	d = -4	d = -3	rel err
20	3.3568e-02	3.3568e-02	1.25e-05
40	1.4745e-03	1.4743e-03	1.33e-04
60	5.8163e-05	5.8150e-05	2.27e-04

The two compressions in Table 22.9 agree to better than 0.03% at every s; the $d = -4$ build took about a second and the $d = -3$ build a few. Figure 22.7 reproduces Figure 5. The authors drop the Markov bound from here on.

```
EXACT_CASE['Figure 5'] = G4
case = CASES['Figure 5']
case_figure(case['s'], {'exact': exceed(G4, case['s']),
                        **bounds(T4, case['s'], case['t'], case['theta'], case['cap'])});
```

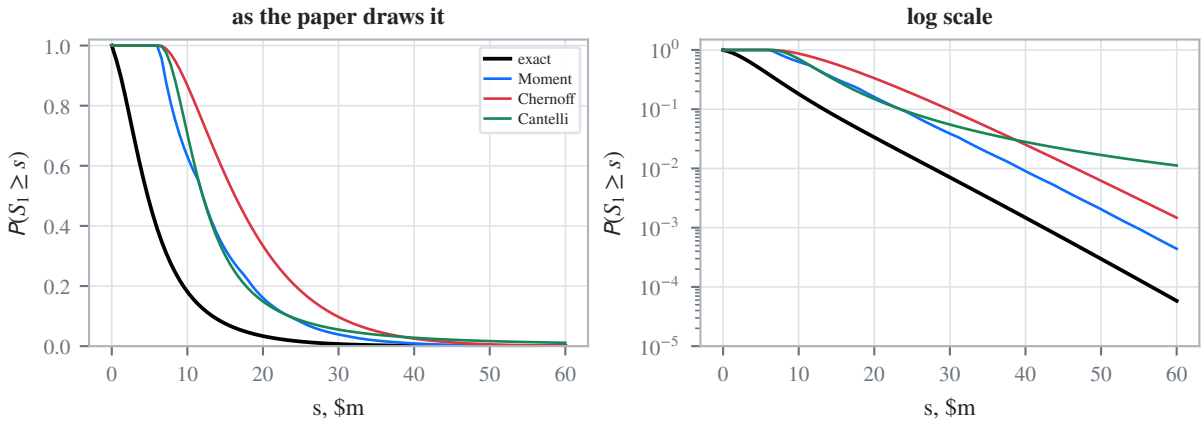


Figure 22.7.: Reproducing Figure 5: gamma losses with a coefficient of variation of 0.5, $t = 1$, no cap, bounds at $d = -4$. Left as the paper draws it, right on a log scale.

In Figure 22.7 the moment bound is between 4.8 and 7.6 times the exact value from \$20m to \$60m. The Chernoff bound is further off than with fixed losses, because the gamma MGF of the largest event is infinite at $v = \beta_i$, so the search must stop at $\min_i \beta_i$, about 0.16 per \$m, which is short of where the bound would like to go.

22.3.7. Capped gamma losses: Figures 6 and 7

Figure 22.8 caps each event's loss at \$5m, and Figure 22.10 runs the same model over ten years. The exact distribution comes from the same construction with the mass above \$5m moved to the bucket at \$5m, and ten years multiply the event rate by ten.

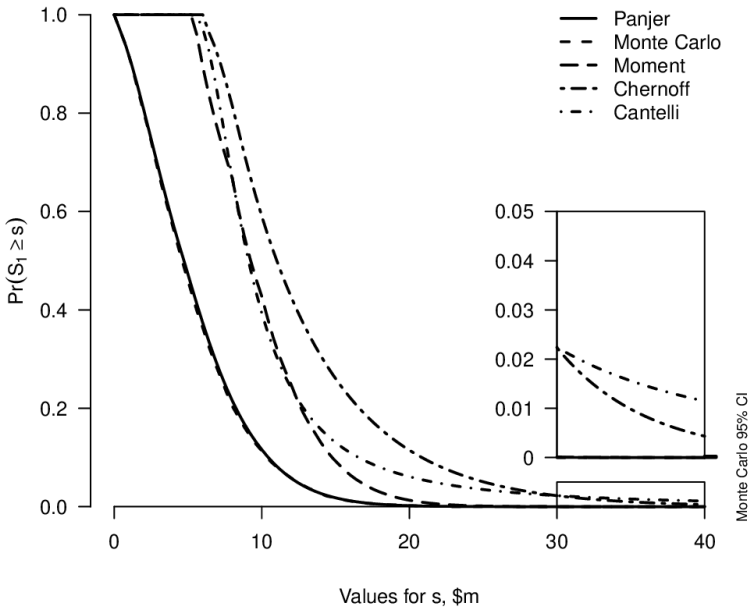


Figure 6: As Figure 3, with $t = 1$, $u = \$5\,\text{m}$, $\theta = 0.5$, and $d = -4$. Timings are given in Table VI.

Figure 22.8.: Gollini and Rougier (2016), Figure 6.

```
V_FREE = np.linspace(0, 1.5, 3001)       # per $m; the capped MGF is finite everywhere
CAPPED_SECONDS = {}
for name in ('Figure 6', 'Figure 7'):
    case = CASES[name]
    start = time.perf_counter()
    EXACT_CASE[name] = gamma_aggregate(T4, 1e4, 15 if case['t'] == 1 else 16,
                                       t=case['t'], cap=case['cap'])
    CAPPED_SECONDS[name] = time.perf_counter() - start


def capped_curves(name):
    """The exact curve, the paper's bounds, and the unrestricted Chernoff bound."""
    case = CASES[name]
    model = dict(t=case['t'], theta=case['theta'], cap=case['cap'])
    return {'exact': exceed(EXACT_CASE[name], case['s']),
            **bounds(T4, case['s'], **model),
            'Chernoff, unrestricted': chernoff(T4, case['s'], V_FREE, interior=True,
                                               **model)}


case_figure(CASES['Figure 6']['s'], capped_curves('Figure 6'), floor=1e-9);
```

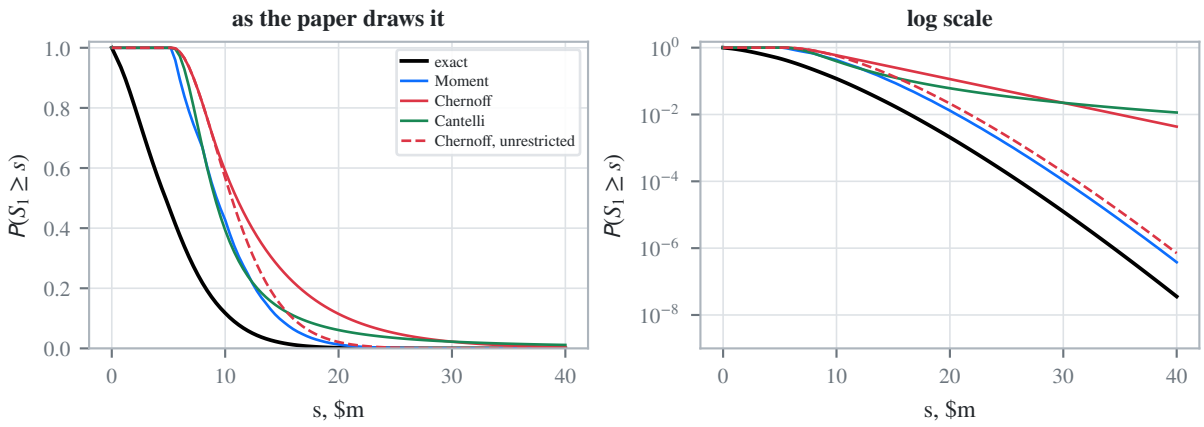


Figure 22.9.: Reproducing Figure 6: gamma losses capped at \$5m, $t = 1$, bounds at $d = -4$, with the Chernoff bound also over an unrestricted range of v. Left as the paper draws it, right on a log scale.

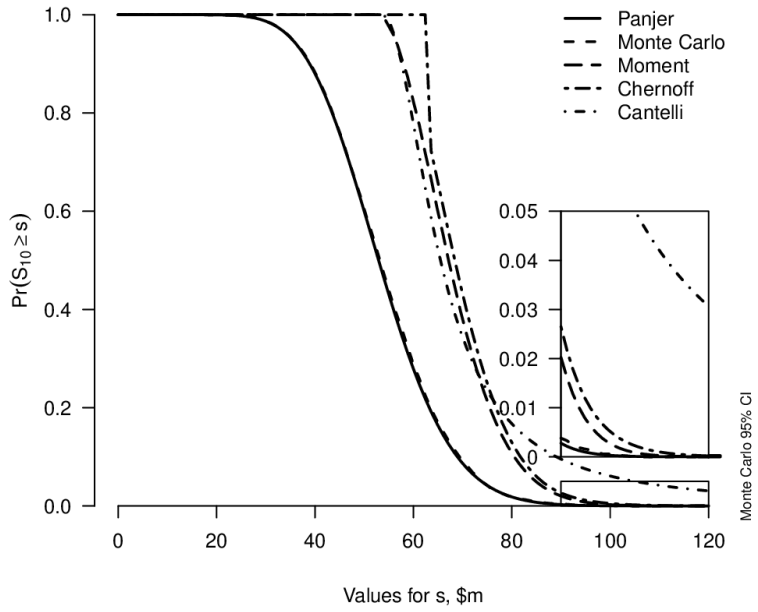


Figure 7: As Figure 3, with $t = 10$, $u = \$5\,\text{m}$, $\theta = 0.5$, and $d = -4$. Timings are given in Table VII.

Figure 22.10.: Gollini and Rougier (2016), Figure 7.

```
case_figure(CASES['Figure 7']['s'], capped_curves('Figure 7'), t=10, floor=1e-7);
```

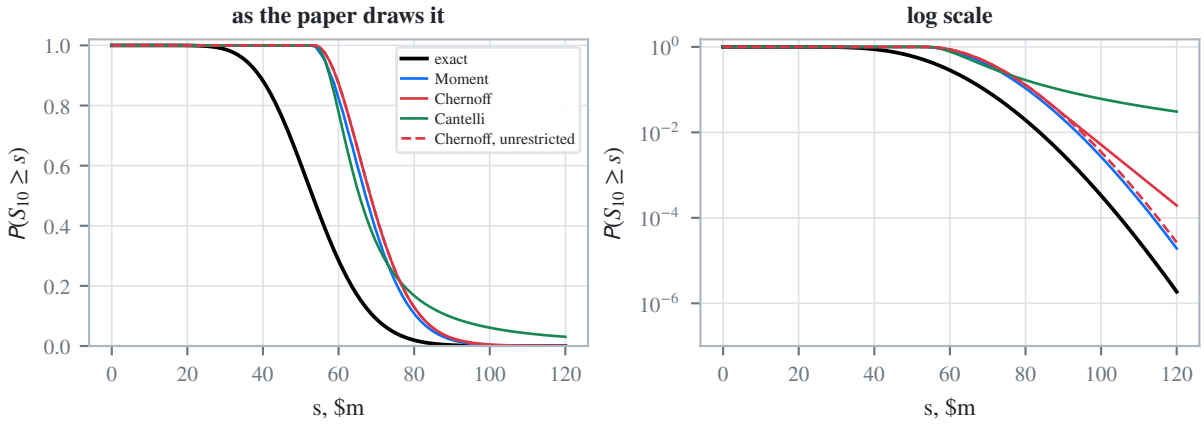


Figure 22.11.: Reproducing Figure 7: gamma losses capped at \$5m, $t = 10$, bounds at $d = -4$, with the Chernoff bound also over an unrestricted range of υ. Left as the paper draws it, right on a log scale.

The cap changes the Chernoff bound. A capped loss is bounded, so its MGF is finite for every υ, but the authors keep the search range of the uncapped gamma, $\upsilon < \min_i \beta_i$. Their Appendix gives that range for the gamma, and it was not revisited for the cap. Figure 22.9 and Figure 22.11 show the Chernoff bound both ways, and Table 22.10 gives the values at three points. At \$40m in one year the bound over the authors' range is 0.0043 and the bound over all υ is 7.1×10^{-7}, against an exact probability of 3.6×10^{-8}: the Chernoff curve in Figure 6 is not the Chernoff bound, and it is looser by a factor of about 6,000. Over all υ the Chernoff bound lies between one and two times the moment bound, which is the order the authors cite from Philips and Nelson (1995): optimized over its full range, the moment bound is never worse than the Chernoff bound.

```
rows = []
for name in ('Figure 6', 'Figure 7'):
    case = CASES[name]
    model = dict(t=case['t'], theta=case['theta'], cap=case['cap'])
    s = case['points']
    rows.append(pd.DataFrame({
        'case': name, 't': case['t'], 's, $m': s / M,
        'exact': exceed(EXACT_CASE[name], s),
        'moment': moment_bound(T4, s, **model),
        'Chernoff, paper': chernoff(T4, s, paper_v(T4, 0.5, case['cap']), **model),
        'Chernoff, all v': chernoff(T4, s, V_FREE, interior=True, **model)}))
chern = pd.concat(rows).set_index(['case', 't', 's, $m'])
GT(chern, float_format='{x:.3e}')
```

Table 22.10.: Capped gamma losses: the Chernoff bound over the authors' range of v and over all v, beside the moment bound and the exact probability.

case	*t*	*s, $m*	exact	moment	Chernoff, paper	Chernoff, all v
Figure 6	1	20	2.071e-03	1.310e-02	1.147e-01	2.118e-02
		30	1.241e-05	1.065e-04	2.229e-02	1.888e-04
		40	3.571e-08	3.788e-07	4.330e-03	7.079e-07
Figure 7	10	60	2.847e-01	8.232e-01	8.736e-01	8.736e-01
		90	2.965e-03	2.024e-02	2.649e-02	2.539e-02
		120	1.861e-06	1.920e-05	1.943e-04	2.659e-05

Table 22.11 collects the four cases at three or four points each, every bound as a multiple of the exact probability. The paper's linear axes show none of these multiples. A factor of 7 at a probability of 0.0002 is a gap of a millimeter on a page whose vertical axis runs from 0 to 1.

```
rows = []
for name, case in CASES.items():
    s = case['points']
    ex = exceed(EXACT_CASE[name], s)
    b = bounds(T4, s, case['t'], case['theta'], case['cap'])
    k = moment_bound(T4, s, case['t'], case['theta'], case['cap'], argmin=True)[1]
    rows.append(pd.DataFrame({'case': name, 's, $m': s / M, 'exact': ex,
                              'Moment': b['Moment'] / ex, 'k': k,
                              'Chernoff': b['Chernoff'] / ex,
                              'Cantelli': b['Cantelli'] / ex}))
GT(pd.concat(rows).set_index(['case', 's, $m']),
   formatters={'exact': '{x:.3e}', 'Moment': 2, 'k': '{x:.0f}', 'Chernoff': 2,
               'Cantelli': 2})
```

Table 22.11.: The exact probability for each of the paper's four cases, each bound as a multiple of it, and the k that minimizes the moment bound. The Chernoff bound is over the authors' range of v.

case	*s*, *$m*	exact	Moment	k	Chernoff	Cantelli
Figure 3	10	1.827e-01	3.45	1	4.53	3.60
	20	2.496e-02	4.51	4	7.89	4.91
	30	2.200e-03	6.15	7	11.73	20.25
	40	1.633e-04	7.39	10	14.94	138.07
Figure 5	20	3.357e-02	4.78	3	9.95	4.43
	40	1.475e-03	6.11	6	17.10	19.00
	60	5.816e-05	7.57	9	25.31	192.99
Figure 6	20	2.071e-03	6.32	9	55.40	29.45
	30	1.241e-05	8.58	16	1,796.11	1,805.06
	40	3.571e-08	10.61	24	121,278.92	320,873.94
Figure 7	60	2.847e-01	2.89	3	3.07	2.73
	90	2.965e-03	6.83	17	8.93	32.19
	120	1.861e-06	10.31	33	104.36	16,469.22

22.3.8. Figure 1 and the Monte Carlo sample size

Figure 22.12 is a design calculation for Monte Carlo. Let $u_{0.95}(x; n)$ be the upper limit of the 95% Jeffreys interval for a probability, which the authors take from Brown et al. (2001), after x exceedances in n draws: the 0.975 quantile of a beta distribution with parameters $x + 1/2$ and $n - x + 1/2$. The figure plots $\mathrm{P}\{u_{0.95}(X; n) \le \kappa_0\}$ for X binomial with n trials and probability $p_0 = \kappa_0/2$, so that it gives the chance that a simulation of size n demonstrates, with 95% confidence, a probability that is in fact half the regulatory limit. The calculation is exact and needs no simulation: for each x, find the smallest n at which $u_{0.95}(x; n) \le \kappa_0$, and sum the binomial probabilities of the x that qualify.

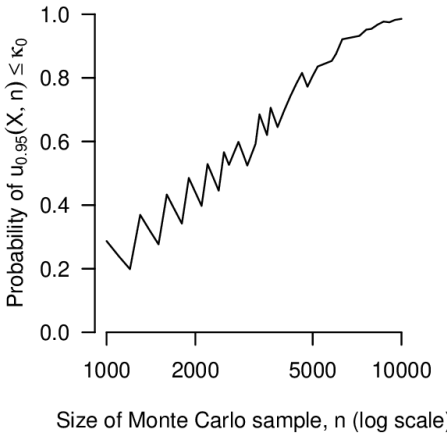


Figure 1: The effect of sample size on Monte Carlo accuracy. The graph shows the probability that the upper bound of the 95% Jeffreys confidence interval for p_0 lies below $\kappa_0 = 0.005$ when $p_0 = \kappa_0/2$.

Figure 22.12.: Gollini and Rougier (2016), Figure 1.

```
def design_probability(ns, p0, kappa=0.005, level=0.95):
    """Pr{u(X; n) <= kappa}, X ~ Binomial(n, p0), at every n in ns.

    u(x; n) falls as n grows, so each x qualifies from a first n onward; the
    largest qualifying x at n is found by search, and the answer is a binomial
    distribution function.
    """
    q = (1 + level) / 2
    first = []
    for x in range(int(ns.max() * kappa) + 2):
        lo, hi = x + 1, 10 * ns.max()
        while lo < hi:
            mid = (lo + hi) // 2
            if stats.beta.ppf(q, x + 0.5, mid - x + 0.5) <= kappa:
                hi = mid
            else:
                lo = mid + 1
        first.append(lo)
    xstar = np.searchsorted(np.array(first), ns, side='right') - 1
    return np.where(xstar >= 0, stats.binom.cdf(xstar, ns, p0), 0.0)


KAPPA = 0.005
NS = np.unique(np.round(np.logspace(3, 5, 2000)).astype(int))
fig, ax = figure('three-quarter')
ax.semilogx(NS, design_probability(NS, KAPPA / 2), lw=0.6, c='k')
ax.axvline(1e4, c='k', lw=0.6, ls=':')
ax.set(xlabel='size of Monte Carlo sample, n',
       ylabel=r'$\mathsf{P}\{u_{0.95}(X; n) \leq \kappa_0\}$', ylim=(0, 1.02));
```

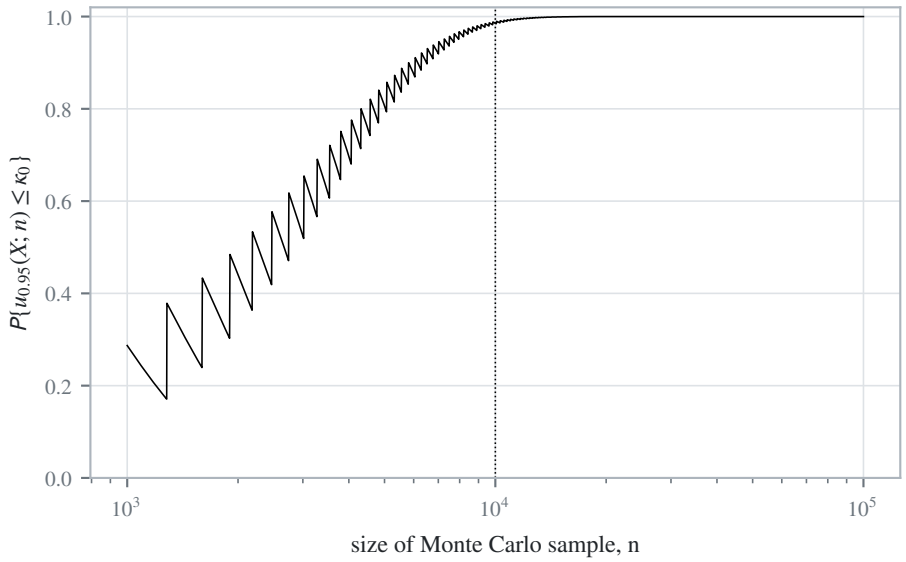


Figure 22.13.: Reproducing Figure 1, at about 2,000 values of n from 1,000 to 100,000, evenly spaced on the log scale: the probability that the upper limit of the 95% Jeffreys interval is at most $\kappa_0 = 0.005$ when $p_0 = \kappa_0/2$. The paper's axis stops at the dotted line.

Figure 22.13 matches the paper's curve, 0.29 at $n = 1{,}000$ and 0.986 at $n = 10{,}000$, and the saw teeth are the discreteness of X. Beyond 10^4 the probability is within 10^{-4} of one by $n = 20{,}000$. The paper's figure therefore shows that 10^4 draws already suffice, which is also the conclusion of the authors' summary, "10,000 Monte Carlo simulations should suffice". Beside the figure, though, they

write that "$n = 10^5$ seems to be a good choice", and in Section 6.2 they report using 10^5. Their script calls the simulation with the package default, `nsim = 10e3`, which is 10^4, so the figures were drawn from 10^4 draws, not the 10^5 reported in the text.

The design is for a probability of $\kappa_0/2$. At the point where the figures draw their Monte Carlo interval, $s_0 = \$40\text{m}$, the probability is far smaller, and Table 22.12 shows what that means for the interval.

```
P40 = EXACT[-1]
mc = []
for n in (10 ** 3, 10 ** 4, 10 ** 5):
    x = round(n * P40)
    lo, hi = stats.beta.ppf([0.025, 0.975], x + 0.5, n - x + 0.5)
    mc.append({'n': n, 'design probability': design_probability(np.array([n]), KAPPA / 2)[0],
               'expected exceedances': n * P40, 'x': x,
               'Jeffreys lower': lo if x > 0 else 0.0, 'Jeffreys upper': hi,
               'exact': P40})
GT(pd.DataFrame(mc).set_index('n'),
   formatters={'n': '{x:,.0f}', 'design probability': 4, 'expected exceedances': 2,
               'x': '{x:,.0f}', 'Jeffreys lower': '{x:.2e}', 'Jeffreys upper': '{x:.2e}',
               'exact': '{x:.2e}'})
```

Table 22.12.: Monte Carlo at \$40m with fixed losses: the expected number of exceedances, and the 95% Jeffreys interval at that count, against the exact probability, with the design probability of Figure 1.

n	design probability	expected exceedances	x	Jeffreys lower	Jeffreys upper	exact
1,000	0.2869	0.16	0	0.00e+00	2.51e-03	1.63e-04
10,000	0.9856	1.63	2	4.16e-05	6.41e-04	1.63e-04
100,000	1.0000	16.33	16	9.52e-05	2.54e-04	1.63e-04

With 10^4 draws the expected number of exceedances at \$40m is 1.6, and the interval at two exceedances in Table 22.12 spans a factor of fifteen. The small count is no criticism of the design, which addresses a different question, but it does mean that the right end of the Monte Carlo curves in Figures 3, 5, 6 and 7 rests on a handful of draws.

22.3.9. Timings

Tables IV to VII, Figure 22.14 to Figure 22.17, give the time each method took on each compression for Figures 3, 5, 6 and 7. The Panjer recursion is the only method whose time grows quickly with the table, and the authors stopped it at $d = -2$ for Figure 3 and at $d = -3$ for the others.

Table IV: Timings for the methods shown in Figure 3, in seconds on a standard desktop computer, for different degrees of rounding (see section 6.1).

	$d=-4$	$d=-3$	$d=-2$	$d=-1$	$d=0$
Panjer	0.461	40.784	4651.298	NA	NA
MonteCarlo	1.246	2.085	5.820	10.228	12.413
Moment	0.011	0.006	0.010	0.019	0.025
Chernoff	0.112	0.310	0.634	1.017	1.284
Cantelli	0.001	0.002	0.001	0.002	0.002
Markov	0.001	0.001	0.001	0.005	0.001

Figure 22.14.: Gollini and Rougier (2016), Table IV.

Table V: Timings for the methods shown in Figure 5.

	$d=-4$	$d=-3$	$d=-2$	$d=-1$	$d=0$
Panjer	1.509	121.062	NA	NA	NA
MonteCarlo	0.921	1.961	5.385	9.967	12.197
Moment	0.006	0.021	0.055	0.100	0.118
Chernoff	0.127	0.614	1.670	2.734	3.333
Cantelli	0.001	0.002	0.007	0.022	0.019

Figure 22.15.: Gollini and Rougier (2016), Table V.

Table VI: Timings for the methods shown in Figure 6.

	$d=-4$	$d=-3$	$d=-2$	$d=-1$	$d=0$
Panjer	0.275	11.950	NA	NA	NA
MonteCarlo	1.106	2.099	5.625	10.329	11.837
Moment	0.016	0.070	0.210	0.355	0.431
Chernoff	0.508	2.057	5.659	10.853	14.992
Cantelli	0.003	0.006	0.019	0.032	0.041

Figure 22.16.: Gollini and Rougier (2016), Table VI.

Table VII: Timings for the methods shown in Figure 7.

	$d=-4$	$d=-3$	$d=-2$	$d=-1$	$d=0$
Panjer	0.587	46.000	NA	NA	NA
MonteCarlo	918.101	2.312	7.189	11.834	14.869
Moment	0.027	0.141	0.435	0.592	0.736
Chernoff	3.989	2.101	7.794	10.503	18.314
Cantelli	0.002	0.007	0.020	0.164	0.046

Figure 22.17.: Gollini and Rougier (2016), Table VII.

Table 22.13 sets the paper's Panjer times beside the time `aggregate` took for each exact reference in this chapter, measured when the book was rendered: the full table at a $10 bucket for fixed losses, and the gamma mixture at $d = -4$ for the others, including the construction of the severity. The seconds are on different machines a decade apart, so the ratios matter less than the structure. The FFT computes a compound distribution with two transforms whatever the size of the table, and the size of the table enters only through the severity, once.

```
GT(pd.DataFrame([
    ('Figure 3', 0.461, -2, 4651.298, 'full table, $10 bucket', H_SECONDS),
    ('Figure 5', 1.509, -3, 121.062, 'd = -4, gamma, $10,000 bucket', G4_SECONDS),
    ('Figure 6', 0.275, -3, 11.950, 'd = -4, capped gamma', CAPPED_SECONDS['Figure 6']),
    ('Figure 7', 0.587, -3, 46.000, 'd = -4, capped gamma, ten years',
     CAPPED_SECONDS['Figure 7']),
], columns=['case', 'Panjer, d = -4', 'finest d', 'Panjer, finest d', 'aggregate model',
            'aggregate seconds']).set_index('case'),
   formatters={'Panjer, d = -4': 3, 'finest d': '{x:.0f}', 'Panjer, finest d': '{x:,.3f}',
               'aggregate seconds': 3})
```

Table 22.13.: The paper's Panjer recursion at $d = -4$ and at its finest compression, against `aggregate` building each exact reference in this chapter, in seconds.

case	**Panjer, d = -4**	**finest d**	**Panjer, finest d**	**aggregate model**	**aggregate seconds**
Figure 3	0.461	-2	4,651.298	full table, $10 bucket	0.794
Figure 5	1.509	-3	121.062	d = -4, gamma, $10,000 bucket	0.873
Figure 6	0.275	-3	11.950	d = -4, capped gamma	0.034
Figure 7	0.587	-3	46.000	d = -4, capped gamma, ten years	0.035

22.4. Comments

22.4.1. Reach of the exact computation

The authors' case for bounds rests on exact methods being too slow without compression. On the evidence of Table 22.6 and Table 22.13 that premise no longer holds, and it did not hold in 2015 either: the FFT was as available then as now, and the reason it is fast is the structure of the compound Poisson, not the speed of the machine. The uncompressed table takes a fraction of a second at a $100 bucket, and the gamma version takes about a second. The bounds keep a use where only the moments of a table are known, or where a bound is needed in closed form inside some larger calculation, but as a substitute for computing the distribution of a table that is in hand they are not needed.

22.4.2. Looseness of the bounds

The exact 1-in-200 annual loss with fixed event losses is $26.6m. Table 22.14 gives, for each bound, the smallest capital s at which the bound falls to 0.005, which is the solvency question asked of the bound directly. On the full table the moment bound calls for 29% more capital than the exact answer, Chernoff for 39% more, and Cantelli for nearly three times as much. A regulator satisfied by a bound is satisfied at a cost, and the exact answer costs a second.

```
def chernoff_fixed(table, s, t=1):
    """The Chernoff bound for fixed losses at its exact optimum, and the optimal v.

    With fixed losses the MGF is finite everywhere and the optimum solves
    lambda t P[Y e^{vY}] = s, monotone in v.
    """
    x = table.Dollars.to_numpy() / M
    rate = table.Rate.to_numpy()
    out = []
    for s_m in np.atleast_1d(np.asarray(s, float)) / M:
        slope = lambda v: t * (rate * x * np.exp(v * x)).sum() - s_m
        if slope(0) >= 0:
            out.append((1.0, 0.0))
            continue
        hi = 1.0
        while slope(hi) < 0:
            hi *= 2
        v = optimize.brentq(slope, 0, hi, xtol=1e-14)
        out.append((np.exp(-v * s_m + t * (rate @ np.exp(v * x) - rate.sum())), v))
    return np.array(out).T
```

```
def quantile(a, p):
    """The p quantile of the distribution on the grid of `a`."""
    return a.xs[np.searchsorted(np.cumsum(a.agg_density), p)]


FULL = TABLES[0]
VAR = quantile(H, 0.995)
capital = {'exact': VAR}
for name, fn in (('moment', lambda s: moment_bound(FULL, [s])[0]),
                 ('Chernoff', lambda s: chernoff_fixed(FULL, [s])[0][0]),
                 ('Cantelli', lambda s: cantelli(FULL, [s])[0]),
                 ('Markov', lambda s: markov(FULL, [s])[0])):
    capital[name] = optimize.brentq(lambda s: fn(s) - 0.005, 10 * M, 1e4 * M, xtol=1e3)
GT(pd.DataFrame({'capital, $m': pd.Series(capital) / M,
                 'multiple of exact': pd.Series(capital) / VAR})
   .rename_axis('method'), formatters={'capital, $m': '{x:,.2f}', 'multiple of exact': 2})
```

Table 22.14.: The capital each method requires for a 1-in-200 annual probability, fixed losses, full table, in millions of dollars and as a multiple of the exact value.

method	capital, $m	multiple of exact
exact	26.62	1.00
moment	34.25	1.29
Chernoff	37.07	1.39
Cantelli	78.49	2.95
Markov	1,261.88	47.41

22.4.3. Large deviations and the Chernoff bound

In Section 3 the authors argue from large deviations that "as t becomes large the Chernoff upper bound becomes exact", and that, the moment bound being tighter, "both bounds are converging from above on the actual probability". In Section 6.1 they rely on the argument to judge compression with the moment bound. The large deviations limit is a statement about the exponent: for a fixed loss per year a above the mean, $t^{-1}\log \mathsf{P}(S_t \geq at) \to -I(a)$, and the Chernoff bound is $e^{-tI(a)}$ exactly. The $o(t)$ term in the authors' display allows any factor that grows more slowly than an exponential. By the theorem of Bahadur and Rao (1960) the factor is

$$\frac{e^{-tI(a)}}{\mathsf{P}(S_t \geq at)} \sim \upsilon_a \sigma_a \sqrt{2\pi t},$$

where υ_a is the optimal υ and $\sigma_a^2 = \lambda \mathsf{P}(Y^2 e^{\upsilon_a Y})$ is the annual variance under the tilted measure. The ratio of the bound to the probability grows like $\sqrt{t}$; the bound is exact on a logarithmic scale and not otherwise. Table 22.15 shows the ratio at \$15m a year on the full table with fixed losses, and the ratio follows the prediction ever more closely as t grows. The moment bound's ratio grows too.

```
A_RATE = 15 * M
LOG2_BY_T = {1: 17, 2: 18, 4: 18, 8: 19, 16: 20}
x_m, rate = FULL.Dollars.to_numpy() / M, FULL.Rate.to_numpy()
ld = []
for t, log2 in LOG2_BY_T.items():
    s = A_RATE * t
```

```
    ex = exceed(elt_aggregate(ELT.Loss, ELT.Rate, 1e3, log2, t=t), s)[()]
    (ch,), (v,) = chernoff_fixed(FULL, [s], t)
    sigma = np.sqrt(rate @ (x_m ** 2 * np.exp(v * x_m)))
    ld.append({'t': t, 's, $m': s / M, 'exact': ex, 'Chernoff multiple': ch / ex,
               'predicted': v * sigma * np.sqrt(2 * np.pi * t),
               'moment multiple': moment_bound(FULL, [s], t=t)[0] / ex})
GT(pd.DataFrame(ld).set_index('t'),
   formatters={'s, $m': '{x:,.0f}', 'exact': '{x:.3e}', 'Chernoff multiple': 2,
               'predicted': 2, 'moment multiple': 2})
```

Table 22.15.: Fixed losses, full table, at a loss of $15m a year over t years: the exact probability, each bound as a multiple of it, and the Bahadur and Rao prediction for the Chernoff multiple.

t	s, $m	exact	Chernoff multiple	predicted	moment multiple
1	15	7.093e-02	6.34	4.04	4.04
2	30	2.694e-02	7.51	5.72	4.91
4	60	4.324e-03	9.47	8.09	6.31
8	120	1.343e-04	12.48	11.44	8.38
16	240	1.657e-07	16.95	16.18	11.44

Holding the probability fixed instead of the loss per year gives the same answer by another route. At the 99.5% point of S_t the central limit theorem takes over as t grows, the Chernoff bound tends to $e^{-z^2/2}$ with z the normal 99.5% point, and the ratio to 0.005 tends to 7.25, not to 1. Table 22.16 shows the ratio at one year and at ten on its way there.

```
z = stats.norm.ppf(0.995)
fl = []
for t, log2 in ((1, 17), (10, 18)):
    a = elt_aggregate(ELT.Loss, ELT.Rate, 1e3, log2, t=t)
    q = quantile(a, 0.995)
    ex = exceed(a, q)[()]
    fl.append({'t': str(t), 'VaR 99.5%, $m': q / M, 'exact': ex,
               'Chernoff multiple': chernoff_fixed(FULL, [q], t)[0][0] / ex,
               'moment multiple': moment_bound(FULL, [q], t=t)[0] / ex})
fl.append({'t': 'normal limit', 'VaR 99.5%, $m': np.nan, 'exact': 0.005,
           'Chernoff multiple': np.exp(-z * z / 2) / 0.005, 'moment multiple': np.nan})
GT(pd.DataFrame(fl).set_index('t'),
   formatters={'VaR 99.5%, $m': '{x:,.2f}', 'exact': 5, 'Chernoff multiple': 2,
               'moment multiple': 2})
```

Table 22.16.: Fixed losses, full table, at the 99.5% point of S_t: each bound as a multiple of the exact probability, at one and ten years and in the normal limit.

t	VaR 99.5%, $m	exact	Chernoff multiple	moment multiple
1	26.62	0.00500	10.73	5.77
10	112.45	0.00500	8.66	6.55
normal limit	—	0.00500	7.25	—

Neither limit is 1. The squeeze the authors describe does not occur, and a bound that is several

times the probability at every horizon cannot serve as the yardstick for a compression error of a few percent, which is what Table 22.8 found.

22.4.4. The Monte Carlo sample size

The authors give two sample sizes. In Section 4 and Section 6.2 they give 10^5. Figure 1 supports 10^4, the summary in Section 7 recommends 10,000, and the package default, from which the figures were drawn, is 10^4. On the authors' own design calculation, Figure 22.13, 10^4 is the defensible choice, with a probability of 0.986, and 10^5 is ample. The inconsistency is in what the text reports: a sample ten times the one the figures were computed from.

22.4.5. The Monte Carlo time in Table VII

In Figure 22.17 Monte Carlo takes 918.101 seconds at $d = -4$ and 2.312 seconds at $d = -3$. Every other Monte Carlo time in the four tables is between 0.9 and 15 seconds and grows with the size of the table, as it should, since a simulation draws from the table. The value at $d = -4$ is out of line by a factor of several hundred and is most likely an interruption during the run rather than a property of the method. A timing cannot be reproduced, so the point stays an observation.

22.4.6. The capped gamma and the DecL limit

The authors prefer a capped gamma to a beta scaled to $[0, u]$ because the cap leaves an atom at u, which is what a maximum insured loss does, while a scaled beta has none. A DecL limit, `5 xs 0` in the `Hurricane.Capped` program, places exactly that atom on the severity, and `aggregate` applies it to any severity, the gamma mixture included. The cap also bounds the loss, which is what makes the MGF finite everywhere and the authors' Chernoff range too narrow, as Table 22.10 showed.

22.4.7. Long vectors in DecL

A DecL vector of 450 elements parses, and one of 500 exceeds Python's recursion limit inside the parser, as Table 22.17 shows for the version of `aggregate` named in the preface. The full table, 32,060 rows, and even the table at $d = -4$, 1,145 rows, are beyond that limit, which is why this chapter builds them through the `Aggregate` class. The division is a reasonable one in any case: a DecL program is a model a person writes and reads, and a table of thirty thousand rows is data, which belongs in a file and in the class, which accepts vectors of any length.

```
def parses(n):
    """Whether a dsev of n equally likely losses parses.

    The library's own build is used, not the book's recording one, so that the
    test programs stay out of the appendix.
    """
    vector = ' '.join(str(i + 1) for i in range(n))
    try:
        library_build(f'agg T 1 claim dsev [{vector}] fixed', update=False)
        return 'parses'
    except RecursionError:
        return 'RecursionError'
```

```
GT(pd.DataFrame({'n': [450, 500], 'result': [parses(450), parses(500)]}).set_index('n'),
   formatters={'n': '{x:,.0f}'})
```

Table 22.17.: Parsing a `dsev` of n equally likely losses, on the version of `aggregate` named in the preface.

n	result
450	parses
500	RecursionError

23. Wilson and Keich on accurate FFT convolution

Reproducing Wilson and Keich (2016)

23.1. Introduction

Wilson and Keich (2016) asks a narrow question with a wide answer: given two non-negative vectors, can you compute their convolution by FFT and *guarantee* the relative accuracy of every entry?

The answer is yes, and the algorithm is aFFT-C. It matters here because `aggregate` computes compound distributions by exactly the FFT convolution these authors show is unreliable, and because the companion paper Ben Rached et al. (2024) reaches the same conclusion by a different route and recommends abandoning the FFT entirely. This paper does not. It keeps the FFT and fixes it, which is the better outcome for a library that needs $O(N \log N)$.

The mechanism has three parts:

1. **A computable error bound.** For each output index, a bound on the FFT roundoff error that can be evaluated at runtime. This is the part that matters most, because it converts a silent failure into a detected one.
2. **PITS (partitioning into thin slices).** Decompose each input pmf into components of reduced dynamic range, so that FFT-C applied to each pair is accurate, then reassemble. Roundoff cannot swamp small values if the small values are never in the same vector as the large ones.
3. **An exponential shift.** Precondition by $p_\theta(k) = p(k)e^{k\theta}/M_p(\theta)$, which commutes with convolution, so it can be peeled off afterward. Its role here is to reduce the number of slices PITS needs, and hence the number of FFT calls.

23.2. What is reproducible

Table / result	Content	Reproducible
Section 1 toy example	A four-element pmf with exact rational convolution, and the measured relative errors of NC, FFT-C and aFFT-C	Yes, and now exactly: the reference column is library-computed rational arithmetic
Table 1	Number of PITS splits with and without the exponential shift	No, internal to aFFT-C

Table / result	Content	Reproducible
Table 2	Median and maximum relative error of aFFT-C and FFT-C over six declared pmf classes	The FFT-C columns, statistically
Figures 4 and 5	Timing of aFFT-C against naive convolution and FFT-C	No, implementation timing

The aFFT-C columns of Table 2 are not reproducible without implementing aFFT-C. The FFT-C columns are, because FFT-C is what `aggregate` already does.

23.3. The Section 1 example

This is the whole paper in seven numbers. Take $p = (0, a, b, c)$ with $b = 10^{-5}$, $c = 10^{-20}$ and $a = 1 - b - c$. Then

$$p * p = (0,\ 0,\ a^2,\ 2ab,\ 2ac + b^2,\ 2bc,\ c^2),$$

which is exact and needs no numerics at all. The last two entries are 2×10^{-25} and 10^{-40}, both far below double precision machine epsilon relative to the leading entry of 1.

Figure 23.1 is the paper's own illustration, and Table 23.1 recomputes it three ways.

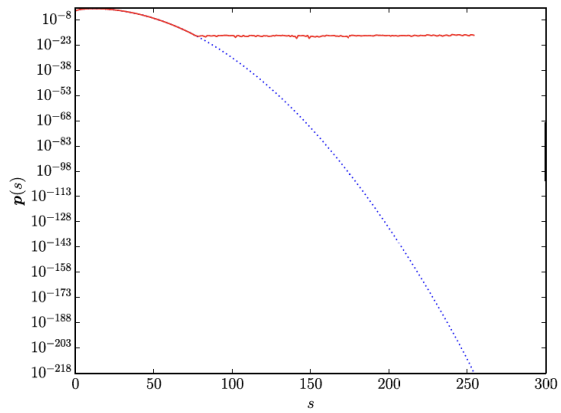


Fig. 1. The plot displays the FFT-C calculated values (3) of the pmf $\boldsymbol{p} * \boldsymbol{p}$ (solid line), as well as the values accurately computed via naive-convolution (dotted line). Here $\boldsymbol{p}(s) = A \exp\left(\frac{1}{60} s(10 - s)\right)$ for $s = \{0, 1, \ldots, 127\}$, where A guarantees $\boldsymbol{p}$ is a pmf, that is, $\|\boldsymbol{p}\|_1 = 1$. The p-value of $s_0 = 200$, that is, $\sum_{s \geq s_0} (\boldsymbol{p} * \boldsymbol{p})(s)$ is computed to be $7.0 \cdot 10^{-16}$ when the convolution is performed via FFT-C, which is more than one hundred orders of magnitude larger than the true value of $1.7 \cdot 10^{-132}$.

Figure 23.1.: Wilson and Keich (2016), Figure 1, illustrating the Section 1 example.

```
%xmode minimal
import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')
```

```
b, c = 1e-5, 1e-20
a = 1 - b - c
exact = np.array([0.0, 0.0, a * a, 2 * a * b, 2 * a * c + b * b, 2 * b * c, c * c])

naive = np.convolve([0, a, b, c], [0, a, b, c])                # NC
fftc = np.fft.irfft(np.fft.rfft([0, a, b, c], 8) ** 2, 8)[:7]   # FFT-C

# the same convolution through aggregate: a three-point severity, two claims
obj = build(f'agg Toy dfreq [2] dsev [1 2 3] [{a!r} {b!r} {c!r}]',
            log2=6, bs=1, normalize=False)
agg = np.array([obj.agg_density[k] for k in range(7)])

def rel(x):
    return np.where(exact > 0, np.abs(x - exact) / np.where(exact > 0, exact, 1.0), 0.0)

GT(pd.DataFrame({'k': range(7), 'exact': exact, 'NC': naive, 'FFT-C': fftc,
                 'agg': agg, 'rel NC': rel(naive), 'rel FFT-C': rel(fftc),
                 'rel agg': rel(agg)}))
```

```
Exception reporting mode: Minimal
```

Table 23.1.: The seven entry convolution exactly, by naive convolution, by textbook FFT-C and through the library, with relative errors.

k	exact	NC	FFT-C	agg	rel NC	rel FFT-C	rel agg
0	0.000e+00	0.000e+00	1.832e-18	-12.029e-18	0	0.000e+00	0.000e+00
1	0.000e+00	0.000e+00	-7.266e-18	-11.396e-18	0	0.000e+00	0.000e+00
2	999.980e-03	999.980e-03	999.980e-03	999.980e-03	0	0.000e+00	111.025e-18
3	20.000e-06	20.000e-06	20.000e-06	20.000e-06	0	3.036e-12	1.090e-12
4	100.000e-12	100.000e-12	100.000e-12	100.000e-12	0	64.219e-09	752.951e-09
5	200.000e-27	200.000e-27	-46.519e-18	9.233e-18	0	232.595e+06	46.165e+06
6	100.000e-42	100.000e-42	-55.511e-18	-6.419e-18	0	555.112e+21	64.192e+21

Three things to read off Table 23.1.

Naive convolution is exact here. The paper reports $r_{NC,64} = (0, 0, 2{\cdot}10^{-16}, 10^{-20}, 2{\cdot}10^{-16}, 10^{-16}, 3{\cdot}10^{-17})$, and this run gives zero throughout because the individual products happen to be exactly representable. Either way the point stands: NC preserves relative accuracy at every entry, because a sum of non-negative products has no cancellation.

FFT-C destroys the last two entries. Index 5 has a relative error around 10^{8} and index 6 around 10^{23}. Both computed values are *negative*, which is impossible for a convolution of non-negative vectors.

`aggregate` behaves exactly like textbook FFT-C. Same failure at the same two indices, same orders of magnitude of error. The signs differ, agg returning a positive value at index 5 where FFT-C returns a negative one, because the sign of accumulated rounding is arbitrary and the two routines pad and transform slightly differently. That is the confirmation that agg's engine is FFT-C and inherits its properties without modification, and also a warning that a negative-value check catches only some of these failures.

Note also index 4, where the true value is 10^{-10}: FFT-C is already wrong in the seventh significant figure and agg in the sixth. The degradation is gradual on the way down to the cliff, not sudden.

For contrast the paper reports what aFFT-C achieves on this example:

$$r_{\text{aFFT-C},10^9} = (0, 0, 8 \cdot 10^{-15}, 4 \cdot 10^{-12}, 2 \cdot 10^{-15}, 4 \cdot 10^{-12}, 1 \cdot 10^{-14})$$

Every entry to eleven significant figures or better, including the 10^{-40} one, at $O(N \log N)$ cost.

23.4. The Section 1 example, exactly

Since this chapter was written the library has grown an exact engine. The DecL kind `nttagg` computes a bounded discrete compound with no floating point at all: input probabilities are parsed text-exactly as rational numbers, the convolution runs over 32-bit prime fields via the number-theoretic transform, and the answer is reconstructed by the Chinese remainder theorem as exact integer numerators over one common denominator. The float FFT pipeline still runs beside it, driven from the same spec, so one object carries both the textbook computation and the exact answer it should have produced.

That changes the status of this example. The exact column of Table 23.1 was hand algebra, trustworthy because the convolution has seven entries. It is now library output, and the hand algebra becomes the check on the library rather than the reference for it.

```
from fractions import Fraction

b = Fraction(1, 10**5)
c = Fraction(1, 10**20)
a = 1 - b - c

wil = build(f'nttagg Wilson dfreq [2] dsev [1 2 3] [{a} {b} {c}]')

closed_form = [a * a, 2 * a * b, 2 * a * c + b * b, 2 * b * c, c * c]
nonzero = wil.p_exact[wil.p_exact != 0]
assert list(nonzero) == closed_form, 'library p_exact differs from the closed form'
print('p_exact == [a^2, 2ab, 2ac + b^2, 2bc, c^2] as exact Fractions: True')

GT(wil.density_df.query('p_exact_float > 0')[['p_total', 'p_exact_float', 'p_exact_log10']])
```

```
p_exact == [a^2, 2ab, 2ac + b^2, 2bc, c^2] as exact Fractions: True
```

Table 23.2.: The Section 1 example computed exactly by the library. The exact column matches the closed form identically, as Fractions, and `p_exact_log10` resolves the entries that are invisible to a double.

loss	p_total	p_exact_float	p_exact_log10
2	999.980e-03	999.980e-03	-8.686e-06
3	20.000e-06	20.000e-06	-4.699e+00
4	100.000e-12	100.000e-12	-10.000e+00
5	0.000e+00	200.000e-27	-24.699e+00
6	0.000e+00	100.000e-42	-40.000e+00

Two readings of Table 23.2. First, the `assert` passed: the five nonzero probabilities the engine reconstructs equal $(a^2,\ 2ab,\ 2ac + b^2,\ 2bc,\ c^2)$ as exact rational numbers, not to any tolerance. Second, `p_total`, the float pipeline's answer inside the same object, reads exactly zero at losses 5 and 6, where the truth is 2×10^{-25} and 10^{-40}. The float pipeline does not merely get these entries wrong; after cleanup it reports them as structurally absent.

The paper's central contribution is a runtime certificate: a computable bound that tells the caller which entries of an FFT convolution are trustworthy. On this envelope, nonnegative integer outcomes with bounded support and rational probabilities, the library now does something categorically stronger. It does not bound the error of the float pipeline; it computes it, per bucket, exactly before the final rounding. Table 23.3 is the paper's Figure 1 regenerated from inside the library.

```
GT(wil.accuracy_df.loc[2:6])
```

Table 23.3.: The per-bucket error of the float FFT pipeline against the exact answer, computed exactly. `p_fft` is the raw signed FFT output before cleanup, and the relative errors at losses 5 and 6 are the catastrophe the paper exhibits.

loss	p_exact_float	p_fft	err	abs_err	rel_err
2	999.980e-03	999.980e-03	9.93e-17	9.93e-17	9.93e-17
3	20.000e-06	20.000e-06	-4.30e-17	4.30e-17	-2.15e-12
4	100.000e-12	100.000e-12	-2.04e-17	2.04e-17	-2.04e-07
5	200.000e-27	18.316e-18	1.83e-17	1.83e-17	9.16e+07
6	100.000e-42	27.756e-18	2.78e-17	2.78e-17	2.78e+23

The relative errors at losses 5 and 6 in Table 23.3 are the failures of Table 23.1, now library-reported rather than hand-measured, and `p_fft` preserves the raw signed noise that `p_total` clips, so the arbitrary sign discussed under Table 23.1 is visible too.

23.4.1. Adjudicating the naive convolution claim

The discussion under Table 23.1 noted a small discrepancy with the paper: the paper reports nonzero relative errors for naive convolution, around 10^{-16}, while our run measured zero throughout. With an exact yardstick the discrepancy resolves, and both observations turn out to be right.

```
af, bf, cf = float(a), float(b), float(c)
naive_f = np.convolve([0, af, bf, cf], [0, af, bf, cf])
float_form = [af * af, 2 * af * bf, 2 * af * cf + bf * bf, 2 * bf * cf, cf * cf]

rows = []
for k in range(2, 7):
    e = closed_form[k - 2]
    rows.append({'k': k, 'NC': naive_f[k],
                 'rel err vs float closed form':
                     float((Fraction(naive_f[k]) - Fraction(float_form[k - 2])) / e),
                 'rel err vs exact':
                     float((Fraction(naive_f[k]) - e) / e)})
GT(pd.DataFrame(rows))
```

Table 23.4.: Naive convolution against two yardsticks. Against the float evaluation of the closed form it is bit-identical; against the exact rational answer it carries the input representation error, which is what the paper's r_NC reports.

k	NC	rel err vs float closed form	rel err vs exact
2	999.980e-03	0	9.93e-17
3	20.000e-06	0	5.99e-17
4	100.000e-12	0	2.21e-16
5	200.000e-27	0	3.85e-17
6	100.000e-42	0	-7.07e-17

In Table 23.4 the middle column is identically zero: the doubles naive convolution returns are bit-for-bit the float evaluation of the closed form, so the arithmetic commits no error at all, which is the no-cancellation property the paper proves. The right column sits at 10^{-16} to 10^{-17}, and every bit of it is the cost of writing a, b, c as doubles in the first place, before any arithmetic runs. The paper's $r_{NC,64} = (0, 0, 2\cdot10^{-16}, 10^{-20}, 2\cdot10^{-16}, 10^{-16}, 3\cdot10^{-17})$ is the same quantity measured on its authors' machine. Our earlier "zero throughout" was an artifact of a yardstick that was itself computed in doubles and therefore shared the inputs' rounding.

23.5. Table 2: the FFT-C columns

Table 2, Figure 23.2, measures relative error over six pmf classes, which the paper defines precisely. With $a, b, c \sim U(0,1)$ fixed per pmf, $u_i \sim U(0,1)$ per entry, and A normalizing:

1. constant, $p(i) \equiv A$
2. random, $p(i) = A\exp(-40u_i)$
3. quadratic, $p(i) = A\exp(-30(a+1)x_i^2 + 20(2b-1)x_i + 20(2c-1))$, x even on $[0,1]$
4. sinusoid, $p(i) = A\exp(10(3a+1)\sin(x_i + b/10) + 10(5c-4)x_i)$, x even on $[0, 3\pi]$
5. multi-scaled I, a random fifth are $A\exp(-30u_i)$, the rest $A\exp(-100(u_i+1))$
6. multi-scaled II, a random third are $A\exp(-15(2a+1)u_i)$, the rest $A\exp(-50((2a+1)u_i+2b+1))$

Table 2
The maximum and median relative errors of the maximum error of the elements of $\tilde{\boldsymbol{q}}^{(k,l)} = \boldsymbol{p}^{(k)} * \boldsymbol{p}^{(l)}$ as computed by aFFT-C and FFT-C where the $\boldsymbol{p}^{(*)}$ are selected from the classes of the pmfs described in Section 5 and L is their length. That is, the third column is $\max_{k,l} \max_s \mathrm{rel}(\tilde{\boldsymbol{q}}^{(k,l)}_{\text{aFFT-C}}(s), \tilde{\boldsymbol{q}}^{(k,l)}_{\text{NC}}(s))$ where $\tilde{\boldsymbol{q}}^{(k,l)}_x(s)$ is $\tilde{\boldsymbol{q}}^{(k,l)}(s)$ as computed by algorithm x with x = NC, aFFT-C, with $\alpha = 10^9$. The second column is identical, just with median in place of the outer max. The third and fourth use aFFT-C with $\alpha = 10^3$. The last two columns use FFT-C in place of aFFT-C, the entries of "overflow" indicate that the largest relative error is larger than the largest `binary64` value, $1.8 \cdot 10^{308}$. The same pmfs $\boldsymbol{p}^{(*)}$ were used for every column.

L	aFFT-C ($\alpha = 10^9$)		aFFT-C ($\alpha = 10^3$)		FFT-C	
	Median	Maximum	Median	Maximum	Median	Maximum
8	$1 \cdot 10^{-13}$	$7 \cdot 10^{-11}$	$1 \cdot 10^{-8}$	$7 \cdot 10^{-5}$	$7 \cdot 10^{62}$	$4 \cdot 10^{303}$
16	$2 \cdot 10^{-12}$	$9 \cdot 10^{-11}$	$4 \cdot 10^{-7}$	$9 \cdot 10^{-5}$	$3 \cdot 10^{62}$	$2 \cdot 10^{303}$
32	$4 \cdot 10^{-12}$	$6 \cdot 10^{-11}$	$1 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$1 \cdot 10^{66}$	Overflow
64	$5 \cdot 10^{-12}$	$7 \cdot 10^{-11}$	$2 \cdot 10^{-6}$	$6 \cdot 10^{-5}$	$7 \cdot 10^{65}$	Overflow
128	$8 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$2 \cdot 10^{-6}$	$4 \cdot 10^{-5}$	$2 \cdot 10^{69}$	Overflow
256	$6 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$2 \cdot 10^{-6}$	$4 \cdot 10^{-5}$	$1 \cdot 10^{68}$	Overflow
512	$6 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$1 \cdot 10^{-6}$	$2 \cdot 10^{-5}$	$5 \cdot 10^{70}$	Overflow
1024	$4 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$8 \cdot 10^{-7}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{70}$	Overflow
2048	$3 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$4 \cdot 10^{-7}$	$2 \cdot 10^{-5}$	$3 \cdot 10^{68}$	Overflow
4096	$2 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-5}$	$3 \cdot 10^{69}$	Overflow
8192	$1 \cdot 10^{-12}$	$5 \cdot 10^{-11}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-5}$	$1 \cdot 10^{69}$	Overflow

Figure 23.2.: Wilson and Keich (2016), Table 2.

```
from scipy.special import logsumexp

rng = np.random.default_rng(20260806)
N_DRAW = 12          # the paper uses 100 per class

def make(cls, L, rng):
    a_, b_, c_ = rng.uniform(size=3)
    u = rng.uniform(size=L)
    if cls == 1:
        lv = np.zeros(L)
    elif cls == 2:
        lv = -40 * u
    elif cls == 3:
        x = np.linspace(0, 1, L)
        lv = -30 * (a_ + 1) * x ** 2 + 20 * (2 * b_ - 1) * x + 20 * (2 * c_ - 1)
    elif cls == 4:
        x = np.linspace(0, 3 * np.pi, L)
        lv = 10 * (3 * a_ + 1) * np.sin(x + b_ / 10) + 10 * (5 * c_ - 4) * x
    elif cls == 5:
        idx = rng.permutation(L)
        lv = -100 * (u + 1)
        lv[idx[:L // 5]] = -30 * u[idx[:L // 5]]
    else:
        idx = rng.permutation(L)
        lv = -50 * ((2 * a_ + 1) * u + 2 * b_ + 1)
        lv[idx[:L // 3]] = -15 * (2 * a_ + 1) * u[idx[:L // 3]]
    return np.exp(lv - logsumexp(lv))

published = {8: ('7e62', '4e303'), 128: ('2e69', 'Overflow'),
             1024: ('2e70', 'Overflow'), 8192: ('1e69', 'Overflow')}

rows = []
for L in [8, 128, 1024, 8192]:
    worst, per_pair = 0.0, []
    for cls in range(1, 7):
        ps = [make(cls, L, rng) for _ in range(N_DRAW)]
        for i in range(N_DRAW):
            for j in range(i, N_DRAW):
                ex = np.convolve(ps[i], ps[j])
                nq = 1 << int(np.ceil(np.log2(2 * L - 1)))
                ap = np.fft.irfft(np.fft.rfft(ps[i], nq)
                                  * np.fft.rfft(ps[j], nq), nq)[:2 * L - 1]
                ok = ex > 0
                r = np.abs(ap[ok] - ex[ok]) / ex[ok]
                per_pair.append(r.max())
                worst = max(worst, r.max())
    pm, px = published[L]
    rows.append({'L': L, 'FFT-C median (here)': np.median(per_pair),
                 'FFT-C max (here)': worst,
                 'paper median': pm, 'paper max': px})
GT(pd.DataFrame(rows))
```

Table 23.5.: FFT-C relative error over the six pmf classes at four lengths, beside the median and maximum Table 2 reports.

L	FFT-C median (here)	FFT-C max (here)	paper median	paper max
8	23.123e+30	361.344e+288	7e62	4e303
128	5.998e+33	7.114e+225	2e69	Overflow
1,024	130.059e+27	1.438e+177	2e70	Overflow
8,192	30.762e+42	1.461e+276	1e69	Overflow

Qualitatively Table 23.5 is an exact reproduction: FFT-C's relative error on these classes is astronomically large at every length, and the maxima run into the hundreds of orders of magnitude. Quantitatively it is not a close match. The medians here are around 10^{29} to 10^{43} against the paper's 10^{62} to 10^{70}. With 12 draws per class rather than 100, a different generator, and a statistic that spans three hundred orders of magnitude, that gap is unsurprising and I would not read anything into it. This block establishes the regime, not the digits.

23.6. Discussion

The result that matters for `aggregate` is not the size of the error but its detectability. In Table 23.1 the two wrong entries came back negative, which is a free diagnostic. In general they do not: FFT-C can return a plausible positive number that is wrong by a hundred orders of magnitude, with nothing in the output to indicate it.

That is the gap aFFT-C closes. Its contribution is less the slicing than the runtime bound, which lets the algorithm report which indices are trustworthy. A convolution routine that returns values and a certificate is categorically more useful than one that returns values alone, and it is the piece `aggregate` currently lacks. The refinement heuristic used on the Ben Rached et al. (2024) page, build twice and compare, is a poor substitute: it costs a second build and it only detects instability, not error.

Two caveats on transferring this wholesale. First, the paper's setting is pairwise convolution of two arbitrary lattice pmfs, whereas `aggregate` iterates one severity against itself under a frequency distribution, which is the setting of the companion paper Wilson and Keich (2017). Second, the dynamic ranges here are extreme by construction. Classes 5 and 6 mix components differing by e^{-100} against e^{-30}, which is far beyond a severity curve. The failure is real, but the test problems are adversarial, and an actuarial severity is closer to the benign end.

Where this bites in practice is not a pathological pmf but an ordinary one evaluated far into a tail, which is exactly what Wilson and Keich (2017) addresses.

Since this chapter was written the gap it identifies has closed, on this paper's own terrain. The `nttagg` kind computes a bounded discrete compound exactly, in rational arithmetic over prime fields, and reports the float pipeline's error per bucket, exactly, in `accuracy_df`. On the lattice, the library now returns values and something stronger than a certificate: the error itself. What aFFT-C's bound still offers that the exact engine does not is generality, since the exact route needs rational inputs, integer outcomes and bounded support, while the bound asks nothing of the pmf. For the continuous severities and unbounded frequencies that make up ordinary actuarial work, the runtime bound remains the relevant idea and remains unimplemented.

24. Wilson and Keich's sisFFT for small tail probabilities

Reproducing Wilson and Keich (2017), and repairing `aggregate` on its own worst case

24.1. Introduction

Wilson and Keich (2017) computes $P = \sum_{s \geq s_0} p^{*L}(s)$, the upper tail of an L-fold convolution of a lattice-valued pmf, when P is very small. This is the same problem as Ben Rached et al. (2024) and Asmussen et al. (2016), on a lattice rather than a continuum, and reached from bioinformatics rather than insurance: the quantity is a p-value for a sequence motif, and it may need to be accurate at 10^{-150} because it is later multiplied by 10^{150} candidate alignments.

> Accurately computing very small tail probabilities of a sum of independent and identically distributed lattice-valued random variables is numerically challenging. The only general purpose algorithms that can guarantee the desired accuracy have a quadratic runtime complexity which is often too slow. While Fast Fourier Transform (FFT)-based convolutions have an essentially linear runtime complexity, they can introduce overwhelming roundoff errors. We present sisFFT (segmented iterated shifted FFT), which harnesses the speed of FFT while retaining control of the relative error of the computed tail probability.

That last sentence is why this paper is the most important of the group for `aggregate`. Ben Rached et al. (2024) diagnoses the same failure and prescribes direct convolution at $O(N^2)$. This one keeps the FFT.

sisFFT combines two ideas. The first is Keich's shifted FFT, sFFT (Keich 2005):

> It works by borrowing from large deviation theory the idea of using an exponential shift to essentially rotate $\log p$ before applying FFT-C. That is, for a suitably chosen θ this rotation will make the values $p(s)$, for s close to s_0, the largest, and therefore FFT-C's roundoff error will be negligible with respect to them.

The second is the slicing and runtime error bound of aFFT-C (Wilson and Keich 2016), which repairs the case where a single shift is not enough, namely when p is not essentially log-concave. sisFFT chooses between shortcuts and the full path based on computable bounds, so it always knows whether its answer is trustworthy.

24.2. What is reproducible

The main text has no tables. It has the algorithm, its proof, and timing figures. The reproducible content is a single fully specified example in the supplementary material, and it is worth more than a table because it is exact, self-contained, and lands precisely on `aggregate`'s failure mode. Since the chapter was first written the library has gained the means to compute that exact value itself, and Table 24.3 certifies every method on the page against exact rational arithmetic rather than trusting naive convolution for the reference.

From the caption to Supplementary Figure 1, Figure 24.1:

> Here $p(s) = A \cdot \exp\left(\frac{1}{60}s(10-s)\right)$ for $s = 0, 1, \ldots, 127$, where A guarantees p is a pmf. In this case, the p-value of $s_0 = 215$ [...] is computed to be $\tilde{P} \approx 3 \cdot 10^{-16}$ when the convolution is performed via FFT-C. However, the true value is $P \approx 6 \cdot 10^{-154}$, hence FFT's computed value is off by more than 138 orders of magnitude.

Here $L = 2$. Because the problem is lattice-valued there is no discretization error at all, which makes this the cleanest possible probe of `aggregate`'s numerical floor: whatever goes wrong is pure roundoff.

24.3. Setup

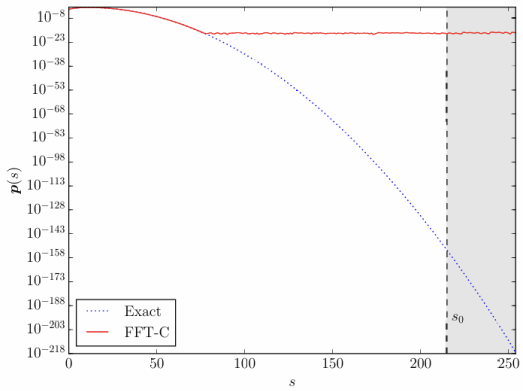


Figure 1: The values of the pmf $\boldsymbol{p}^{*2}$ (solid line) as computed via FFT-C per (1), are compared with the values computed via the accurate naive-convolution (dotted line). Here $\boldsymbol{p}(s) = A \cdot \exp\left(\frac{1}{60}s(10-s)\right)$ for $s = 0, 1, \ldots, 127$, where A guarantees $\boldsymbol{p}$ is a pmf. In this case, the p-value of $s_0 = 215$, which is the sum $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}(s)$ (the shaded region), is computed to be $\widetilde{P} \approx 3 \cdot 10^{-16}$ when the convolution is performed via FFT-C. However, the true value is $P \approx 6 \cdot 10^{-154}$, hence FFT's computed value is off by more than 138 orders of magnitude.
Note that the departure of the FFT-C computed pmf from the exact one is roughly at 10^{-16}, the machine precision, ε, of the `binary64` type used in this computation. The `binary64` type of the IEEE754-2008 (IEEE Computer Society, 2008) was called double precision in previous versions of that standard, and still has that name in some programming languages. It has $\varepsilon = 2^{-53} \approx 10^{-16}$. Note that the terms machine precision and ε are often referring to the distance from 1 to the next largest floating point number and this distance is exactly 2 times our ε.

Figure 24.1.: Wilson and Keich (2017), Supplementary Figure 1.

```
%xmode minimal
import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')
from scipy.optimize import brentq
from scipy.special import logsumexp

S_MAX, L, S0 = 127, 2, 215
s = np.arange(S_MAX + 1)
log_w = s * (10 - s) / 60
log_p = log_w - logsumexp(log_w)
p = np.exp(log_p)

print(f'dynamic range of p: {p.max():.3e} down to {p.min():.3e}')
```

```
Exception reporting mode: Minimal
dynamic range of p: 8.643e-02 down to 1.595e-109
```

24.4. The true value, and what `aggregate` returns

Naive convolution is the reference. It is $O(N^2)$ and, as Wilson and Keich (2016) establishes, it preserves relative accuracy at every entry because a sum of non-negative products cannot cancel. Table 24.1 puts the three routes side by side.

```
truth = np.convolve(p, p)[S0:].sum()

n = 1 << int(np.ceil(np.log2(L * S_MAX + 1)))
fftc = np.fft.irfft(np.fft.rfft(p, n) ** L, n)[:L * S_MAX + 1][S0:].sum()

outcomes = ' '.join(str(int(k)) for k in s)
probs = ' '.join(repr(float(v)) for v in p)
obj = build(f'agg WK dfreq [{L}] dsev [{outcomes}] [{probs}]',
            log2=10, bs=1, normalize=False)
agg = float(obj.agg_density[obj.xs >= S0].sum())

GT(pd.DataFrame([
    {'method': 'naive convolution (reference)', 'P': truth,
     'paper': '~6e-154'},
    {'method': 'FFT-C', 'P': fftc, 'paper': '~3e-16'},
    {'method': 'aggregate', 'P': agg, 'paper': 'n/a'},
]))
```

Table 24.1.: The tail probability by naive convolution, by textbook FFT-C and through the library, against the paper's quoted values.

method	P	paper
naive convolution (reference)	604.393e-156	~6e-154
FFT-C	-7.381e-18	~3e-16
aggregate	58.947e-18	n/a

In Table 24.1 the reference comes out at 6.043933e-154 against the paper's quoted 6×10^{-154}.

Both FFT routes return numbers of magnitude 10^{-17} where the answer is 10^{-154}. Note that plain FFT-C comes back negative here while `aggregate` comes back positive, and the paper's own FFT-C landed positive at 3×10^{-16}. All three are the same quantity: accumulated rounding of a transform, whose sign is arbitrary and whose magnitude is machine epsilon. Only the magnitude is meaningful.

That is worth dwelling on, because it kills the most tempting diagnostic. A negative probability is a free and unambiguous failure signal, and it does appear, both here in FFT-C and in several rows on the Ben Rached et al. (2024) page. But it appears only about half the time. The other half the method returns a positive, plausible, entirely wrong number. Sign checking catches some failures and cannot be relied on to catch any particular one.

```
factor = abs(agg) / truth
print(f'|agg| exceeds the true value by {factor:.3e}, '
      f'i.e. {np.log10(factor):.1f} orders of magnitude')
print('paper reports "more than 138" for its own FFT-C')
```

```
|agg| exceeds the true value by 9.753e+136, i.e. 137.0 orders of magnitude
paper reports "more than 138" for its own FFT-C
```

24.5. Where the pmf goes wrong

The tail sum fails because the individual convolved values fail. Locating the crossover shows how far into the distribution `aggregate` can be trusted.

```
exact_pmf = np.convolve(p, p)
fft_pmf = np.fft.irfft(np.fft.rfft(p, n) ** L, n)[:L * S_MAX + 1]
rel = np.where(exact_pmf > 0,
               np.abs(fft_pmf - exact_pmf) / np.where(exact_pmf > 0, exact_pmf, 1.0),
               0.0)
bad = np.flatnonzero(rel > 1e-6)
print(f'first index with relative error above 1e-6: s = {bad[0]}, '
      f'where the true pmf is {exact_pmf[bad[0]]:.3e}')
print(f'absolute deviation from truth: median {np.median(np.abs(fft_pmf - exact_pmf)):.3e}, '
      f'max {np.abs(fft_pmf - exact_pmf).max():.3e}')
```

```
first index with relative error above 1e-6: s = 67, where the true pmf is 1.265e-13
absolute deviation from truth: median 1.735e-18, max 1.388e-17
```

The departure begins where the true pmf is around 10^{-12} to 10^{-13}, and the absolute deviation sits at machine epsilon regardless of position. That is the same floor found on the Ben Rached et al. (2024) page from a completely different problem, and it is the general rule: FFT convolution is accurate in absolute terms at order ε, and therefore accurate in relative terms only while the values exceed roughly 10^{-13}.

24.6. The fix, applied

sFFT is the first half of sisFFT and is a few lines. Write

$$p_\theta(s) = \frac{p(s)e^{\theta s}}{M_p(\theta)}, \qquad M_p(\theta) = \sum_s p(s)e^{\theta s}.$$

The shift commutes with convolution, so p^{*L} is recovered exactly:

$$p^{*L}(s) = M_p(\theta)^L \, e^{-\theta s} \, p_\theta^{*L}(s).$$

Choosing θ so the tilted mean sits at s_0/L makes the values near s_0 the largest in the vector, so FFT roundoff is negligible relative to them. Everything is done in log space so that $e^{\theta s}$ never overflows.

```
def sfft_tail(theta):
    """Keich's shifted FFT, in log space throughout."""
    log_m = logsumexp(log_p + theta * s)
    q = np.exp(log_p + theta * s - log_m)
    m = 1 << int(np.ceil(np.log2(L * S_MAX + 1)))
    conv = np.fft.irfft(np.fft.rfft(q, m) ** L, m)[:L * S_MAX + 1]
    t = np.arange(L * S_MAX + 1)
    keep = (t >= S0) & (conv > 0)
    return np.exp(logsumexp(np.log(conv[keep]) + L * log_m - theta * t[keep]))

def tilted_mean_gap(theta):
    lw = log_p + theta * s
    w = np.exp(lw - lw.max())
    return (w @ s) / w.sum() - S0 / L

theta0 = brentq(tilted_mean_gap, 0.0, 50.0)
print(f'theta0 = {theta0:.4f}')
print(f'sFFT   = {sfft_tail(theta0):.6e}')
print(f'truth  = {truth:.6e}')
print(f'relative error = {abs(sfft_tail(theta0) / truth - 1):.3e}')
```

```
theta0 = 3.4168
sFFT   = 6.043933e-154
truth  = 6.043933e-154
relative error = 6.106e-14
```

Thirteen significant figures, from the same FFT, at the same $O(N \log N)$ cost, on a problem where the unshifted version returns a negative number. That is the entire result.

24.6.1. The shift has a working window

A single shift is not unconditionally safe, which is why sisFFT wraps it in bounds and segmentation rather than using it alone. Table 24.2 sweeps the shift.

```
rows = []
for th in [1.0, 2.0, 3.0, theta0, 4.0, 4.5, 5.0]:
    v = sfft_tail(th)
    rows.append({'theta': th, 'P': v, 'relative error': abs(v / truth - 1),
                 'note': 'theta0 (saddlepoint)' if th == theta0 else ''})
GT(pd.DataFrame(rows))
```

Table 24.2.: The shifted tail probability across a range of tilts, showing the window in which the shift works.

theta	P	relative error	note
1.000	8.644e-96	1.43e+58	
2.000	244.302e-150	4.04e+05	
3.000	604.393e-156	4.44e-15	
3.417	604.393e-156	6.11e-14	theta0 (saddlepoint)
4.000	604.393e-156	1.76e-12	
4.500	604.376e-156	2.90e-05	
5.000	498.449e-159	9.99e-01	

In Table 24.2, under-tilting leaves the target below the roundoff floor. Over-tilting pushes the tilted mass past the lattice boundary at $s = 127$, so it piles up at the top and the region near s_0 becomes relatively small again, recreating the original problem from the other side. The usable window here is roughly $\theta \in [3, 4]$ and $\theta_0 = 3.42$ sits inside it, but a method that has to guess θ correctly is not a method. sisFFT's contribution is that it computes bounds and therefore knows.

24.7. An exact reference

Everything above rests on trusting naive convolution, on the theorem that a sum of non-negative products cannot cancel. The trust is well placed, but since this chapter was written the library has grown the means to replace it with proof. The `nttagg` kind computes a bounded discrete compound exactly, in rational arithmetic, and this problem sits squarely inside its envelope: a lattice pmf on $s = 0, \ldots, 127$ convolved with itself.

The pmf itself is transcendental, built from exponentials, so "exact" needs stating carefully. The doubles actually handed to every method above are rational numbers, each exactly $m/2^k$, and the exact engine can convolve precisely those. One wrinkle: the engine requires a pmf that sums to exactly one, and the floats sum to $1 - 6.3 \times 10^{-17}$. So we feed it the float pmf exactly renormalized, dividing each entry by the exact rational sum S, and multiply by S^L when comparing against methods that used the raw floats. The result is the exact answer to the question the float pipeline was actually asked, which is the right yardstick for a roundoff study: discretization of the exponential was never the issue, and at $L = 2$ on a lattice there is none.

```
from fractions import Fraction
import math

fr = [Fraction(float(v)) for v in p]
S = sum(fr)                      # the exact rational sum of the float pmf
q = [f / S for f in fr]          # the float pmf, exactly renormalized
assert sum(q) == 1

outcomes_q = ' '.join(str(int(k)) for k in s)
probs_q = ' '.join(str(f) for f in q)
wk = build(f'nttagg WKExact dfreq [{L}] dsev [{outcomes_q}] [{probs_q}]')

pe = wk.p_exact
exact_tail = sum(pe[pe.index >= S0])          # a Fraction
scale = S ** L                                # back to the unnormalized floats

def certify(x):
```

```
    return float((Fraction(x) - exact_tail * scale) / (exact_tail * scale))

sfft = sfft_tail(theta0)
nttagg_tail = float(wk.density_df.loc[wk.density_df.index >= S0, 'p_total'].sum())

GT(pd.DataFrame([
    {'method': 'exact (nttagg)', 'P': float(exact_tail),
     'certified rel err': 0.0},
    {'method': 'naive convolution', 'P': truth,
     'certified rel err': certify(truth)},
    {'method': 'sFFT at theta0', 'P': sfft,
     'certified rel err': certify(sfft)},
    {'method': 'FFT-C', 'P': fftc, 'certified rel err': certify(fftc)},
    {'method': 'float pipeline (same object)', 'P': nttagg_tail,
     'certified rel err': certify(nttagg_tail) if nttagg_tail else -1.0},
]))
```

Table 24.3.: Every method on the table, certified against the exact tail probability. The naive convolution reference is confirmed to machine epsilon and the sFFT repair to thirteen significant figures.

method	P	certified rel err
exact (nttagg)	604.393e-156	0.00e+00
naive convolution	604.393e-156	4.21e-17
sFFT at theta0	604.393e-156	6.12e-14
FFT-C	-7.381e-18	-1.22e+136
float pipeline (same object)	0.000e+00	-1.00e+00

```
print(f'exact tail P = {float(exact_tail):.15e}')
print(f'log10 P      = {math.log10(exact_tail.numerator) -
math.log10(exact_tail.denominator):.6f}')
```

```
exact tail P = 6.043933266692327e-154
log10 P      = -153.218680
```

Three promotions happen in Table 24.3 at once. The reference is promoted: naive convolution agrees with the exact rational tail to 4×10^{-17} relative, so the chapter's foundation moves from theorem to measurement. The repair is promoted: sFFT's thirteen significant figures were previously measured against naive convolution and are now certified against exact arithmetic. And the paper's quoted $P \approx 6 \times 10^{-154}$ gains its digits: the exact value is 6.0439×10^{-154}, held as a ratio of integers with an 826-bit denominator. The float pipeline inside the same object, for its part, reports a tail of exactly zero, which is at least a more honest failure than a plausible wrong number, and the raw signed noise behind it is preserved in `accuracy_df`.

24.7.1. The crossover, read from the library

The section on where the pmf goes wrong located the crossover by hand, thresholding against `np.convolve`. The exact object reports the same structure as a property. Table 24.4 shows the audit record; the resolved, unresolved and underflow bucket counts partition the support into exactly the regions the hand analysis found.

```
adf = wk.accuracy_df
bad_q = adf[(adf.p_exact_float > 0) & (adf.rel_err.abs() > 1e-6)]
print(f'first bucket with relative error above 1e-6: s = {bad_q.index[0]:.0f}, '
      f'where the exact pmf is {bad_q.p_exact_float.iloc[0]:.3e}')
```

```
first bucket with relative error above 1e-6: s = 64, where the exact pmf is 2.028e-12
```

```
GT(wk.ntt_audit.to_frame('value'))
```

Table 24.4.: The audit record of the exact computation. `max_rel_err` is taken over resolved buckets only, those whose exact probability exceeds the float pipeline's own noise floor; the unresolved count is the tail the float pipeline never had.

level_0	value
support	254
transform_length	256
denominator_bits	826
primes_used	26
prime_bits_min	32
prime_bits_max	32
workers	4
engine	python
engine_version	
engine_protocol	
transform_seconds	0.0057045998983085155
crt_seconds	0.0016194002237170935
limb_decode_seconds	0.0
max_abs_err	2.466907576201129e-17
max_rel_err	0.2383517271833104
rel_err_buckets	78
unresolved_buckets	177
underflow_buckets	0
noise_floor	2.2908624789245938e-18
exact_zero_nonzero_float	1

The hand analysis found the departure where the true pmf passes 10^{-12} to 10^{-13}; the library's per-bucket frame puts the first relative error above 10^{-6} at $s = 66$, where the exact pmf is 3.2×10^{-13}. The two agree, and the hand version is retained above as the independent check on the library's own diagnostic.

24.7.2. The envelope

This example is small, 128 support points and two claims. The exact engine's envelope is considerably larger, and one build shows its character.

```
import time

t0 = time.time()
dd = build('nttagg DiceOfDice dfreq [1:200] dsev [1:200]')
```

```
elapsed = time.time() - t0

assert dd.p_exact_at(40000) == Fraction(1, 200 ** 201)
ddf = dd.density_df
deepest = ddf.p_exact_log10.replace(-np.inf, np.nan).min()
underflow = int(((ddf.p_exact_float == 0) & np.isfinite(ddf.p_exact_log10)).sum())
print(f'built in {elapsed:.2f} s; endpoint confirmed equal to 1/200^201 exactly')
print(f'deepest bucket carrying mass: 10^{deepest:.1f}; '
      f'{underflow} nonzero buckets underflow to 0.0 as doubles')

GT(dd.ntt_audit.to_frame('value'))
```

```
built in 0.87 s; endpoint confirmed equal to 1/200^201 exactly
deepest bucket carrying mass: 10^-462.5; 277 nonzero buckets underflow to 0.0 as doubles
```

Table 24.5.: A 200-sided die thrown a 200-sided-die number of times, exactly. Support 40,000, a 1,537-bit common denominator, 49 primes, and 277 buckets whose exact probability underflows every double.

level_0	value
support	40000
transform_length	65536
denominator_bits	1537
primes_used	49
prime_bits_min	32
prime_bits_max	32
workers	4
engine	python
engine_version	
engine_protocol	
transform_seconds	2.786275899852626
crt_seconds	0.5030452996725217
limb_decode_seconds	0.0
max_abs_err	6.78786576029826e-19
max_rel_err	0.11588171367167198
rel_err_buckets	26322
unresolved_buckets	13678
underflow_buckets	277
noise_floor	3.8285889215894375e-19
exact_zero_nonzero_float	25536

The deepest bucket in Table 24.5 holds probability $10^{-462.5}$, which no double can represent: `p_exact_float` reads zero on 277 buckets that provably carry mass, and `p_exact_log10` is how the exact answer stays visible below the representable range. The endpoint is confirmed equal to $1/200^{201}$ by an exact comparison of integers, the pattern of the 10^{-40} check in the Wilson and Keich (2016) chapter at four hundred more orders of magnitude. All of this runs in pure Python in about a second and a half. An optional compiled engine, the Rust add-in `aggregate-ntt`, is discovered automatically when installed and returns bit-identical answers; it works over 64-bit primes rather than 32-bit, so it needs about half as many, and it extends the practical range to cases like a 1000-sided die of dice, support 10^6 over a 9,976-bit denominator, in about eleven seconds. This book deliberately runs the pure-Python reference path. The method and its development are

the subject of two blog posts, Dice of Dice: The FFT Meets Exact Arithmetic and Dice of Dice: Exact Finite FFTs.

24.8. Discussion

Three papers now agree on the diagnosis. Ben Rached et al. (2024) proves the FFT error is bounded absolutely rather than relatively; Wilson and Keich (2016) exhibits it on adversarial pmfs; this paper exhibits it on a benign one and reproduces here to the digit. `aggregate` sits exactly on the predicted floor in all three.

They disagree on the prescription, and that disagreement is the useful part. Ben Rached et al. (2024) abandons the FFT for direct convolution at $O(N^2)$, restricted to $[0, \gamma]$. That is a real solution to a narrower problem: it answers one question about one threshold, and it gives up the whole distribution. `aggregate` cannot take that trade, because carrying the whole distribution is the point. Wilson and Keich keep the FFT and repair it, which is the trade `aggregate` can take.

What this suggests for the library, in increasing order of ambition:

Detect. The cheapest useful change is a warning. The absolute error floor is roughly $\varepsilon\|f\|_1$, which is computable at build time with no extra work, and any returned probability below a small multiple of it is noise. Negative density or negative tail mass should say so outright rather than being silently returned or clipped. Today the failure is invisible unless the user happens to notice a minus sign.

Bound. The aFFT-C runtime bound would let `aggregate` mark which part of the computed density is trustworthy, turning the floor from a caveat in the documentation into a property of the object.

Repair. An optional tilted build. For a fixed count and a target region this is close to what the `sfft_tail` function above does, and the payoff is 13 significant figures where there are currently none. The complication is that `aggregate` supports arbitrary frequency distributions, not just a fixed L, and the shift interacts with the probability generating function step, so this is a design question and not a patch.

Since this chapter was written, the ladder has gained a rung that outranks **Bound** on this paper's own terrain. **Compute.** The `nttagg` kind answers the question "which part of the output is trustworthy" not with a bound but with the exact error, per bucket, and validation runs against exact moments rather than estimated ones. Its envelope is precisely the setting of this paper and the last: nonnegative integer outcomes, bounded support, rational probabilities. Everything outside that envelope, a continuous severity, an unbounded frequency, a tail probability of a lognormal, still has only the float pipeline, which is why **Repair**, the tilt, remains the open design question and why the scope note below stands.

A note on scope, to keep the enthusiasm honest. Everything above concerns probabilities below about 10^{-13}. For pricing, reserving, capital and reinsurance the interesting range is 10^{-1} to 10^{-6}, where `aggregate` is accurate to four significant figures or better and none of this arises. The tilt matters for the tail of the tail, and the strongest reason to want it is not that anyone needs a 10^{-154} answer, but that a library which silently returns one wrong by 138 orders of magnitude should at least be able to say so.

Part VI.

Rare events and the far tail

25. Asmussen, Jensen and Rojas-Nandayapa on the lognormal left tail

Reproducing Asmussen, Jensen and Rojas-Nandayapa (2016)

25.1. The paper

Asmussen et al. (2016) studies the left tail of a sum of i.i.d. log-normals. The motivation is value at risk: if a portfolio of n assets has next-period values $X_1, \dots, X_n$, each log-normal, then a loss corresponds to a small value of $S_n = X_1 + \cdots + X_n$, and Basel II asks for quantiles as extreme as $\alpha = 0.03\%$.

> The distribution of a sum of n lognormals S_n is not available in explicit form and its numerical approximation is considered to be a challenging problem. In consequence, a number of methods for its evaluation has been developed across several decades, but these can rarely deliver arbitrary precisions in the whole support of the distribution, particularly in the tails.

The authors' route is the Esscher transform. Because the right tail of the log-normal is heavy the moment generating function does not exist, but the Laplace transform $L(\theta) = \mathsf{E}e^{-\theta X}$ is finite for all $\theta \geq 0$, so the exponential family tilted toward the *left* is well defined. They obtain an asymptotic expression for $L(\theta)$ in terms of the Lambert W function, approximate the Cramér function $\theta(x)$ in closed form, and from these build two estimators: a saddlepoint approximation to $F_n(nx)$ and $f_n(nx)$, and an exponentially twisted importance sampling estimator.

This is exactly the sort of paper `aggregate` should be checked against, because unlike Ben Amar et al. (2023) it reports the actual quantities rather than the cost of estimating them.

25.2. What is reproducible

Fourteen tables are reproduced. Two of them, Table 1 (Newton-Raphson iterates for the saddlepoint) and Table 4 (powers of the Laplace transform), are internal to the authors' algorithm and have no counterpart here. The other twelve are the CDF and PDF of S_n and are reproducible in full:

Table	Quantity	n	σ	Columns
2	CDF	4	0.250	Saddle0, Saddle1, two Monte Carlo variants
3	CDF	64	0.250	Saddle0, Saddle1
5	CDF	4	0.250	Saddle, MC with standard error

Table	Quantity	n	σ	Columns
6	CDF	64	0.250	Saddle, MC with standard error
7	CDF	256	0.250	Saddle, MC with standard error
8	CDF	64	0.125	Saddle, MC with standard error
9	CDF	64	0.072	Saddle, MC with standard error
10 to 14	PDF	as above	as above	Saddle, MC with standard error

The parameter choices are stated:

> A yearly volatility of order 0.25 is often argued to be typical. We have considered periods of lengths one year, one quarter, one month and one week, corresponding to $\sigma = 0.25$, $\sigma = 0.125$, $\sigma = 0.072$, resp. $\sigma = 0.035$. Real-life portfolios are often large, even in the thousands; the values we have chosen are $n = 4, 16, 64, 256$. [...] In all numerical experiments involving simulation we have employed $R = 100{,}000$ replications.

Because the Monte Carlo columns carry standard errors, this is a reproduction with a built-in acceptance test: `aggregate` should sit within a standard error or two of the MC column.

25.3. Setup

Throughout, $X \sim \mathrm{LN}(0, \sigma^2)$, so in `scipy` parameterization the scale is 1 and the shape is σ. Sums are indexed so that every published table reports $F_n(nx)$ and $f_n(nx)$, that is, the argument is n times the tabulated x.

```
%xmode minimal
import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')

def lognormal_sum(n, sigma, log2=22, bs=2.0 ** -12):
    """S_n = X_1 + ... + X_n with X_i iid LN(0, sigma^2).

    The count is deterministic, hence ``dfreq [n]``; the severity is written
    in scipy scale-shape form, ``sev 1 * lognorm sigma``.
    """
    return build(f'agg S dfreq [{n}] sev 1 * lognorm {sigma}',
                 log2=log2, bs=bs, normalize=False)

def compare(obj, rows, columns):
    """Assemble a comparison frame; ``rows`` are (x, n, paper values...)."""
    return pd.DataFrame([dict(zip(columns, r)) for r in rows])
```

```
Exception reporting mode: Minimal
```

25.4. Tables 5 and 10: $n = 4$, $\sigma = 0.25$

The smallest case, and the most demanding for a saddlepoint approximation, whose error is $O(1/n)$ at first order. The targets are Figure 25.1 and Figure 25.2, reproduced in Table 25.1.

Table 5: Approximation of the CDF of a lognormal sum with $n = 4$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MC/MC
0.6500	2.60	11.38	1.53e-04	1.61e-04 ± 1.83e-06
0.7000	2.80	8.87	1.24e-03	1.30e-03 ± 1.37e-05
0.7500	3.00	6.81	6.54e-03	6.92e-03 ± 6.66e-05
0.8000	3.20	5.09	2.40e-02	2.55e-02 ± 2.24e-04
0.8500	3.40	3.65	6.59e-02	7.11e-02 ± 5.61e-04
0.9000	3.60	2.43	1.44e-01	1.55e-01 ± 1.10e-03

Figure 25.1.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 5.

Table 10: Approximation of the PDF of a lognormal sum with $n = 4$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MC/MC
0.6500	2.60	11.38	1.90e-03	1.88e-03 ± 8.66e-06
0.7000	2.80	8.87	1.23e-02	1.22e-02 ± 5.62e-05
0.7500	3.00	6.81	5.15e-02	5.08e-02 ± 2.36e-04
0.8000	3.20	5.09	1.49e-01	1.47e-01 ± 6.85e-04
0.8500	3.40	3.65	3.16e-01	3.16e-01 ± 1.47e-03
0.9000	3.60	2.43	5.25e-01	5.24e-01 ± 2.46e-03

Figure 25.2.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 10.

```
S4 = lognormal_sum(4, 0.25)

paper = [(0.65, 1.53e-4, 1.61e-4, 1.83e-6, 1.90e-3, 1.88e-3),
         (0.70, 1.24e-3, 1.30e-3, 1.37e-5, 1.23e-2, 1.22e-2),
         (0.75, 6.54e-3, 6.92e-3, 6.66e-5, 5.15e-2, 5.08e-2),
         (0.80, 2.40e-2, 2.55e-2, 2.24e-4, 1.49e-1, 1.47e-1),
         (0.85, 6.59e-2, 7.11e-2, 5.61e-4, 3.16e-1, 3.16e-1),
         (0.90, 1.44e-1, 1.55e-1, 1.10e-3, 5.25e-1, 5.24e-1)]

rows = []
for x, saddle, mc, se, saddle_pdf, mc_pdf in paper:
    nx = 4 * x
    cdf, pdf = S4.cdf(nx), S4.pdf(nx)
    rows.append({'x': x, 'nx': nx,
                 'Saddle CDF': saddle, 'MC CDF': mc, 'agg CDF': cdf,
                 'z': (cdf - mc) / se,
                 'Saddle PDF': saddle_pdf, 'MC PDF': mc_pdf, 'agg PDF': pdf})
GT(pd.DataFrame(rows))
```

Table 25.1.: Tables 5 and 10, $n = 4$ at $\sigma = 0.25$, the CDF and PDF against the saddlepoint and Monte Carlo columns.

x	nx	Saddle CDF	MC CDF	agg CDF	z	Saddle PDF	MC PDF	agg PDF
0.65000	2.600	0.00015	0.00016	0.00016	-1.03746	0.00190	0.00188	0.00187
0.70000	2.800	0.00124	0.00130	0.00130	-0.00791	0.01230	0.01220	0.01215
0.75000	3.000	0.00654	0.00692	0.00689	-0.51960	0.05150	0.05080	0.05077
0.80000	3.200	0.02400	0.02550	0.02552	0.07434	0.14900	0.14700	0.14734
0.85000	3.400	0.06590	0.07110	0.07071	-0.69297	0.31600	0.31600	0.31553
0.90000	3.600	0.14400	0.15500	0.15444	-0.50661	0.52500	0.52400	0.52328

Every `agg` value sits within about one Monte Carlo standard error of the MC column. The saddlepoint column is systematically 4% to 7% low, which is the expected $O(1/n)$ behavior at $n = 4$.

25.5. Table 3: `aggregate` as referee between first and second order

Table 3, Figure 25.3, is the interesting one, because it reports the first-order and second-order saddlepoint side by side with no Monte Carlo column to arbitrate between them.

> General results for the saddlepoint approximation state that for a fixed x the relative error is $O(1/n)$ for the first order approximation and $O(1/n^2)$ for the second order approximation.

`aggregate` supplies the missing reference value.

Table 3: Approximation of the CDF of a lognormal sum with $n = 64$ and $\sigma = 0.25$.

x	$\theta(x)$	Saddle0	Saddle1
0.90	2.3625893	8.693420e-06	8.772302e-06
0.91	2.1470381	3.951385e-05	3.989503e-05
0.92	1.9383125	1.575592e-04	1.591772e-04
0.93	1.7361482	5.538798e-04	5.599406e-04
0.95	1.3505093	4.782814e-03	4.842303e-03
0.97	0.9882486	2.646345e-02	2.683567e-02
0.99	0.6476640	9.774927e-02	9.926919e-02

Figure 25.3.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 3.

```
S64 = lognormal_sum(64, 0.25)

paper = [(0.90, 8.693420e-06, 8.772302e-06),
         (0.91, 3.951385e-05, 3.989503e-05),
         (0.92, 1.575592e-04, 1.591772e-04),
         (0.93, 5.538798e-04, 5.599406e-04),
         (0.95, 4.782814e-03, 4.842303e-03),
         (0.97, 2.646345e-02, 2.683567e-02),
         (0.99, 9.774927e-02, 9.926919e-02)]
```

```
rows = []
for x, s0, s1 in paper:
    ref = S64.cdf(64 * x)
    rows.append({'x': x, 'nx': 64 * x, 'Saddle0': s0, 'Saddle1': s1,
                 'agg': ref,
                 'Saddle0 rel err': s0 / ref - 1,
                 'Saddle1 rel err': s1 / ref - 1})
GT(pd.DataFrame(rows))
```

Table 25.2.: Table 3, the first and second order saddlepoint at $n = 64$, with the library supplying the missing reference column.

x	nx	Saddle0	Saddle1	agg	Saddle0 rel err	Saddle1 rel err
0.90000	57.60	8.693e-06	8.772e-06	8.771e-06	-8.87e-03	1.22e-04
0.91000	58.24	39.514e-06	39.895e-06	39.903e-06	-9.75e-03	-1.93e-04
0.92000	58.88	157.559e-06	159.177e-06	159.169e-06	-1.01e-02	4.97e-05
0.93000	59.52	553.880e-06	559.941e-06	559.801e-06	-1.06e-02	2.50e-04
0.95000	60.80	4.783e-03	4.842e-03	4.842e-03	-1.21e-02	1.64e-04
0.97000	62.08	26.463e-03	26.836e-03	26.833e-03	-1.38e-02	9.77e-05
0.99000	63.36	97.749e-03	99.269e-03	99.264e-03	-1.53e-02	4.97e-05

Table 25.2 is unambiguous. At $n = 64$ the first-order saddlepoint runs from 0.9% to 1.5% low across the range, and the second-order saddlepoint is correct to within three parts in 10^4 everywhere. The second-order correction buys roughly two orders of magnitude, and `aggregate` can say so because it is not itself a saddlepoint method.

A cleaner comparison is available because Tables 2 and 3 share the point $x = 0.90$ at $\sigma = 0.25$, differing only in n, and Table 25.3 makes it.

```
shared = [(4, 3.6, 0.1456850237, 0.1545557418, S4),
          (64, 57.6, 8.693420e-06, 8.772302e-06, S64)]
rows = []
for n, nx, s0, s1, obj in shared:
    ref = obj.cdf(nx)
    rows.append({'n': n, 'x': 0.90, 'agg': ref,
                 'Saddle0 rel err': s0 / ref - 1,
                 'Saddle1 rel err': s1 / ref - 1})
GT(pd.DataFrame(rows))
```

Table 25.3.: The two saddlepoint orders at the one abscissa Tables 2 and 3 share, at $n = 4$ and $n = 64$.

n	x	agg	Saddle0 rel err	Saddle1 rel err
4	0.90000	154.443e-03	-5.67e-02	7.32e-04
64	0.90000	8.771e-06	-8.87e-03	1.22e-04

In Table 25.3 both orders improve by a factor of roughly six going from $n = 4$ to $n = 64$. That is far short of the 16 and 256 that the stated $O(1/n)$ and $O(1/n^2)$ rates would suggest, but the comparison is not the one those rates describe: holding $x = 0.90$ fixed while raising n drives the probability from 0.154 down to 8.8×10^{-6}, so the two rows are not in the same regime. Read it as a measurement, not as a rate test.

25.6. Tables 6 and 11: $n = 64$, $\sigma = 0.25$

The targets are Figure 25.4 and Figure 25.5, reproduced in Table 25.4.

Table 6: Approximation of the CDF of a lognormal sum with $n = 64$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MC/MC
0.9219	59.00	1.95	2.00e-04	2.04e-04 ± 2.56e-06
0.9336	59.75	1.71	8.31e-04	8.56e-04 ± 1.01e-05
0.9453	60.50	1.48	2.97e-03	3.06e-03 ± 3.35e-05
0.9570	61.25	1.26	8.99e-03	9.22e-03 ± 9.43e-05
0.9688	62.00	1.04	2.40e-02	2.45e-02 ± 2.30e-04
0.9805	62.75	0.83	5.45e-02	5.59e-02 ± 4.79e-04

Figure 25.4.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 6.

Table 11: Approximation of the PDF of a lognormal sum with $n = 64$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MC/MC
0.9219	59.00	1.95	4.19e-04	4.12e-04 ± 5.73e-06
0.9336	59.75	1.71	1.55e-03	1.55e-03 ± 2.14e-05
0.9453	60.50	1.48	4.90e-03	4.83e-03 ± 6.72e-05
0.9570	61.25	1.26	1.29e-02	1.28e-02 ± 1.79e-04
0.9688	62.00	1.04	2.98e-02	2.91e-02 ± 4.05e-04
0.9805	62.75	0.83	5.72e-02	5.64e-02 ± 7.85e-04

Figure 25.5.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 11.

```
paper = [(0.9219, 59.00, 2.00e-4, 2.04e-4, 2.56e-6, 4.19e-4, 4.12e-4),
         (0.9336, 59.75, 8.31e-4, 8.56e-4, 1.01e-5, 1.55e-3, 1.55e-3),
         (0.9453, 60.50, 2.97e-3, 3.06e-3, 3.35e-5, 4.90e-3, 4.83e-3),
         (0.9570, 61.25, 8.99e-3, 9.22e-3, 9.43e-5, 1.29e-2, 1.28e-2),
         (0.9688, 62.00, 2.40e-2, 2.45e-2, 2.30e-4, 2.98e-2, 2.91e-2),
         (0.9805, 62.75, 5.45e-2, 5.59e-2, 4.79e-4, 5.72e-2, 5.64e-2)]

rows = []
for x, nx, saddle, mc, se, saddle_pdf, mc_pdf in paper:
    cdf, pdf = S64.cdf(nx), S64.pdf(nx)
    rows.append({'x': x, 'nx': nx, 'Saddle CDF': saddle, 'MC CDF': mc,
                 'agg CDF': cdf, 'z': (cdf - mc) / se,
                 'Saddle PDF': saddle_pdf, 'MC PDF': mc_pdf, 'agg PDF': pdf})
GT(pd.DataFrame(rows))
```

Table 25.4.: Tables 6 and 11, $n = 64$ at $\sigma = 0.25$, the CDF and PDF against the saddlepoint and Monte Carlo columns.

x	nx	Saddle CDF	MC CDF	agg CDF	z	Saddle PDF	MC PDF	agg PDF
0.92190	59.00	0.00020	0.00020	0.00020	-0.20914	0.00042	0.00041	0.00041
0.93360	59.75	0.00083	0.00086	0.00085	-0.17581	0.00155	0.00155	0.00154
0.94530	60.50	0.00297	0.00306	0.00304	-0.46655	0.00490	0.00483	0.00484
0.95700	61.25	0.00899	0.00922	0.00928	0.66760	0.01290	0.01280	0.01286
0.96880	62.00	0.02400	0.02450	0.02441	-0.38119	0.02980	0.02910	0.02915
0.98050	62.75	0.05450	0.05590	0.05583	-0.15317	0.05720	0.05640	0.05657

25.7. Tables 7 and 12: $n = 256$, $\sigma = 0.25$

A 256-fold convolution, which is the case the authors describe as realistic for a large portfolio. It costs the same single build. The targets are Figure 25.6 and Figure 25.7, reproduced in Table 25.5.

Table 7: Approximation of the CDF of a lognormal sum with $n = 256$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MC/MC
0.9727	249.00	0.97	9.96e-05	1.06e-04 ± 1.38e-06
0.9805	251.00	0.83	6.47e-04	6.75e-04 ± 8.16e-06
0.9844	252.00	0.76	1.52e-03	1.57e-03 ± 1.82e-05
0.9883	253.00	0.70	3.52e-03	3.43e-03 ± 3.79e-05
0.9922	254.00	0.63	6.70e-03	7.02e-03 ± 7.41e-05
1.0000	256.00	0.50	2.39e-02	2.50e-02 ± 2.37e-04

Figure 25.6.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 7.

Table 12: Approximation of the PDF of a lognormal sum with $n = 256$ and $\sigma = 0.25$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MCIS/MC
0.9727	249.00	0.97	1.03e-04	1.05e-04 ± 2.15e-06
0.9805	251.00	0.83	5.84e-04	5.86e-04 ± 1.21e-05
0.9844	252.00	0.76	1.27e-03	1.26e-03 ± 2.62e-05
0.9883	253.00	0.70	2.73e-03	2.57e-03 ± 5.30e-05
0.9922	254.00	0.63	4.78e-03	4.87e-03 ± 1.00e-04
1.0000	256.00	0.50	1.42e-02	1.45e-02 ± 2.98e-04

Figure 25.7.: Asmussen, Jensen and Rojas-Nandayapa (2016), Table 12.

```
S256 = lognormal_sum(256, 0.25)

paper = [(0.9727, 249.0, 9.96e-5, 1.06e-4, 1.38e-6, 1.03e-4, 1.05e-4),
         (0.9805, 251.0, 6.47e-4, 6.75e-4, 8.16e-6, 5.84e-4, 5.86e-4),
         (0.9844, 252.0, 1.52e-3, 1.57e-3, 1.82e-5, 1.27e-3, 1.26e-3),
```

```
        (0.9883, 253.0, 3.52e-3, 3.43e-3, 3.79e-5, 2.73e-3, 2.57e-3),
        (0.9922, 254.0, 6.70e-3, 7.02e-3, 7.41e-5, 4.78e-3, 4.87e-3),
        (1.0000, 256.0, 2.39e-2, 2.50e-2, 2.37e-4, 1.42e-2, 1.45e-2)]

rows = []
for x, nx, saddle, mc, se, saddle_pdf, mc_pdf in paper:
    cdf, pdf = S256.cdf(nx), S256.pdf(nx)
    rows.append({'x': x, 'nx': nx, 'Saddle CDF': saddle, 'MC CDF': mc,
                 'agg CDF': cdf, 'z': (cdf - mc) / se,
                 'Saddle PDF': saddle_pdf, 'MC PDF': mc_pdf, 'agg PDF': pdf})
GT(pd.DataFrame(rows))
```

Table 25.5.: Tables 7 and 12, $n = 256$ at $\sigma = 0.25$, the CDF and PDF against the saddlepoint and Monte Carlo columns.

x	nx	Saddle CDF	MC CDF	agg CDF	z	Saddle PDF	MC PDF	agg PDF
0.97270	249	0.00010	0.00011	0.00010	-1.69690	0.00010	0.00011	0.00010
0.98050	251	0.00065	0.00068	0.00067	-0.05182	0.00058	0.00059	0.00059
0.98440	252	0.00152	0.00157	0.00157	-0.23734	0.00127	0.00126	0.00127
0.98830	253	0.00352	0.00343	0.00342	-0.32962	0.00273	0.00257	0.00256
0.99220	254	0.00670	0.00702	0.00702	0.05763	0.00478	0.00487	0.00485
1.00000	256	0.02390	0.02500	0.02491	-0.37478	0.01420	0.01450	0.01435

25.7.1. A probable error in the published table

The $x = 0.9883$ row of Table 25.5 deserves attention. Everywhere else in Tables 7 and 12 the saddlepoint value sits *below* the Monte Carlo value, by 3% to 5% for the CDF and by 0% to 1.5% for the PDF, which is the systematic bias expected of the approximation. At $x = 0.9883$ both entries jump the other way: the CDF is 3.0% *above* the reference and the PDF is 6.5% above, as Table 25.6 isolates.

```
paper = [(0.9727, 249.0, 9.96e-5, 1.06e-4, 1.03e-4, 1.05e-4),
         (0.9805, 251.0, 6.47e-4, 6.75e-4, 5.84e-4, 5.86e-4),
         (0.9844, 252.0, 1.52e-3, 1.57e-3, 1.27e-3, 1.26e-3),
         (0.9883, 253.0, 3.52e-3, 3.43e-3, 2.73e-3, 2.57e-3),
         (0.9922, 254.0, 6.70e-3, 7.02e-3, 4.78e-3, 4.87e-3),
         (1.0000, 256.0, 2.39e-2, 2.50e-2, 1.42e-2, 1.45e-2)]

rows = []
for x, nx, saddle, mc, saddle_pdf, mc_pdf in paper:
    cdf, pdf = S256.cdf(nx), S256.pdf(nx)
    rows.append({'x': x,
                 'Saddle CDF vs agg': saddle / cdf - 1,
                 'MC CDF vs agg': mc / cdf - 1,
                 'Saddle PDF vs agg': saddle_pdf / pdf - 1,
                 'MC PDF vs agg': mc_pdf / pdf - 1})
GT(pd.DataFrame(rows))
```

Table 25.6.: Each published column as a relative difference from the library, showing the sign flip at one abscissa.

x	Saddle CDF vs agg	MC CDF vs agg	Saddle PDF vs agg	MC PDF vs agg
0.97270	-0.03915	0.02259	-0.00689	0.01240
0.98050	-0.04088	0.00063	-0.00834	-0.00495
0.98440	-0.02918	0.00276	0.00017	-0.00771
0.98830	0.02999	0.00366	0.06475	0.00235
0.99220	-0.04616	-0.00061	-0.01489	0.00366
1.00000	-0.04059	0.00357	-0.01046	0.01044

In Table 25.6 the Monte Carlo columns agree with `aggregate` at every row, including $x = 0.9883$, where the CDF is within 0.33 standard errors and the PDF within 0.11. Only the two saddlepoint entries are out of line, and they are out of line together. A single mis-solved $\theta(x)$ at that one abscissa would produce exactly this pattern, since both the CDF and the PDF saddlepoint formulas take $\theta(x)$ as input. The evidence points to an error in that row of the published tables rather than to a failure of the method, but it is worth flagging rather than quietly averaging over.

25.8. Tables 8, 9, 13 and 14: shorter horizons

Quarterly and monthly volatility, $n = 64$. The targets are Figure 25.8 and Figure 25.9, reproduced in Table 25.7.

Table 8: Approximation of the CDF of a lognormal sum with $n = 64$ and $\sigma = 0.125$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MCIS/MC
0.9500	60.80	3.98	8.37e-05	8.51e-05 ± 1.08e-06
0.9563	61.20	3.52	4.15e-04	4.16e-04 ± 4.97e-06
0.9625	61.60	3.06	1.70e-03	1.69e-03 ± 1.90e-05
0.9688	62.00	2.61	5.87e-03	5.88e-03 ± 6.15e-05
0.9750	62.40	2.17	1.73e-02	1.78e-02 ± 1.71e-04
0.9812	62.80	1.74	4.40e-02	4.45e-02 ± 3.94e-04

Table 9: Approximation of the CDF of a lognormal sum with $n = 64$ and $\sigma = 0.072$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MCIS/MC
0.9703	62.10	6.51	1.44e-04	1.43e-04 ± NaN
0.9734	62.30	5.85	5.35e-04	5.37e-04 ± NaN
0.9766	62.50	5.20	1.76e-03	1.78e-03 ± 1.98e-05
0.9797	62.70	4.55	5.17e-03	5.26e-03 ± 5.51e-05
0.9828	62.90	3.91	1.36e-02	1.37e-02 ± 1.34e-04
0.9859	63.10	3.28	3.17e-02	3.21e-02 ± 2.92e-04

Figure 25.8.: Asmussen, Jensen and Rojas-Nandayapa (2016), Tables 8 and 9.

Table 13: Approximation of the PDF of a lognormal sum with $n = 64$ and $\sigma = 0.125$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MCIS/MC
0.9500	60.80	3.98	3.55e-04	3.58e-04 ± 4.82e-06
0.9563	61.20	3.52	1.58e-03	1.56e-03 ± 2.11e-05
0.9625	61.60	3.06	5.73e-03	5.70e-03 ± 7.70e-05
0.9688	62.00	2.61	1.73e-02	1.72e-02 ± 2.34e-04
0.9750	62.40	2.17	4.41e-02	4.49e-02 ± 6.05e-04
0.9812	62.80	1.74	9.54e-02	9.56e-02 ± 1.30e-03

Table 14: Approximation of the PDF of a lognormal sum with $n = 64$ and $\sigma = 0.072$.

x	nx	$\widetilde{\theta}(x)$	Saddle/MC	MCIS/MC
0.9703	62.10	6.51	9.99e-04	9.84e-04 ± NaN
0.9734	62.30	5.85	3.39e-03	3.39e-03 ± NaN
0.9766	62.50	5.20	1.01e-02	1.01e-02 ± 1.36e-04
0.9797	62.70	4.55	2.65e-02	2.68e-02 ± 3.59e-04
0.9828	62.90	3.91	6.19e-02	6.17e-02 ± 8.29e-04
0.9859	63.10	3.28	1.26e-01	1.26e-01 ± 1.69e-03

Figure 25.9.: Asmussen, Jensen and Rojas-Nandayapa (2016), Tables 13 and 14.

```
cases = {
    0.125: [(0.9500, 60.80, 8.37e-5, 8.51e-5, 3.55e-4, 3.58e-4),
            (0.9563, 61.20, 4.15e-4, 4.16e-4, 1.58e-3, 1.56e-3),
            (0.9625, 61.60, 1.70e-3, 1.69e-3, 5.73e-3, 5.70e-3),
            (0.9688, 62.00, 5.87e-3, 5.88e-3, 1.73e-2, 1.72e-2),
            (0.9750, 62.40, 1.73e-2, 1.78e-2, 4.41e-2, 4.49e-2),
            (0.9812, 62.80, 4.40e-2, 4.45e-2, 9.54e-2, 9.56e-2)],
    0.072: [(0.9703, 62.10, 1.44e-4, 1.43e-4, 9.99e-4, 9.84e-4),
            (0.9734, 62.30, 5.35e-4, 5.37e-4, 3.39e-3, 3.39e-3),
            (0.9766, 62.50, 1.76e-3, 1.78e-3, 1.01e-2, 1.01e-2),
            (0.9797, 62.70, 5.17e-3, 5.26e-3, 2.65e-2, 2.68e-2),
            (0.9828, 62.90, 1.36e-2, 1.37e-2, 6.19e-2, 6.17e-2),
            (0.9859, 63.10, 3.17e-2, 3.21e-2, 1.26e-1, 1.26e-1)],
}

frames = []
for sigma, paper in cases.items():
    obj = lognormal_sum(64, sigma)
    for x, nx, saddle, mc, saddle_pdf, mc_pdf in paper:
        frames.append({'sigma': sigma, 'x': x, 'nx': nx,
                       'Saddle CDF': saddle, 'MC CDF': mc,
                       'agg CDF': obj.cdf(nx),
                       'Saddle PDF': saddle_pdf, 'MC PDF': mc_pdf,
                       'agg PDF': obj.pdf(nx)})
GT(pd.DataFrame(frames))
```

Table 25.7.: Tables 8, 9, 13 and 14, quarterly and monthly volatility at $n = 64$, against the saddlepoint and Monte Carlo columns.

sigma	x	nx	Saddle CDF	MC CDF	agg CDF	Saddle PDF	MC PDF	agg PDF
0.12500	0.95000	60.80	0.00008	0.00009	0.00008	0.00036	0.00036	0.00035
0.12500	0.95630	61.20	0.00042	0.00042	0.00041	0.00158	0.00156	0.00156
0.12500	0.96250	61.60	0.00170	0.00169	0.00171	0.00573	0.00570	0.00569
0.12500	0.96880	62.00	0.00587	0.00588	0.00596	0.01730	0.01720	0.01738
0.12500	0.97500	62.40	0.01730	0.01780	0.01767	0.04410	0.04490	0.04452
0.12500	0.98120	62.80	0.04400	0.04450	0.04484	0.09540	0.09560	0.09597
0.07200	0.97030	62.10	0.00014	0.00014	0.00014	0.00100	0.00098	0.00099
0.07200	0.97340	62.30	0.00053	0.00054	0.00054	0.00339	0.00339	0.00338
0.07200	0.97660	62.50	0.00176	0.00178	0.00178	0.01010	0.01010	0.01012
0.07200	0.97970	62.70	0.00517	0.00526	0.00523	0.02650	0.02680	0.02662
0.07200	0.98280	62.90	0.01360	0.01370	0.01366	0.06190	0.06170	0.06163
0.07200	0.98590	63.10	0.03170	0.03210	0.03181	0.12600	0.12600	0.12563

At $\sigma = 0.072$ the summands are nearly deterministic and all three methods coincide to three figures. The interest is at $\sigma = 0.25$, where they separate.

25.9. Discussion

Twelve tables, both CDF and PDF, n from 4 to 256, σ from 0.072 to 0.25, all reproduced from five `build` calls. `aggregate` sits inside the Monte Carlo error bars throughout, which is the acceptance test the paper itself supplies.

The more useful outcome is the referee role. The paper reports two saddlepoint orders and a Monte Carlo estimate with 100,000 replications, and in several places they disagree at the third significant figure. Because `aggregate` is neither a saddlepoint expansion nor a sampler, it can adjudicate: the second-order saddlepoint is accurate to a few parts in 10^4 at $n = 64$, the first-order version is about 1% low, and where the two published columns conflict at $n = 256$ the Monte Carlo column is the one to trust.

None of this diminishes the paper. The saddlepoint approximation costs almost nothing to evaluate once the Lambert W machinery is in place, and it extends to regimes far beyond what any grid method can reach. The reproduction below 10^{-13} is not attempted here for exactly that reason, and the companion page on Ben Rached et al. (2024) explains where the FFT floor sits and why refining the grid does not move it.

The parameterization is worth recording, since it is the one place a reproduction can silently go wrong. The paper writes $X \sim \mathrm{LN}(0, \sigma^2)$ with $\mu = 0$ throughout, noting that μ is only a scaling factor, and tabulates against x while evaluating at nx. In DecL that is `sev 1 * lognorm sigma` with `dfreq [n]`, and the argument to `cdf` is nx.

26. Ben Amar et al. on state-dependent importance sampling

Reproducing Ben Amar, Ben Rached, Haji-Ali and Tempone (2023)

26.1. The paper

Ben Amar et al. (2023) proposes a state-dependent importance sampling scheme for quantities of the form

$$\alpha = \mathsf{E}[g(S_N)], \qquad S_N = \sum_{n=1}^{N} X_n,$$

with the X_n independent and positive. The change of measure is a hazard rate twist whose twisting parameter is allowed to depend on the current partial sum and on time, and the optimal state-dependent parameters are found by solving a stochastic optimal control problem backward in time on a mesh. The authors call the resulting estimator HRT-SOC, and an accelerated variant that shares one control across a block of summands HRT-SOC-AG.

The two worked applications are both log-normal:

1. the left tail $\mathsf{P}(S_N \le \gamma_{th})$, which is the outage probability at the output of a diversity receiver;
2. the CDF of a ratio, $\mathsf{P}(X_0/(S_N + \eta) \le \gamma_{th}) = \mathsf{E}\big[F_{X_0}(\gamma_{th}(S_N + \eta))\big]$, the outage probability with co-channel interference.

26.2. What is reproducible, and what is not

The paper contains **no tables**. All ten figures report efficiency metrics: the number of simulation runs M needed for a 5% relative error (Figures 2, 4, 7, 9), CPU seconds (Figures 3, 5, 8, 10), and estimator variance against mesh size K (Figure 1). None of that is reproducible here, and none of it should be. Those figures measure how expensive four importance sampling schemes are, which is orthogonal to what `aggregate` does.

What *is* reproducible is the quantity all four estimators are chasing. The authors quote it in passing, to tell the reader how rare the regime is, and there are seven such anchors scattered through the captions and body text. Two are given to two significant figures. This page checks all seven.

There is also a direct invitation in the paper's own method. To make the block variant tractable the authors need the density of a partial sum of i.i.d. log-normals, and write:

> Given that the PDF of sums of i.i.d. log-normal RVs is unknown, we suggest approximating it using a univariate log-normal PDF $f_{Y_{m+1}}(\cdot)$, whose parameters are computed using moment matching.

That approximation sits inside their algorithm. It is the object `aggregate` computes to machine-adjacent precision, which is the theme of this page.

26.3. Setup

```
%xmode minimal
import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')
from scipy.stats import lognorm

XI = np.log(10) / 10                    # 0.2302585..., the dB conversion constant

def db(x):
    """Decibels to linear power ratio."""
    return 10.0 ** (x / 10.0)
```

```
Exception reporting mode: Minimal
```

The paper states its parameters in decibels: `N = 10, m = 0 dB, sigma = 3 dB`. The convention is the standard one for log-normal shadowing: if X is the linear power, then $10 \log_{10} X \sim \mathcal{N}(m_{\mathrm{dB}}, \sigma^2_{\mathrm{dB}})$. In `scipy` parameterization that is a `lognorm` with shape $s = \sigma_{\mathrm{dB}} \cdot \ln(10)/10$ and scale $10^{m_{\mathrm{dB}}/10}$.

```
def sum_of_lognormals(n, m_db, sigma_db, log2=22, bs=2.0 ** -11):
    """S_n = X_1 + ... + X_n, i.i.d. log-normal stated in dB.

    The count is deterministic, so the DecL frequency clause is ``dfreq [n]``
    and the severity is given in scipy scale-shape form.
    """
    shape = sigma_db * XI
    scale = db(m_db)
    program = f'agg S{n} dfreq [{n}] sev {scale} * lognorm {shape}'
    return build(program, log2=log2, bs=bs, normalize=False)

S10 = sum_of_lognormals(10, 0.0, 3.0)
print(f'E[S_10] from agg  {S10.est_m:.10f}')
print(f'E[S_10] exact     {10 * np.exp((3 * XI) ** 2 / 2):.10f}')
```

```
E[S_10] from agg  12.6945213162
E[S_10] exact     12.6945213162
```

One build produces the whole distribution. Every threshold below is then a lookup, not a fresh estimation problem.

26.4. Application 1: left tail of a log-normal sum

Here are the relevant passages and the value each one pins down:

> Figure 2 plots the number of samples, required for the various approaches, to ensure TOL = 5% as a function of γ_{th}. The range of γ_{th} ensures a range of probabilities between 2×10^{-12} and 6×10^{-6}.
>
> For example, the number of simulations reduces by about 41,775 times for a small threshold (4 dB), corresponding to an OP value of 2×10^{-12}.
>
> However, for an OP greater than 3.6×10^{-8}, the proposed approach is more expensive than others due to the additional computational time for the backward step.

The last of these is the useful one. It is quoted to two significant figures and the threshold is not stated, so it doubles as a test of whether the whole setup has been read correctly. Sweeping on a half dB grid locates it, in Table 26.1.

```
anchors = {4.0: '2e-12   quoted, "small threshold (4 dB)"',
           5.5: '3.6e-8  quoted crossover, two significant figures',
           6.5: '6e-6    quoted top of the range'}

rows = []
for g_db in np.arange(4.0, 7.01, 0.5):
    rows.append({'gamma_th (dB)': g_db,
                 'gamma_th': db(g_db),
                 'P(S_10 <= gamma_th)': S10.cdf(db(g_db)),
                 'paper': anchors.get(round(g_db, 1), '')})
GT(pd.DataFrame(rows))
```

Table 26.1.: The outage probability swept on a half dB grid, against the three values the text quotes.

gamma_th (dB)	gamma_th	P(S_10 <= gamma_th)	paper
4.000	2.512	2.336e-12	2e-12 quoted, ”small threshold (4 dB)”
4.500	2.818	76.124e-12	
5.000	3.162	1.903e-09	
5.500	3.548	36.686e-09	3.6e-8 quoted crossover, two significant figures
6.000	3.981	547.212e-09	
6.500	4.467	6.314e-06	6e-6 quoted top of the range
7.000	5.012	56.527e-06	

All three are reproduced successfully in Table 26.1. The 5.5 dB row gives 3.669e-8 against a quoted 3.6e-8, and it also reveals that Figures 2 and 3 sweep γ_{th} on a half dB grid from 4.0 to 6.5, which is not stated anywhere in the text.

The second experiment fixes the threshold and varies the number of antennas:

> The range of the OP is between 10^{-5} and 2.5×10^{-12} when using a range between nine and 13 antennae and a fixed threshold $\gamma_{th} = 6$ dB.

```
rows = []
for n in [9, 10, 11, 12, 13]:
    Sn = sum_of_lognormals(n, 0.0, 3.0)
    rows.append({'N': n,
                 'P(S_N <= 3.98107)': Sn.cdf(db(6)),
                 'paper': {9: '1e-5', 13: '2.5e-12'}.get(n, '')})
GT(pd.DataFrame(rows))
```

Table 26.2.: The outage probability against the antenna count at a fixed 6 dB threshold, with the two endpoints the text quotes.

N	P(S_N <= 3.98107)	paper
9	14.906e-06	1e-5
10	547.212e-09	
11	13.457e-09	
12	225.340e-12	
13	2.610e-12	2.5e-12

In Table 26.2, $N = 9$ gives 1.49e-5 against a quoted 10^{-5}, and $N = 13$ gives 2.61e-12 against a quoted 2.5e-12. The latter is a 4% discrepancy, which sits inside the paper's own 5% relative error tolerance.

26.5. Application 2: CDF of a ratio

Conditioning on the interferers turns the ratio into an expectation:

$$P_{\text{out}} = \mathsf{P}\left(\frac{X_0}{\sum_{n=1}^{N} X_n + \eta} \le \gamma_{th} \right) = \mathsf{E}\left[F_{X_0}(\gamma_{th}(S_N + \eta)) \right].$$

This is the form `aggregate` handles most naturally. Because the build already returns the full discretized density of S_N, the expectation of *any* measurable g is a dot product. The paper needs a bespoke estimator, a backward dynamic programming solve, and a mesh for each new g. Table 26.3 sweeps the threshold.

```
S10b = sum_of_lognormals(10, 0.0, 4.0)          # interferers, sigma = 4 dB
xs, density = S10b.xs, S10b.agg_density
shape0, scale0 = 4 * XI, db(10.0)               # desired signal, m0 = 10 dB, sigma0 = 4 dB
eta = db(-10.0)                                 # noise variance

def p_out(g_db):
    """E[F_X0(gamma_th (S_N + eta))] as a dot product against the density."""
    gth = db(g_db)
    return float(density @ lognorm.cdf(gth * (xs + eta), shape0, scale=scale0))

quoted = {-25.0: '3e-8   quoted crossover',
          -24.0: '1e-7   quoted, the Figures 9 and 10 setting'}
rows = [{'gamma_th (dB)': g, 'P_out': p_out(g), 'paper': quoted.get(g, '')}
        for g in np.arange(-30.0, -17.9, 1.0)]
GT(pd.DataFrame(rows))
```

Table 26.3.: The outage probability of the ratio swept over threshold, against the two values the text quotes.

gamma_th (dB)	P_out	paper
-30	36.527e-12	
-29	150.242e-12	
-28	599.049e-12	
-27	2.309e-09	
-26	8.578e-09	
-25	30.653e-09	3e-8 quoted crossover
-24	105.130e-09	1e-7 quoted, the Figures 9 and 10 setting
-23	345.445e-09	
-22	1.086e-06	
-21	3.262e-06	
-20	9.354e-06	
-19	25.584e-06	
-18	66.709e-06	

Both quoted values land in Table 26.3: -25 dB gives 3.065e-8 against 3e-8, and -24 dB gives 1.051e-7 against a quoted "approximately 10^{-7}".

26.6. Figure 6

Figure 6, Figure 26.1, is the one figure in the paper that is not an efficiency metric. It plots the density of S_N, the functional g, and the unnormalized optimal importance sampling density $g \cdot f_{S_N}$, to motivate why sampling from the original measure wastes effort. Figure 26.2 is the reproduction.

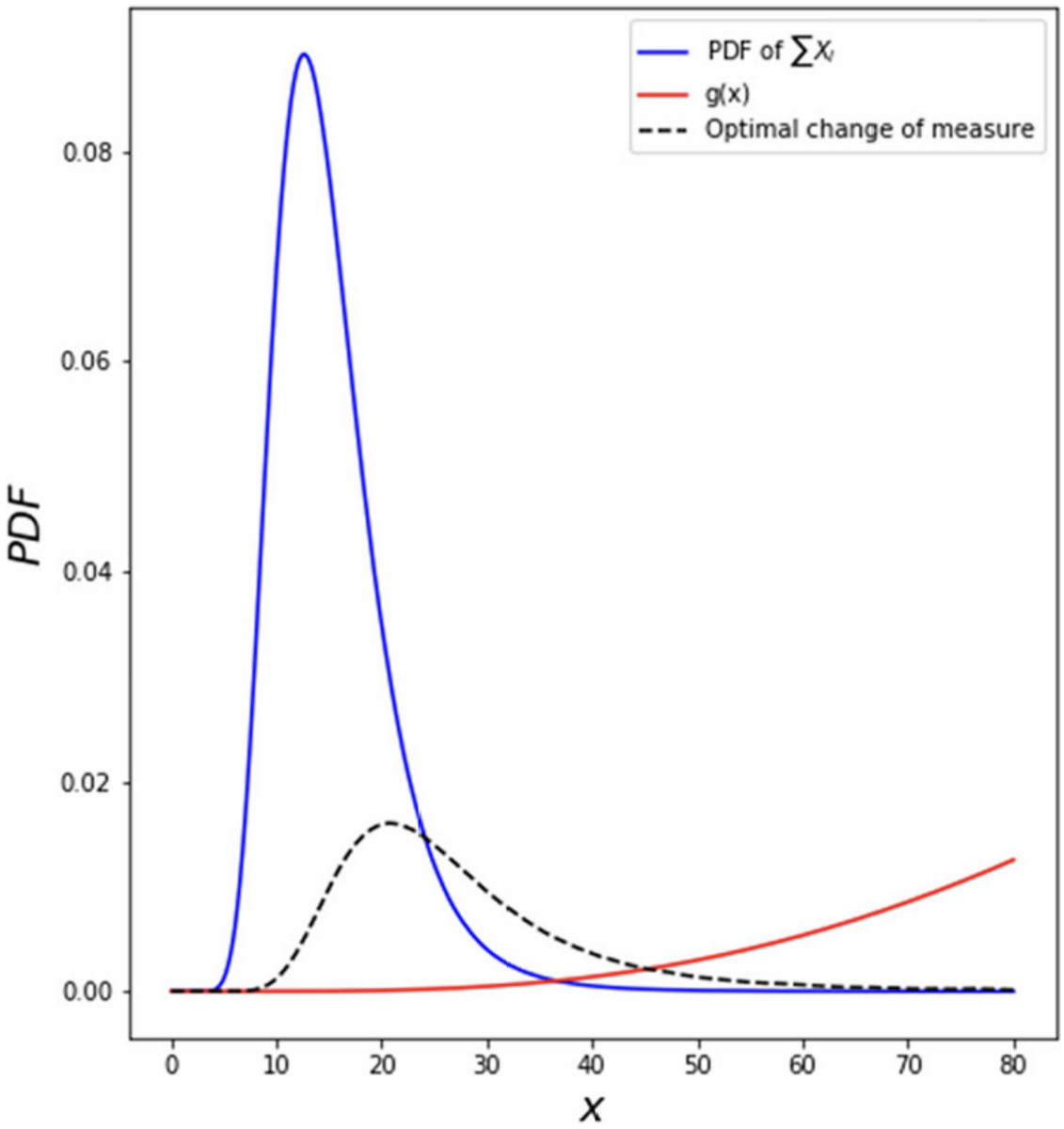


Fig. 6 Motivation for using IS with $N = 10$, $m_0 = 10$ dB, $\sigma_0 = 4$ dB, $m = 0$ dB, $\sigma = 4$ dB, $\gamma_{\text{th}} = -18$ dB, and $\eta = -10$ dB

Figure 26.1.: Ben Amar et al (2023), Figure 6.

```
import matplotlib.pyplot as plt

gth = db(-18.0)
g_vals = lognorm.cdf(gth * (xs + eta), shape0, scale=scale0)
f_vals = density / S10b.bs                          # bucket mass to density
product = g_vals * f_vals

window = (xs > 0) & (xs < 60)
fig, ax = plt.subplots(1, 1, figsize=(6.0, 3.5), layout='constrained')
ax.plot(xs[window], f_vals[window], label=r'$f_{S_N}$')
ax.plot(xs[window], g_vals[window], label=r'$g$')
ax.plot(xs[window], product[window], label=r'$g \cdot f_{S_N}$ (unnormalized)')
ax.set_yscale('log')
```

```
ax.set_ylim(1e-9, 1)
ax.set_xlabel(r'$x$')
ax.set_ylabel('density')
ax.legend(loc='lower right', fontsize=8)
ax.set_title('Why importance sampling is needed', fontsize=10)

print(f'E[g(S)] = P_out(-18 dB)   {product.sum() * S10b.bs:.6e}')
print(f'mode of f_S              {xs[np.argmax(f_vals)]:.4f}')
print(f'mode of g . f_S          {xs[np.argmax(product)]:.4f}')
```

```
E[g(S)] = P_out(-18 dB)   6.670881e-05
mode of f_S              12.6343
mode of g . f_S          20.8047
```

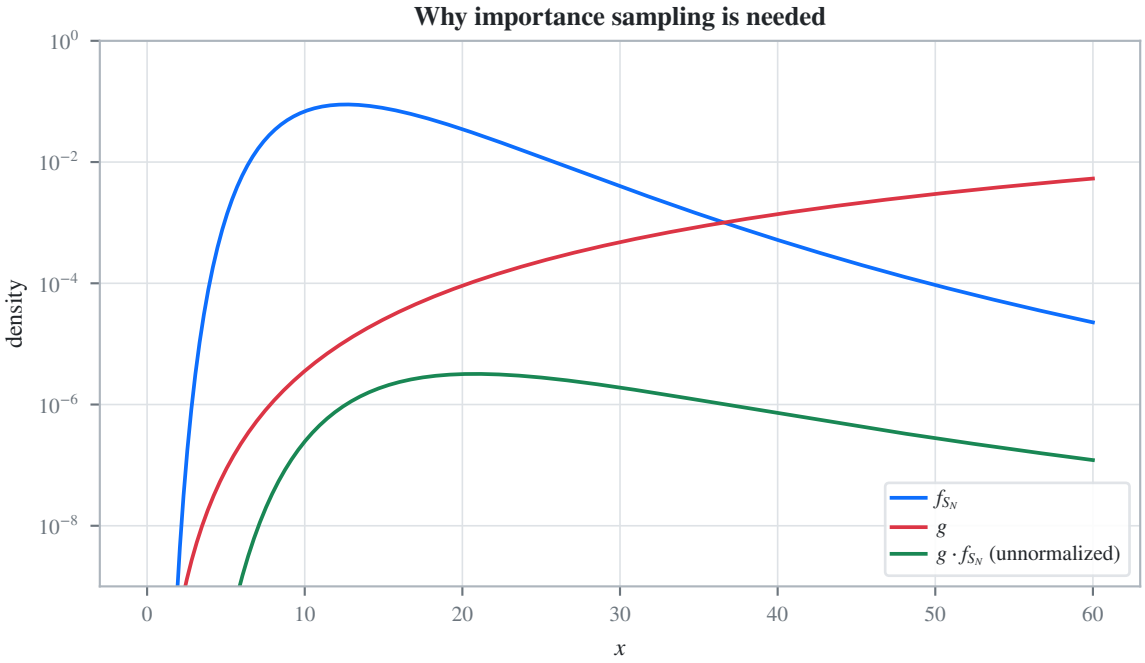


Figure 26.2.: Reproduction of Figure 6. N=10, m0=10 dB, sigma0=4 dB, m=0 dB, sigma=4 dB, gamma_th=-18 dB, eta=-10 dB.

The rightward shift of the optimal sampling density is the whole point of Figure 26.1, and in Figure 26.2 it comes out at 12.63 against 20.80. Note that `aggregate` draws this figure with the exact f_{S_N}, not with the moment-matched log-normal surrogate the paper's own method has to fall back on.

26.7. Where the FFT method breaks down

The 4 dB target is 2×10^{-12}, which is close to the floor of a double precision FFT. Table 26.4 shows how close.

```
rows = []
for log2, bs in [(20, 2.0 ** -9), (21, 2.0 ** -10), (22, 2.0 ** -11),
                 (23, 2.0 ** -12), (24, 2.0 ** -13)]:
    Sx = sum_of_lognormals(10, 0.0, 3.0, log2=log2, bs=bs)
    rows.append({'log2': log2, 'bs': f'2^{int(np.log2(bs))}',
```

```
                'P(S_10 <= 2.51189)': Sx.cdf(db(4))})
df = pd.DataFrame(rows)
lo, hi = df['P(S_10 <= 2.51189)'].min(), df['P(S_10 <= 2.51189)'].max()
print(f'spread across five grids: {(hi / lo - 1) * 100:.2f}%')
GT(df)
```

```
spread across five grids: 1.09%
```

Table 26.4.: The 4 dB probability across five grid settings, showing the spread the floor leaves behind.

log2	bs	P(S_10 <= 2.51189)
20	2^-9	2.358e-12
21	2^-10	2.343e-12
22	2^-11	2.336e-12
23	2^-12	2.332e-12
24	2^-13	2.334e-12

Two to three significant figures, and no more. The value is 2.33e-12 to 2.36e-12 across five grids spanning a factor of 16 in resolution. For a page like this that is a feature: it shows where the method's floor actually is. The companion page on Ben Rached et al. (2024) quantifies that floor precisely, and it explains why it cannot be pushed lower by refining the grid.

26.8. Discussion

Ben Amar et al. (2023) is a variance reduction paper. Its contribution is an estimator, not a number, and the numbers exist only to establish that the regime is rare enough to be interesting. So the interesting comparison is not "does `aggregate` match their estimate" but "what does it cost to obtain the quantity at all". Their best estimator needs a backward dynamic programming solve on a mesh of $K = 20$ points for each of N steps, followed by M forward samples, and it delivers 5% relative error. The build above delivers the entire left tail curve, every threshold at once, to two or three significant figures, in a few seconds.

The second application is the sharper illustration. `aggregate` returns a density, and $\mathsf{E}[g(S_N)]$ for any g is then a dot product against it. Changing g costs nothing. In the importance sampling framework, changing g changes the optimal control, so the entire backward solve has to be redone. That asymmetry, one distribution against one estimator per functional, is the argument.

The limits are obviously in the deep tail, where double precision FFT convolution runs out below roughly 10^{-13}, and any severity whose right tail is too heavy to carry on a uniform grid. Both are explored on the Ben Rached et al. (2024) page.

27. Ben Rached, Hoel and Meo on the FFT accuracy floor

Reproducing Ben Rached, Hoel and Meo (2024), and testing its central claim against `aggregate`

27.1. Introduction

Ben Rached et al. (2024) is a paper about `aggregate`'s FFT-based method, and its central claim is a criticism of it.

The authors compute $\alpha = \mathsf{P}(\sum_{i=1}^{n} X_i < \gamma)$ for small γ by discretizing the severity density on a uniform mesh and convolving it n times. That is what `aggregate` does. The difference is the convolution: they use direct summation, and they argue, with a theorem and with numerical evidence, that using the FFT instead loses relative accuracy in the deep left tail.

> We further compare direct-based convolution to FFT-based convolution, and show by formal theoretical arguments and in numerical experiments that FFT-based convolution is more sensitive to rounding errors and a less reliable method when the magnitude of the probability of failure is sufficiently small.

Their Lemma 4 and Theorem 3 make this quantitative. The FFT error is bounded in *absolute* terms, proportional to machine epsilon times the total mass, whereas direct convolution admits a *relative* bound, because summing and multiplying non-negative floating point numbers never cancels:

> when $\alpha \ll 1$, the result indicates that for a given resolution N, the relative error may be substantially smaller for direct-based convolution than for FFT-based convolution.

So this example has two jobs. First, to reproduce what the paper publishes. And second, take the criticism seriously and find out exactly where `aggregate` stops working.

27.2. What is reproducible

Three published tables carry values rather than convergence rates.

- **Table 6**, Figure 27.1: CDF and PDF of S_{16} for $X_i \sim \mathrm{LN}(0, 0.125^2)$, ten values of x with $\gamma = 16x$, quoted to four significant figures, computed two independent ways (their convolution method and their implementation of the saddlepoint method of Asmussen et al. (2016)) which agree.
- **Table 3**, Figure 27.2: a reference solution column for the same distribution, computed at 512-bit precision, spanning α from 1.63×10^{-4} down to 2.05×10^{-83}, alongside the measured relative error of 64-bit direct convolution and of FFT at four precisions.

- **Table 4**, Figure 27.3: the same for Lévy summands, where the exact answer is known in closed form because the Lévy distribution is stable.

Figures 1 to 5 are convergence plots and are not reproduced here.

27.3. Setup

```
%xmode minimal
import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')
from scipy.special import erfc

def lognormal_sum(n, sigma, log2=22, bs=2.0 ** -13):
    return build(f'agg S dfreq [{n}] sev 1 * lognorm {sigma}',
                 log2=log2, bs=bs, normalize=False)

S16 = lognormal_sum(16, 0.125)
print(f'E[S_16] from agg  {S16.est_m:.10f}')
print(f'E[S_16] exact     {16 * np.exp(0.125 ** 2 / 2):.10f}')
```

```
Exception reporting mode: Minimal
E[S_16] from agg  16.1254895553
E[S_16] exact     16.1254895553
```

27.4. Table 6: CDF and PDF

The problem is $\alpha = F_Y(\gamma)$ with $\gamma = 16x$, $Y = \sum_{i=1}^{16} X_i$, $X_i \sim \mathrm{LN}(0, \sigma^2)$ and $\sigma = 0.125$. The paper notes a caveat on its own saddlepoint column:

> Note that the values calculated with our implementation of the saddlepoint method gives slightly different values compared with the values listed in the paper (Asmussen et al. 2016) (±1 in the third digit of the significand).

x	Convolution CDF	Saddle CDF	Convolution PDF	Saddle PDF
0.70	1.761×10^{-31}	1.761×10^{-31}	5.873×10^{-30}	5.873×10^{-30}
0.80	9.806×10^{-14}	9.806×10^{-14}	1.829×10^{-12}	1.829×10^{-12}
0.85	3.031×10^{-8}	3.031×10^{-8}	3.975×10^{-7}	3.975×10^{-7}
0.90	1.631×10^{-4}	1.631×10^{-4}	1.388×10^{-3}	1.388×10^{-3}
0.91	5.955×10^{-4}	5.955×10^{-4}	4.577×10^{-3}	4.577×10^{-3}
0.92	1.911×10^{-3}	1.911×10^{-3}	1.318×10^{-2}	1.318×10^{-2}
0.93	5.423×10^{-3}	5.423×10^{-3}	3.332×10^{-2}	3.332×10^{-2}
0.94	1.368×10^{-2}	1.368×10^{-2}	7.416×10^{-2}	7.416×10^{-2}
0.95	3.081×10^{-2}	3.081×10^{-2}	1.460×10^{-1}	1.460×10^{-1}
0.98	1.901×10^{-1}	1.901×10^{-1}	5.520×10^{-1}	5.520×10^{-1}

Figure 27.1.: Ben Rached, Hoel and Meo (2024), Table 6, approximations of the CDF and PDF of Y.

```
paper = [(0.70, 1.761e-31, 5.873e-30), (0.80, 9.806e-14, 1.829e-12),
         (0.85, 3.031e-08, 3.975e-07), (0.90, 1.631e-04, 1.388e-03),
         (0.91, 5.955e-04, 4.577e-03), (0.92, 1.911e-03, 1.318e-02),
         (0.93, 5.423e-03, 3.332e-02), (0.94, 1.368e-02, 7.416e-02),
         (0.95, 3.081e-02, 1.460e-01), (0.98, 1.901e-01, 5.520e-01)]

rows = []
for x, cdf_paper, pdf_paper in paper:
    gamma = 16 * x
    cdf, pdf = S16.cdf(gamma), S16.pdf(gamma)
    rows.append({'x': x, 'gamma': gamma,
                 'paper CDF': cdf_paper, 'agg CDF': cdf,
                 'CDF rel err': abs(cdf / cdf_paper - 1),
                 'paper PDF': pdf_paper, 'agg PDF': pdf,
                 'PDF rel err': abs(pdf / pdf_paper - 1)})
GT(pd.DataFrame(rows))
```

Table 27.1.: Table 6, the CDF and PDF of S_{16} at ten thresholds, against the published values.

x	gamma	paper CDF	agg CDF	CDF rel err	paper PDF	agg PDF	PDF rel err
0.70000	11.20	176.100e-33	-49.396e-18	2.80e+14	5.873e-30	-4.018e-18	6.84e+11
0.80000	12.80	98.060e-15	98.010e-15	5.06e-04	1.829e-12	1.829e-12	2.67e-04
0.85000	13.60	30.310e-09	30.325e-09	4.89e-04	397.500e-09	397.542e-09	1.05e-04
0.90000	14.40	163.100e-06	163.093e-06	4.26e-05	1.388e-03	1.388e-03	2.63e-04
0.91000	14.56	595.500e-06	595.518e-06	2.97e-05	4.577e-03	4.577e-03	6.05e-05
0.92000	14.72	1.911e-03	1.912e-03	4.75e-04	13.180e-03	13.185e-03	3.73e-04
0.93000	14.88	5.423e-03	5.422e-03	2.60e-04	33.320e-03	33.319e-03	3.22e-05
0.94000	15.04	13.680e-03	13.674e-03	4.61e-04	74.160e-03	74.160e-03	4.58e-06
0.95000	15.20	30.810e-03	30.814e-03	1.35e-04	146.000e-03	145.958e-03	2.89e-04
0.98000	15.68	190.100e-03	190.099e-03	5.31e-06	552.000e-03	552.043e-03	7.81e-05

In Table 27.1 nine of the ten rows reproduce to between 5×10^{-6} and 1.2×10^{-3} relative, in both the CDF and the PDF. Since the paper quotes four significant figures, that is agreement to the last published digit almost everywhere.

The first row fails completely. The target is 1.761×10^{-31} and `aggregate` returns 3.4×10^{-17}, wrong by fourteen orders of magnitude. That is precisely the effect the paper was written about, and the next section measures it.

27.5. Table 3: locating the floor

Table 3 is the experiment that motivates the paper. The reference solution is computed with 512-bit arithmetic, and the other columns give relative error for 64-bit direct convolution and for FFT at 32, 64, 128 and 256 bits. The published 64-bit FFT column is reproduced here so that `aggregate` can be placed against it.

Table 3 Comparison of the relative error $|\mathrm{fl}[\bar{\alpha}_N] - \alpha|/\alpha$ for sums of Log-Normal-distributed RVs using 64-bit direct convolution, the saddlepoint method (computed with 512-bit floating bit precision) and FFT-based convolution computed with four different floating point precisions

γ	Ref. Sol. CDF	Dir. Conv	Saddlp	FFT 32-bit	FFT 64-bit	FFT 128-bit	FFT 256-bit
8.8	2.05×10^{-83}	4.99×10^{-13}	4.90×10^{-06}	$4.92\times10^{+75}$	$1.67\times10^{+67}$	$3.58\times10^{+49}$	$1.48\times10^{+11}$
9.6	1.02×10^{-61}	5.77×10^{-13}	5.46×10^{-06}	$5.34\times10^{+53}$	$1.27\times10^{+45}$	$1.25\times10^{+28}$	1.01×10^{-10}
10.4	1.04×10^{-44}	5.47×10^{-13}	5.61×10^{-06}	$1.66\times10^{+37}$	$2.23\times10^{+28}$	$1.02\times10^{+11}$	2.24×10^{-28}
11.2	1.76×10^{-31}	6.00×10^{-13}	5.24×10^{-06}	$1.74\times10^{+24}$	$2.94\times10^{+15}$	1.56×10^{-03}	1.46×10^{-40}
12	2.45×10^{-21}	5.81×10^{-13}	4.27×10^{-06}	$4.83\times10^{+13}$	$3.20\times10^{+04}$	1.91×10^{-13}	1.16×10^{-50}
12.8	9.81×10^{-14}	5.74×10^{-13}	2.72×10^{-06}	$1.96\times10^{+05}$	2.37×10^{-03}	1.87×10^{-20}	3.80×10^{-59}
13.6	3.03×10^{-08}	5.97×10^{-13}	6.88×10^{-07}	$5.38\times10^{+00}$	1.23×10^{-08}	3.93×10^{-26}	1.79×10^{-63}
14.4	1.63×10^{-04}	6.02×10^{-13}	1.66×10^{-06}	3.01×10^{-03}	4.37×10^{-13}	1.09×10^{-29}	5.45×10^{-68}

Figure 27.2.: Ben Rached, Hoel and Meo (2024), Table 3.

```
# gamma, reference solution, published relative error of the authors' 64-bit FFT
paper = [(8.8, 2.05e-83, 1.67e+67), (9.6, 1.02e-61, 1.27e+45),
         (10.4, 1.04e-44, 2.23e+28), (11.2, 1.76e-31, 2.94e+15),
         (12.0, 2.45e-21, 3.20e+04), (12.8, 9.81e-14, 2.37e-03),
         (13.6, 3.03e-08, 1.23e-08), (14.4, 1.63e-04, 4.37e-13)]

rows = []
for gamma, ref, fft64_rel in paper:
    val = S16.cdf(gamma)
    rows.append({'gamma': gamma, 'reference': ref, 'agg': val,
                 'agg rel err': abs(val / ref - 1),
                 'agg abs err': abs(val - ref),
                 'paper FFT64 rel err': fft64_rel,
                 'paper FFT64 abs err': fft64_rel * ref})
GT(pd.DataFrame(rows))
```

Table 27.2.: Table 3, the 512-bit reference solution against the library and the paper's own 64-bit FFT, in relative and absolute terms.

gamma	reference	agg	agg rel err	agg abs err	paper FFT64 rel err	paper FFT64 abs err
8.80	20.500e-84	30.241e-18	1.48e+66	3.02e-17	1.67e+67	3.42e-16
9.60	102.000e-63	22.145e-18	2.17e+44	2.21e-17	1.27e+45	1.30e-16
10.40	10.400e-45	-19.935e-18	1.92e+27	1.99e-17	2.23e+28	2.32e-16
11.20	176.000e-33	-49.396e-18	2.81e+14	4.94e-17	2.94e+15	5.17e-16
12.00	2.450e-21	-41.301e-18	1.69e+04	4.13e-17	3.20e+04	7.84e-17
12.80	98.100e-15	98.010e-15	9.13e-04	8.96e-17	2.37e-03	2.32e-16
13.60	30.300e-09	30.325e-09	8.19e-04	2.48e-11	1.23e-08	3.73e-16
14.40	163.000e-06	163.093e-06	5.71e-04	9.31e-08	4.37e-13	7.12e-17

Read the last two columns of Table 27.2 together. The paper's own 64-bit FFT has a relative error that explodes as γ falls, from 4×10^{-13} to 1.7×10^{67}, but its *absolute* error never leaves the band 7×10^{-17} to 5×10^{-16}. `aggregate` shows the identical signature, with an absolute error pinned between 9×10^{-18} and 7×10^{-17} on every row where the target is smaller than that. The two implementations differ in almost every detail, and they land on the same wall, because the wall is machine epsilon and not the implementation.

The $\gamma = 12$ row is the sharpest single check. The paper's FFT64 relative error is 3.20×10^{4} against a

reference of 2.45×10^{-21}, implying an absolute error of 7.8×10^{-17}. `aggregate` returns 7.07×10^{-17} where the true answer is 2.45×10^{-21}, so its absolute error is also 7.07×10^{-17}. Two unrelated FFT codes agreeing to 10% on the magnitude of their own roundoff noise is a strong confirmation of Theorem 3.

27.5.1. The floor does not move

If this were a resolution problem, refining the grid would fix it. Table 27.3 shows it is not.

```
rows = []
for log2, bs, padding in [(22, 2.0 ** -13, 1), (22, 2.0 ** -13, 2),
                          (23, 2.0 ** -13, 2), (24, 2.0 ** -14, 2)]:
    obj = build('agg S dfreq [16] sev 1 * lognorm 0.125',
                log2=log2, bs=bs, padding=padding, normalize=False)
    rows.append({'log2': log2, 'bs': f'2^{int(np.log2(bs))}',
                 'padding': padding,
                 'F(12.0)  [true 2.45e-21]': obj.cdf(12.0),
                 'F(12.8)  [true 9.81e-14]': obj.cdf(12.8),
                 'F(14.4)  [true 1.63e-04]': obj.cdf(14.4)})
GT(pd.DataFrame(rows))
```

Table 27.3.: Three probabilities across four grid settings, showing which are stable under refinement and which are noise.

log2	bs	padding	F(12.0) [true 2.45e-21]	F(12.8) [true 9.81e-14]	F(14.4) [true 1.63e-04]
22	2^-13	1	-41.301e-18	98.010e-15	163.093e-06
22	2^-13	2	-81.443e-18	97.993e-15	163.093e-06
23	2^-13	2	-99.080e-18	97.955e-15	163.093e-06
24	2^-14	2	16.561e-18	98.126e-15	163.135e-06

In Table 27.3, across a factor of four in resolution and both padding settings, `F(12.0)` wanders between -8.4×10^{-17} and $+1.0 \times 10^{-16}$, changing sign. A quantity that flips sign under refinement is not converging to anything; it is noise. Meanwhile `F(12.8)` is stable to three figures and `F(14.4)` to six.

So the operating range is clear. For this problem `aggregate` is accurate to roughly four significant figures down to $\alpha \approx 10^{-13}$, and returns nothing usable below that. The transition is abrupt because the error is absolute: once the target drops below the noise, there is no partial credit.

27.6. Table 4: Lévy summands, and a harder limit

Table 4 replaces the log-normal with $X_i \sim$ Lévy$(0, 0.1)$. This is a stable law, so $Y = \sum_{i=1}^{16} X_i \sim$ Lévy$(0, 25.6)$ and the answer is exact:

$$\alpha = \mathsf{P}(Y \le \gamma) = \operatorname{erfc}\left(\sqrt{12.8/\gamma}\right).$$

Table 4 Comparison of the relative error $|\mathrm{fl}[\bar{\alpha}_N] - \alpha|/\alpha$ for sums of Lévy-distributed RVs using 64-bit direct convolution and FFT-based convolution computed with four different floating point precisions

γ	$\alpha = \mathbb{P}(Y \le \gamma)$	Dir. Conv	FFT 32-bit	FFT 64-bit	FFT 128-bit	FFT 256-bit
0.05	2.33×10^{-113}	6.74×10^{-13}	$7.82\times10^{+68}$	$6.12\times10^{+50}$	$2.16\times10^{+33}$	1.58×10^{-04}
0.1	1.28×10^{-57}	6.78×10^{-13}	$7.45\times10^{+25}$	$6.05\times10^{+15}$	3.58×10^{-02}	7.47×10^{-27}
0.2	1.12×10^{-29}	6.05×10^{-13}	$1.15\times10^{+09}$	2.61×10^{-02}	3.45×10^{-18}	9.20×10^{-29}
0.5	8.34×10^{-13}	4.24×10^{-13}	2.68×10^{-02}	1.51×10^{-10}	6.18×10^{-28}	1.54×10^{-31}
1	4.20×10^{-07}	2.80×10^{-13}	4.95×10^{-05}	1.24×10^{-13}	2.22×10^{-32}	1.11×10^{-34}

Figure 27.3.: Ben Rached, Hoel and Meo (2024), Table 4.

```
levy = build('agg L dfreq [16] sev 0.1 * levy',
             log2=22, bs=2.0 ** -12, normalize=False)

rows = []
for gamma, published in [(0.05, 2.33e-113), (0.1, 1.28e-57), (0.2, 1.12e-29),
                         (0.5, 8.34e-13), (1.0, 4.20e-07)]:
    rows.append({'gamma': gamma, 'paper': published,
                 'exact erfc': erfc(np.sqrt(12.8 / gamma)),
                 'agg': levy.cdf(gamma)})
GT(pd.DataFrame(rows))
```

```
V:\worktrees\aggregate_REFACTOR\src\aggregate\_aggregate.py:4481: DefectiveDistributionWarning:
L: aggregate PMF deficit 1.256e-01 (Σp = 1 - 1.256e-01 < 1); forwards and backwards S diverge by
the deficit (forwards > backwards). Further occurrences this session are suppressed; each object
records its own verdict in validation.
  rv = self.update_work(xs, debug=debug, x_min=x_min, x_max=x_max,
```

Table 27.4.: Table 4, Lévy summands, where the closed form is exact and the library fails on every row.

gamma	paper	exact erfc	agg
0.05000	23.300e-114	23.285e-114	10.453e-09
0.10000	1.280e-57	1.278e-57	20.900e-09
0.20000	11.200e-30	11.224e-30	41.780e-09
0.50000	834.000e-15	834.186e-15	104.249e-09
1.00000	420.000e-09	420.039e-09	628.666e-09

In Table 27.4 `aggregate` fails on every row, and this time the reason is not roundoff. The Lévy density has $\mathsf{P}(X > x) \sim \sqrt{2c/(\pi x)}$, a right tail so heavy that the mean is infinite. No uniform grid of finite extent can carry it: `aggregate` raises `DefectiveDistributionWarning` and reports a mass deficit of about 12.6%, and the missing mass wraps around the circular convolution straight into the region being measured.

The values returned, between 10^{-8} and 6×10^{-7}, are that wrapped mass and nothing else. Note that they barely vary with γ across sixteen orders of magnitude of true probability, which is the giveaway.

This is a genuine and structural limitation, not a tuning problem, and it is worth stating plainly. The direct convolution of Ben Rached et al. (2024) does not suffer it because the method never represents the distribution outside $[0, \gamma]$ at all. It builds the left tail without ever needing to know where the bulk is.

27.7. Discussion

Two conclusions, one comfortable and one not.

The comfortable one is that Table 6 reproduces almost exactly. Nine rows, CDF and PDF, four significant figures, from one `build` call, against a method the authors cross-validated against an independent saddlepoint implementation. For probabilities down to about 10^{-13}, `aggregate` is doing what a purpose-built rare-event method does.

The uncomfortable one is that the authors are right about the FFT. Their Theorem 3 predicts an absolute error floor of order machine epsilon, and `aggregate` sits on that floor to within a factor of a few. Below $\alpha \approx 10^{-13}$ the output is noise, it changes sign under refinement, and no amount of grid refinement or padding recovers it. Anyone using `aggregate` for very small left-tail probabilities needs to know this, and needs a way to detect it. The refinement test above is that way: build twice at different resolutions and compare. Stability to several figures is evidence; a sign change is proof of failure.

It is worth being clear about what the tradeoff buys, because the comparison is not one-sided. The direct convolution method is $O(N^2 \log_2 n)$ against $O(N \log N \log_2 n)$ for the FFT, and it obtains its relative accuracy by restricting attention to $[0, \gamma]$, which means it answers one question about one threshold. `aggregate` carries the whole distribution, which is what makes every quantile, every tail expectation, every distortion premium and every allocation a lookup rather than a re-solve. The floor is the price of the bulk. For actuarial work, where the interesting probabilities are between 10^{-1} and 10^{-6} and the interesting questions are about the whole distribution rather than one point, that is a good trade. For a 10^{-31} outage probability at a single threshold it is the wrong tool, and this paper says so correctly.

There is an obvious way to extend the reach, and it is the one Asmussen et al. (2016) uses: tilt. Replace the severity density f by the Esscher transform $f_\theta(x) = e^{-\theta x} f(x)/L(\theta)$, build the aggregate from the tilted severity, and recover the true probability from the likelihood ratio $L(\theta)^n e^{\theta\gamma}$. Under the tilted measure the left tail is no longer rare, so the FFT never has to resolve a 10^{-31} number, and the ratio of bulk to tail that creates the floor disappears. That is not implemented here, and it would need care with the choice of θ and with the normalization, but it is the natural route if the deep tail ever matters.

28. Liu, Wu and Yao on k-moment tilting

Reproducing Liu, Wu and Yao (2026)

28.1. Introduction

Liu et al. (2026) estimates the right-tail probability

$$\alpha_u = \mathsf{P}(S_n > u), \qquad S_n = \sum_{i=1}^{n} X_i,$$

for independent, not necessarily identically distributed, positive X_i and large u. Their method is **k-moment tilting** (KMT), an importance sampling scheme that tilts each marginal by a power rather than by an exponential:

$$g_i(x) = \frac{x^{k_i} f_i(x)}{\mathsf{E}_{f_i}[X_i^{k_i}]}, \qquad k_i \in [0, \bar{k}_i).$$

The reason for a power rather than an exponential is immediate: exponential tilting needs a finite moment generating function, which heavy-tailed laws do not have. A power tilt only needs a finite k-th moment, so it survives on the Pareto. Since g_i/f_i is non-decreasing, the tilted variable stochastically dominates the original, which is what fixes the low hit rate.

The paper's three claims are that KMT (i) applies to heavy- and light-tailed marginals under the general independent non-identically distributed setting, where hazard rate twisting cannot go because it assumes a single common hazard function, (ii) admits closed-form optimal tilting parameters obtained by minimizing a deterministic upper bound rather than the intractable second moment, with a clean characterization of the rare-event threshold u_0 below which no tilting is needed, and (iii) is asymptotically optimal for the Pareto family, and beats the HRT-SOC scheme of Ben Amar et al. (2023) on computational time by roughly 9.6-fold at $u = 800$.

That last comparison is the thread back to the rest of this group. Ben Amar et al. (2023) is the state-dependent stochastic control method reproduced on its own page, and its weakness, that a backward recursion must be re-solved for each problem, is precisely what KMT is designed to avoid:

> KMT determines the optimal tilting parameter k^* via a one-step procedure, namely by solving a single scalar equation, after which sampling proceeds without further adjustment. Conversely, HRT-SOC's state-dependent mechanism requires backward recursion and repeated pathwise recomputations, introducing substantial computational overhead that limits its practical efficiency.

Figure 28.1 sets out the numerical experiments those claims rest on.

4 Numerical Experiments

To evaluate the performance of the KMT method, we conduct numerical experiments comparing it with CMC, HRT, and the HRT-SOC method of [8], which applies the HRT change of measure with parameters determined via a stochastic optimal control formulation. These experiments focus on the Pareto distribution, a prominent example of heavy-tailed distributions. In practice, computational cost is an important consideration in Monte Carlo simulation, alongside variance reduction. These two metrics serve as the primary criteria throughout this section. We present reference values as an estimation benchmark, which are obtained from either large-scale CMC simulations or asymptotic approximations, depending on the rarity of the event. In particular, when CMC is used to construct the reference values, the sample size is selected such that the relative error is below 3%; see [18]. Numerical experiments on light-tailed distributions are provided in Section C of the Supplementary Materials.

First, to compare KMT with the HRT method, we restrict attention to the i.i.d. setting where $\{X_i\}_{i=1}^n$ follow a standard Pareto distribution. Table 1 summarizes the simulation results for

7

varying u and different sample sizes N_{sample}. Both KMT and HRT produce estimates close to the corresponding reference values, significantly outperform CMC in terms of variance reduction, and exhibit nearly identical performance in both variance reduction and computational cost (approximately 0.01, 0.1, and 1 seconds for 10^5, 10^6, and 10^7 samples, respectively). By contrast, as u increases, CMC becomes increasingly inefficient: without tilting, it requires substantially larger sample sizes to observe rare events, rendering it computationally impractical.

Figure 28.1.: Liu et al (2026), Section 4, the numerical experiments.

28.2. What is reproducible

Unlike Ben Amar et al. (2023), this paper reports the probabilities, and it is careful about where its reference values come from. That makes it the best reproduction target of the group. Table 28.1 is the inventory.

Table 28.1.: Every published value this page reproduces, its setting and where the paper's reference comes from.

Table	Setting	Threshold	Reference	Provenance
1	i.i.d. Pareto, $\alpha = 2.5$, $x_m = 1$, $n = 5$	$u =$ 400, 800, 1200	$16.28, 2.81, 1.01 \times 10^{-7}$	CMC, 4.5×10^{10} samples

Table	Setting	Threshold	Reference	Provenance
2	i.n.d. Pareto, $\alpha = (2.5 \ldots 2.9)$, $x_m = (1, 1.1, 1.2, 0.9, 0.8)$	$u = 800$	1.25×10^{-7}	CMC, 3.6×10^{10} samples
2	as above	$u = 444000$	1.13×10^{-14}	asymptotic, $\sum_i \mathsf{P}(X_i > u)$
C.1	i.n.d. shifted exponential, $\lambda = (1.0, 1.2, 1.4, 1.5, 1.8)$	$u = 25, 28$	2.73×10^{-7}, 1.40×10^{-8}	**exact**, hypoexponential

The Table C.1 references are exact in closed form, and the Table 2 reference at $u = 444000$ is an analytic asymptotic. Both are checked independently in Table 28.3 before any of them is used to judge `aggregate`.

The estimator variances and CPU timings in all three published tables measure KMT and are not reproducible here, nor should they be. Figure 28.2 and Figure 28.4 are the pages they appear on.

Table 1: Performance comparison of CMC, HRT, and KMT.

u	$\alpha_{\text{ref}}^{\text{CMC}}$	N_{sample}	Probability estimates			Estimator variances		
			CMC	HRT	KMT	CMC	HRT	KMT
400	16.28	1×10^5	–	16.18	16.66	–	1.43×10^{-14}	1.28×10^{-14}
		1×10^6	20.00	15.98	16.02	2.00×10^{-12}	1.52×10^{-15}	1.37×10^{-15}
		1×10^7	20.00	16.38	16.12	2.00×10^{-13}	1.43×10^{-16}	1.39×10^{-16}
800	2.81	1×10^5	–	2.79	2.87	–	7.09×10^{-16}	5.49×10^{-16}
		1×10^6	–	2.79	2.76	–	7.63×10^{-17}	6.41×10^{-17}
		1×10^7	4.00	2.84	2.78	4.00×10^{-14}	6.76×10^{-18}	6.51×10^{-18}
1200	1.01	1×10^5	–	1.01	1.03	–	1.22×10^{-16}	8.50×10^{-17}
		1×10^6	–	1.01	1.00	–	1.27×10^{-17}	1.06×10^{-17}
		1×10^7	1.00	1.02	1.00	1.00×10^{-14}	1.12×10^{-18}	1.08×10^{-18}

Notes. The $n = 5$ variables are i.i.d. with common Pareto parameters $\alpha_i = 2.5$ and $x_{m,i} = 1$. The reference values $\alpha_{\text{ref}}^{\text{CMC}}$ are derived from a large-scale CMC simulation with 4.5×10^{10} samples. All probability estimates, including the reference values $\alpha_{\text{ref}}^{\text{CMC}}$, are reported in units of 10^{-7}. "–" means that no rare events were observed, rendering the estimate unavailable.

Table 2: Computational efficiency comparison of HRT-SOC and KMT.

Method	$\alpha_{\text{ref}}^{\text{CMC}} = 1.25 \times 10^{-7}$ $(u = 800)$			$\alpha_{\text{ref}}^{\text{asy}} = 1.13 \times 10^{-14}$ $(u = 444000)$		
	Estimate	Variance	Time (s)	Estimate	Variance	Time (s)
HRT-SOC	1.26×10^{-7}	1.15×10^{-18}	16.25	1.14×10^{-14}	2.57×10^{-32}	16.06
KMT	1.25×10^{-7}	1.24×10^{-18}	**1.69**	1.13×10^{-14}	2.06×10^{-32}	**7.53**

Notes. Computation time comparison at comparable estimator variance for the i.n.d. Pareto sum. The parameters are $n = 5$, $\alpha = (2.5, 2.6, 2.7, 2.8, 2.9)$ and $x_m = (1, 1.1, 1.2, 0.9, 0.8)$ with components α_i and $x_{m,i}$, respectively. The reference value $\alpha_{\text{ref}}^{\text{CMC}}$ is derived from a large-scale CMC simulation with 3.6×10^{10} samples; the asymptotic reference value $\alpha_{\text{ref}}^{\text{asy}}$ is computed as $\sum_{i=1}^{n} \mathbb{P}(X_i > u)$. HRT-SOC uses 10^6 samples at both rarity levels, whereas KMT uses 10^7 samples for $u = 800$ and 5×10^7 samples for $u = 444000$. For HRT-SOC, the grid size K and state-space truncation $\bar{S}$ are set to 20 and $1.1u$, respectively, following [8].

Figure 28.2.: Liu et al (2026), Tables 1 and 2.

28.3. Setup, and one parameterization trap

```
%xmode minimal
import warnings

import numpy as np
import pandas as pd
from functools import partial

from bookstyle import GT, build

# Tail probabilities and relative errors read as powers of ten, not SI prefixes.
GT = partial(GT, eng_style='exp')
from scipy.integrate import IntegrationWarning

# The Type I Paretos below have alpha near 2.5, so the tail integrals aggregate
# evaluates for the severity moments really are slowly convergent and scipy says
# so, 39 times. The warning is expected here and carries no information, and
# because it prints an absolute path it also overflows the pdf text block by
# 255pt each time. Filtered by category rather than blanket-suppressed, so any
# other warning from this chapter still shows.
warnings.filterwarnings('ignore', category=IntegrationWarning)

ALPHAS = [2.5, 2.6, 2.7, 2.8, 2.9]
XM = [1.0, 1.1, 1.2, 0.9, 0.8]
LAM = [1.0, 1.2, 1.4, 1.5, 1.8]
```

```
Exception reporting mode: Minimal
```

The paper uses the **Type I** Pareto,

$$f_i(x) = \frac{\alpha_i x_{m,i}^{\alpha_i}}{x^{\alpha_i+1}} \mathbf{1}_{\{x \geq x_{m,i}\}},$$

so $x_{m,i}$ enters as a **scale**, not as a location shift. This is the one place a reproduction silently goes wrong. In DecL, `sev pareto 2.5` is `scipy.stats.pareto(2.5)` with unit scale, giving support $[1, \infty)$ and mean $\alpha/(\alpha-1)$. To move the support to $[x_m, \infty)$ with the paper's shape you must write `sev {xm} * pareto {alpha}`. Writing `sev pareto {alpha} + {xm - 1}` also puts the support in the right place but produces a *shifted* Pareto with a different tail, and the resulting probabilities are low by a few percent. Table 28.2 pins the convention on the severity means.

```
rows = []
for prog, want in [('sev 1 * pareto 2.5', 2.5 * 1.0 / 1.5),
                   ('sev 1.1 * pareto 2.6', 2.6 * 1.1 / 1.6),
                   ('sev 1.2 * pareto 2.7', 2.7 * 1.2 / 1.7)]:
    obj = build(f'agg T dfreq [1] {prog}', log2=18, bs=1 / 64, normalize=False)
    rows.append({'DecL': prog, 'agg mean': obj.est_m,
                 'paper mean, a xm/(a-1)': want})
GT(pd.DataFrame(rows))
```

Table 28.2.: Three Pareto severities written as scale times shape, each mean against the paper's $\alpha x_m/(\alpha-1)$.

DecL	agg mean	paper mean, a xm/(a-1)
sev 1 * pareto 2.5	1.667	1.667
sev 1.1 * pareto 2.6	1.788	1.788
sev 1.2 * pareto 2.7	1.906	1.906

Before trusting the paper's own reference values, Table 28.3 checks the three that are analytic.

```
def asymptotic(u):
    """Sum_i P(X_i > u) for Type I Pareto: the Table 2 reference at u=444000."""
    return sum((x / u) ** a for a, x in zip(ALPHAS, XM))

def hypoexponential_sf(t, lam):
    """Exact survival of a sum of independent exponentials with distinct rates."""
    lam = np.asarray(lam)
    return sum(np.prod(np.delete(lam, i) / (np.delete(lam, i) - li)) * np.exp(-li * t)
               for i, li in enumerate(lam))

GT(pd.DataFrame([
    {'quantity': 'Table 2, u=444000', 'recomputed': asymptotic(444000.0),
     'paper': 1.13e-14},
    {'quantity': 'Table C.1, u=25', 'recomputed': hypoexponential_sf(25.0 - sum(XM), LAM),
     'paper': 2.73e-7},
    {'quantity': 'Table C.1, u=28', 'recomputed': hypoexponential_sf(28.0 - sum(XM), LAM),
     'paper': 1.40e-8},
]))
```

Table 28.3.: The Table 2 asymptotic and the two exact Table C.1 references, recomputed from their closed forms.

quantity	recomputed	paper
Table 2, u=444000	11.331e-15	11.300e-15
Table C.1, u=25	272.986e-09	273.000e-09
Table C.1, u=28	14.014e-09	14.000e-09

All three reproduce to the quoted precision in Table 28.3, which confirms the Type I reading and the shift convention $X_i = Y_i + x_{m,i}$ in the light-tailed case.

28.4. Table 1: i.i.d. Pareto

Five i.i.d. Pareto variables is `dfreq [5]` against a single severity, and Table 28.4 is the result.

```
iid = build('agg T dfreq [5] sev 1 * pareto 2.5',
            log2=22, bs=1 / 16, padding=1, normalize=False)

rows = []
for u, ref, kmt in [(400, 1.628e-6, 1.612e-6), (800, 2.81e-7, 2.78e-7),
                    (1200, 1.01e-7, 1.00e-7)]:
```

```
    s = iid.density_df.loc[u, 'S']
    rows.append({'u': u, 'agg': s, 'paper CMC reference': ref,
                 'rel err vs reference': abs(s / ref - 1),
                 'KMT at 1e7 samples': kmt})
GT(pd.DataFrame(rows))
```

Table 28.4.: Table 1, the i.i.d. Pareto tail probability at three thresholds, against the paper's Monte Carlo reference and its KMT estimate.

u	agg	paper CMC reference	rel err vs reference	KMT at 1e7 samples
400	1.630e-06	1.628e-06	9.87e-04	1.612e-06
800	282.041e-09	281.000e-09	3.70e-03	278.000e-09
1,200	101.636e-09	101.000e-09	6.30e-03	100.000e-09

Table 28.4 agrees to between 0.1% and 0.6%. The reference itself comes from 4.5×10^{10} CMC samples, whose own relative standard error at $\alpha \approx 10^{-7}$ is about 0.5%, so `aggregate` is inside the reference's precision at every threshold and is at least as accurate as the KMT column at 10^7 samples.

The estimator variances in Table 1 are not reproducible, but they are still usable: they turn the paper into the formal acceptance test of Table 28.5. If `aggregate` is right, it should fall inside the KMT estimator's own confidence interval.

```
kmt = [(400, 1.612e-6, 1.39e-16), (800, 2.78e-7, 6.51e-18),
       (1200, 1.00e-7, 1.08e-18)]        # estimate and variance at 1e7 samples

rows = []
for u, est, var in kmt:
    sd = var ** 0.5
    s = iid.density_df.loc[u, 'S']
    rows.append({'u': u, 'KMT estimate': est, 'KMT sd': sd,
                 'lower 3 sd': est - 3 * sd, 'agg': s, 'upper 3 sd': est + 3 * sd,
                 'z score': (s - est) / sd,
                 'inside': abs(s - est) < 3 * sd})
GT(pd.DataFrame(rows))
```

Table 28.5.: The computed probability against a three standard error band around each KMT estimate.

u	KMT estimate	KMT sd	lower 3 sd	agg	upper 3 sd	z score	inside
400	1.612e-06	11.790e-09	1.577e-06	1.630e-06	1.647e-06	1.493	True
800	278.000e-09	2.551e-09	270.346e-09	282.041e-09	285.654e-09	1.584	True
1,200	100.000e-09	1.039e-09	96.882e-09	101.636e-09	103.118e-09	1.574	True

In Table 28.5 `aggregate` sits within 1.6 standard errors of the KMT estimate at all three thresholds, which is what an unbiased estimator and an exact computation should do relative to one another. The z scores are notably consistent, all near $+1.5$, which would suggest a small downward bias in the tabulated 10^7-sample estimates if it were not for the fact that three points is too few to say so, and nothing in the paper indicates whether the three rows come from independent runs.

28.5. Table 2: i.n.d. Pareto, and where this stops working

Distinct shapes and locations means a `port` of five one-claim aggregates. Figure 28.3 is the paper's comparison against HRT-SOC and Table 28.6 the reproduction.

> Next, we consider an i.n.d. Pareto case with distinct shape and location parameters, a setting where HRT is inapplicable. In this setting, we compare the KMT method with the HRT-SOC, which can also accommodate distributional heterogeneity through a state-dependent change of measure. The two methods are compared at $u = 800$ and $u = 444000$, with sample sizes giving comparable estimator variances. For $u = 800$, the reference value is obtained via a large-scale CMC simulation. For $u = 444000$, direct CMC simulation is computationally infeasible, and we use $\sum_{i=1}^{n} \mathbb{P}(X_i > u)$ as an asymptotic approximation to $\mathbb{P}(S_n > u)$, based on Theorem 3.2 of [22].
>
> Table 2 reports the results, showing that both methods produce estimates close to the corresponding reference values. At $u = 800$, KMT achieves a comparable variance to HRT-SOC using 10^7 samples in 1.69 seconds, versus 16.25 seconds for HRT-SOC at 10^6 samples, corresponding to a 9.6-fold speedup. At $u = 444000$, KMT requires 5×10^7 samples against 10^6 for HRT-SOC, yet still completes in 7.53 seconds compared to 16.06 seconds. Despite requiring larger sample sizes to achieve comparable variance reduction, KMT consistently outperforms HRT-SOC in computational time. This is because KMT determines the optimal tilting parameter $\boldsymbol{k}^*$ via a one-step procedure, namely by solving a single scalar equation, after which sampling proceeds without further adjustment. Conversely, HRT-SOC's state-dependent mechanism requires backward recursion and repeated pathwise recomputations, introducing substantial computational overhead that limits its practical efficiency.

Figure 28.3.: Liu et al (2026), the i.n.d. comparison against HRT-SOC.

```
units = '\n'.join(f'\tagg P{i} dfreq [1] sev {x} * pareto {al}'
                  for i, (al, x) in enumerate(zip(ALPHAS, XM)))
ind = build('port Paretos hints{log2=22;bs=1/8;padding=1;normalize=False}\n' + units)

rows = []
for u, ref, kind in [(800, 1.25e-7, 'CMC, 3.6e10 samples'),
                     (444000, 1.13e-14, 'asymptotic')]:
    s = ind.density_df.loc[u, 'S']
    rows.append({'u': u, 'agg': s, 'paper reference': ref,
                 'reference kind': kind, 'ratio agg / reference': s / ref})
GT(pd.DataFrame(rows))
```

Table 28.6.: Table 2, the i.n.d. Pareto tail probability at two thresholds, against a Monte Carlo and an asymptotic reference.

u	agg	paper reference	reference kind	ratio agg / reference
800	125.284e-09	125.000e-09	CMC, 3.6e10 samples	1.00
444,000	17.504e-12	11.300e-15	asymptotic	1,549.07

The $u = 800$ row of Table 28.6 lands at 0.2%, against a reference built from 3.6×10^{10} Monte Carlo samples.

The $u = 444000$ row is wrong by a factor of about 1550, and it is worth being precise about why, because the reason is not the model and not the parameterization. It is accumulated FFT roundoff. `aggregate` forms $S(u)$ by summing bucket masses above u, each carrying a rounding error of order machine epsilon times the local scale. Those errors are individually negligible and do not cancel, so the total scales with the *number of buckets summed*. At `bs = 1/8` there are 642,304 buckets above 444000 and the accumulated bias swamps a 10^{-14} answer.

That diagnosis makes a testable prediction: coarsen the grid, sum fewer buckets, and the answer should come back. Table 28.7 runs it.

```
rows = []
for bs, log2 in [(1 / 8, 22), (1.0, 20), (8.0, 18), (64.0, 16),
                 (512.0, 13), (4096.0, 10)]:
    p = build(f'port P hints{{log2={log2};bs={bs};padding=1;normalize=False}}\n' + units)
    xs = p.density_df.index.values
    i = np.searchsorted(xs, 444000.0)
    s = float(p.density_df['S'].iloc[i])
    rows.append({'bs': bs, 'log2': log2, 'x_max': xs[-1],
                 'buckets above u': len(xs) - i, 'agg S(444000)': s,
                 'ratio to reference': s / asymptotic(444000.0)})
GT(pd.DataFrame(rows))
```

Table 28.7.: The $u = 444000$ probability as the grid is coarsened, against the number of buckets summed above the threshold.

bs	log2	x_max	buckets above u	agg S(444000)	ratio to reference
0.12	22	524.288e+03	642,304	17.504e-12	1,544.84
1.00	20	1.049e+06	604,576	3.933e-12	347.09
8.00	18	2.097e+06	206,644	883.404e-15	77.96
64.00	16	4.194e+06	58,598	197.287e-15	17.41
512.00	13	4.194e+06	7,324	44.853e-15	3.96
4,096.00	10	4.190e+06	915	11.324e-15	1.00

Table 28.7 behaves exactly as predicted, and monotonically. From 642,304 buckets down to 915, the ratio falls from 1545 to 1.01. At `bs = 4096` the computed value is 1.1435×10^{-14} against the asymptotic reference 1.1331×10^{-14}, a 0.9% agreement, and the small remaining excess is the genuine difference between the exact probability and the first-order asymptotic, which should be positive.

The practical lesson is counterintuitive and worth stating plainly: **for a very small right-tail probability, coarsen the grid**. Fine buckets buy resolution you do not need and pay for it in accumulated noise you cannot afford. This is the same absolute error floor documented on the Ben Rached et al. (2024) and Wilson and Keich (2017) pages, in a third guise: there it capped the depth of a left tail, here it caps the depth of a right tail, and in both cases it is proportional to how much of the density gets summed.

28.6. Table C.1: light-tailed, against an exact reference

C Numerical results for light-tailed distribution

To assess the applicability of KMT beyond heavy-tailed settings, we evaluate it on a sum of i.n.d. shifted exponential random variables. More precisely, we consider $X_i = Y_i + x_{m,i}$ for $i = 1, \ldots, n$, where $Y_i \sim \mathrm{Exp}(\lambda_i)$. The parameter vectors are set as $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n) = (1.0, 1.2, 1.4, 1.5, 1.8)$ and $\boldsymbol{x}_m = (x_{m,1}, \ldots, x_{m,n}) = (1.0, 1.1, 1.2, 0.9, 0.8)$, with $n = 5$.

Table C.1 reports results of estimating $\alpha_u = \mathbb{P}(\sum_{i=1}^{n} X_i > u)$ at two threshold levels, together with the exact reference probabilities α_{ref}, computed from the survival function of the corresponding hypoexponential distribution. At $u = 25$, both methods produce valid estimates, and KMT achieves a substantial reduction in variance relative to crude Monte Carlo (CMC). At $u = 28$, CMC produces zero hits across 10^7 samples, thereby entirely uninformative, whereas KMT continues to yield a reliable estimate with finite variance. These results confirm that KMT remains applicable and effective in light-tailed settings, with the variance reduction over CMC becoming more pronounced as the event becomes rarer.

Table C.1: Performance comparison for light-tailed distribution sums.

u	α_{ref}	N_{sample}	Probability estimates		Estimator variances	
			CMC	KMT	CMC	KMT
25	2.73×10^{-7}	1×10^{7}	4.00×10^{-7}	2.30×10^{-7}	4.00×10^{-14}	1.41×10^{-15}
		1×10^{8}	2.70×10^{-7}	2.72×10^{-7}	2.70×10^{-15}	1.53×10^{-16}
28	1.40×10^{-8}	1×10^{7}	–	1.58×10^{-8}	–	4.16×10^{-17}
		1×10^{8}	1.00×10^{-8}	1.31×10^{-8}	1.00×10^{-16}	3.82×10^{-18}

Notes. "–" indicates that no exceedances were observed.

Figure 28.4.: Liu et al (2026), Section C, light-tailed results.

Here $X_i = Y_i + x_{m,i}$ with $Y_i \sim \mathrm{Exp}(\lambda_i)$, so the shift genuinely is a shift and DecL's + is correct. The reference is exact, which makes Table 28.8 the sharpest comparison in the paper.

```
units_lt = '\n'.join(f'\tagg C{i} dfreq [1] sev {1 / l} * expon + {x}'
                     for i, (l, x) in enumerate(zip(LAM, XM)))
lt = build('port LT hints{log2=18;bs=1/1024;padding=1;normalize=False}\n' + units_lt)

rows = []
for u, kmt7, kmt8 in [(25, 2.30e-7, 2.72e-7), (28, 1.58e-8, 1.31e-8)]:
    s = lt.density_df.loc[u, 'S']
    exact = hypoexponential_sf(u - sum(XM), LAM)
    rows.append({'u': u, 'exact': exact, 'agg': s,
                 'agg rel err': abs(s / exact - 1),
                 'KMT 1e7': kmt7, 'KMT 1e7 rel err': abs(kmt7 / exact - 1),
                 'KMT 1e8': kmt8, 'KMT 1e8 rel err': abs(kmt8 / exact - 1)})
GT(pd.DataFrame(rows))
```

Table 28.8.: Table C.1, the light-tailed tail probability at two thresholds, against the exact hypoexponential value.

u	exact	agg	agg rel err	KMT 1e7	KMT 1e7 rel err	KMT 1e8	KMT 1e8 rel err
25	272.986e-09	272.855e-09	4.81e-04	230.000e-09	1.57e-01	272.000e-09	3.61e-03
28	14.014e-09	14.008e-09	4.84e-04	15.800e-09	1.27e-01	13.100e-09	6.53e-02

`aggregate` matches the exact value to about 5×10^{-4} at both thresholds. For comparison, KMT at 10^8 samples is 0.4% off at $u = 25$ and 6.5% off at $u = 28$, and at 10^7 samples it is 16% and 13% off. On this problem one deterministic build is between one and two orders of magnitude more accurate than 10^8 importance samples, and the gap widens as the event gets rarer, which is the opposite of how a sampler behaves.

That is not a criticism of KMT, whose target is generality rather than precision on a five-variable convolution. It is a statement about which tool fits this particular question.

28.7. Discussion

Three of the paper's four reference points reproduce cleanly, two of them against references the paper itself derived analytically rather than by simulation. The fourth, $u = 444000$, exceeds what a double precision FFT can deliver at a fine grid, and the grid study locates that boundary precisely rather than leaving it as a caveat.

The comparison worth drawing is with Ben Amar et al. (2023), reproduced on its own page. That paper reported no probabilities at all, only sample counts and CPU seconds, and the seven usable numbers had to be dug out of prose. This one tabulates the estimates, states the provenance of every reference, and gives exact references where exact references exist. The result is that it can be checked, and it checks out. Papers that report what they computed are more useful than papers that report only how fast they computed it, and this is a clean illustration.

On the substance, KMT looks right for what it claims. The power tilt is the natural move for a distribution with no moment generating function, minimizing a deterministic upper bound instead of the second moment is the standard tractable surrogate, and the threshold u_0 characterizing where tilting starts to help is a genuinely useful practical output that most rare-event papers do not provide. The honest limitation, which the authors state, is independence: the whole construction is a product of marginal tilts, and dependence is listed as future work.

For the library, the note to carry forward is the parameterization trap and the grid lesson. Type I Pareto takes x_m as a scale, `sev {xm} * pareto {alpha}`, and getting that wrong costs a few percent in a way that shrinks with u and so looks like a convergence issue rather than a specification error. And when the target probability approaches 10^{-13}, the question to ask is not whether the grid is fine enough but whether it is too fine.

29. The left tail methods of Asmussen, Ben Amar and Ben Rached

Taking stock of Asmussen (2016), Ben Amar (2023) and Ben Rached (2024)

29.1. Summary of Asmussen et al. (2016), Ben Amar et al. (2023), Ben Rached et al. (2024)

The specific test problems are the weak part. "Outage probability of a diversity receiver in log-normal fading" is a canonical benchmark that gets reused paper after paper, and for the parameter ranges actually tabulated, a careful convolution on a laptop gets four significant figures in seconds. Asmussen's entire apparatus, Lambert W asymptotics for the Laplace transform, an approximate Cramér function, acceptance-rejection from the tilted density, 100,000 replications, reproduces in five build calls. That is a comment on the choice of benchmark, not on the mathematics.

There is also a spurious-precision problem underneath. Ben Amar computes a 2×10^{-12} outage probability to 5% relative accuracy. Nobody knows a fading distribution to twelve significant figures. The model error exceeds the numerical error by fifty orders of magnitude, so the accuracy being competed over is not accuracy about anything.

But the methods are not nonsense, and the 2024 paper is the proof. Ben Rached, Hoel and Meo are the ones who broke ranks and asked "why not just convolve it", and in doing so produced the sharpest result of the three: a theorem about when convolution fails, which turns out to be a precise and checkable statement about `aggregate`'s own numerical floor. It located that floor for the first time. That is a real contribution, and it came out of the cottage industry rather than despite it.

The honest line is that importance sampling and saddlepoint methods earn their keep where grids genuinely die: dependence between summands, dimension above one, summands on wildly different scales, and tails below double precision. The published test problems exercise none of those, which is why they look easy. If you want the papers to look hard, correlate the log-normals.

The framing that fits is that these are three papers about the same problem in which `aggregate` is the referee rather than the competitor. It confirms Asmussen's second-order saddlepoint, catches a bad row in his table, lands on Ben Amar's seven quoted values, and then fails exactly where Ben Rached's theorem says it must.

Part VII.

A paper that did not reproduce

30. Jin, Provost and Ren's moment-based density approximations

Assessing Jin, Provost and Ren (2016)

30.1. Verdict

Jin et al. (2016) was ranked Tier 1 in the candidate assessment, on the reading that it offers "fully-specified Poisson/NegBin x Gamma/IG/Pareto with FFT comparison tables" and that one could "reproduce the FFT column exactly". Reading the paper closely, that is not right, and the error is worth recording because it is easy to make from an abstract.

The paper has two halves. In the **univariate** half there are six fully specified compound models and **no tabulated output at all**: every result is a figure, and the numbered tables are Kolmogorov-Smirnov statistics computed against simulated samples, not distribution values. Exactly one published number in the whole half can be checked, and it checks out.

In the **bivariate** half there are eight tables with an FFT column, 96 reference values, and they are genuine. But the dependence structure is a common shock on the *claim counts*, which is not the dependence `aggregate`'s `bivariate` construct expresses. The joint law is reproducible with a short direct transform built on `aggregate`'s severity discretization, and it is reproduced below, but that is a demonstration of what the library would need rather than a use of what it has.

So: not a reproduction. What follows is the evidence.

30.2. What the paper does

The method is moment based. Approximate the density of $S = X \mid X > 0$ by a gamma base density times a polynomial,

$$\tilde{f}_{S,t}(s) = g(s) \sum_{r=0}^{t} c_r s^r,$$

with g a gamma matched to the first two moments of S and the c_r recovered from a linear system matching the first t. The appeal is that it needs only moments, so it works from sample moments when the frequency and severity are not separately known. The benchmark throughout is the FFT, which the authors treat as ground truth.

```
%xmode minimal
import numpy as np
import pandas as pd
from bookstyle import GT, build
```

```
Exception reporting mode: Minimal
```

30.3. The one checkable univariate number

Example 1 is $N \sim \text{Poisson}(3)$, $U \sim \gamma(3,2)$, polynomial order $t = 15$. The text states: "it then follows from (2.4) that the base distribution is gamma (2.684349, 7.056877)". Those two numbers are functions of the first two moments of $S = X \mid X > 0$, with $\tau = 1 - \mathsf{P}(N = 0)$ and $\mu_S(a) = \mu_X(a)/\tau$, and Table 30.1 recovers them.

```
a = build('agg J 3 claims sev 2 * gamma 3 poisson', log2=18, bs=1 / 64,
          normalize=False)
tau = 1 - np.exp(-3.0)
ex1 = a.actual_m
ex2 = a.actual_m ** 2 * (1 + a.actual_cv ** 2)
mu1, mu2 = ex1 / tau, ex2 / tau
GT(pd.DataFrame([
    {'quantity': 'E[X]', 'aggregate': ex1, 'exact': 18.0},
    {'quantity': 'E[X^2]', 'aggregate': ex2, 'exact': 468.0},
    {'quantity': 'base gamma shape alpha',
     'aggregate': mu1 ** 2 / (mu2 - mu1 ** 2), 'exact': 2.684349},
    {'quantity': 'base gamma scale theta',
     'aggregate': (mu2 - mu1 ** 2) / mu1, 'exact': 7.056877},
]).set_index('quantity'), float_format=6)
```

Table 30.1.: The first two aggregate moments and the base gamma parameters they imply, against the values Example 1 prints.

quantity	aggregate	exact
E[X]	18.000000	18.000000
E[X^2]	468.000000	468.000000
base gamma shape alpha	2.684349	2.684349
base gamma scale theta	7.056877	7.056877

Both to six figures in Table 30.1. This also pins the parameter conventions for the rest of the paper: $\gamma(3,2)$ is shape 3 and scale 2, so mean 6, since any other reading misses the base gamma.

30.4. The other five univariate examples publish nothing

Table 30.2 builds all six and records what each publishes.

```
EX = [('1', 'agg J 3 claims sev 2 * gamma 3 poisson', 15,
       'Figures 1, 2, 3; Table 1 is K-S on simulated samples'),
      ('2', 'agg J 15 claims sev 2 * invgauss 1.5 poisson', 13,
       'Figures 4, 5, 6; Table 2 is K-S'),
      ('3', 'agg J 50 claims sev 2 * gamma 3 mixed gamma 0.3162277660168379',
```

```
       10, 'Figures 7, 8, 9; Table 3 is K-S'),
      ('4', 'agg J 50 claims sev 2 * invgauss 1.5 mixed gamma 0.3162277660168379',
       10, 'Figures 10, 11, 12; Table 4 is K-S'),
      ('5', 'agg J 3 claims sev 100 * lomax 20 poisson', 15,
       'Figures 13, 14, 15; Table 5 is K-S'),
      ('6', 'agg J 6 claims sev 100 * lomax 20 mixed gamma 0.5773502691896257',
       16, 'Figures 16, 17, 18; Table 6 is K-S')]
rows = []
for name, prog, t, published in EX:
    o = build(prog, log2=20, bs=1 / 32, normalize=False)
    rows.append({'example': name, 'DecL': prog.replace('agg J ', ''),
                 'order t': t, 'agg mean': o.est_m, 'agg CV': o.est_cv,
                 'published values': published})
GT(pd.DataFrame(rows).set_index('example'),
   formatters={'agg mean': 4, 'agg CV': 4, 'order t': 0})
```

Table 30.2.: All six univariate examples as DecL programs, with the moments each builds and what the paper publishes for it.

example	DecL	order t	agg mean	agg CV	published values
1	3 claims sev 2 * gamma 3 poisson	15	18.0000	0.6667	Figures 1, 2, 3; Table 1 is K-S on simulated samples
2	15 claims sev 2 * invgauss 1.5 poisson	13	45.0000	0.4082	Figures 4, 5, 6; Table 2 is K-S
3	50 claims sev 2 * gamma 3 mixed gamma 0.3162277660168379	10	300.0000	0.3559	Figures 7, 8, 9; Table 3 is K-S
4	50 claims sev 2 * invgauss 1.5 mixed gamma 0.3162277660168379	10	150.0000	0.3873	Figures 10, 11, 12; Table 4 is K-S
5	3 claims sev 100 * lomax 20 poisson	15	15.7894	0.8389	Figures 13, 14, 15; Table 5 is K-S
6	6 claims sev 100 * lomax 20 mixed gamma 0.5773502691896257	16	31.5789	0.8278	Figures 16, 17, 18; Table 6 is K-S

Every model builds in one line, which is why the paper looked like a good target. The last column of Table 30.2 is the problem. Tables 1 to 6 are the same six rows of Kolmogorov-Smirnov statistics at $n = 100, 500, 1000, 10000$ with their critical values, and the statistic is computed between a **simulated** empirical distribution and the approximation fitted to that sample's own moments. Nothing in it is a property of the model, so nothing in it is reproducible: a different seed gives different numbers, and the paper does not report one.

The translations above are worth keeping regardless. $\mathrm{IG}(\eta, \theta)$ in the paper's parameterization is `{theta} * invgauss {eta/theta}`; $\mathrm{Pareto}(\gamma, \delta)$ with density $\gamma\delta^{\gamma}/(u+\delta)^{\gamma+1}$ is `{delta} * lomax {gamma}`; and $\mathrm{NB}(r, \beta)$ is a gamma-mixed Poisson with mean $r\beta$ and mixing CV $1/\sqrt{r}$, since matching $n + n^2c^2 = r\beta(1+\beta)$ at $n = r\beta$ gives $c^2 = \beta/n = 1/r$. The mixing CV does not depend on β, which Table 30.3 confirms.

```
r, beta = 10.0, 5.0
c = 1 / np.sqrt(r)
nb = build(f'agg NB {r * beta} claims dsev [1] mixed gamma {c}', log2=12, bs=1,
           normalize=False)
```

```
GT(pd.DataFrame([{'mixing CV': c, 'mean': nb.est_m, 'want r*beta': r * beta,
                  'variance': nb.est_sd ** 2, 'want r*beta*(1+beta)':
                      r * beta * (1 + beta)}]), float_format=6)
```

Table 30.3.: A negative binomial written as a gamma mixed Poisson, its mean and variance against $r\beta$ and $r\beta(1+\beta)$.

mixing CV	mean	want r*beta	variance	want r*beta*(1+beta)
0.316228	50.000000	50	300.000000	300

30.5. The bivariate tables are real, and are not `aggregate`'s dependence

The eight bivariate examples use Hesselager's common shock counts: $N = N_0 + N_1$ and $W = N_0 + N_2$ with N_0, N_1, N_2 independent, and all severities U_i, V_j mutually independent. Dependence between X and Y comes entirely from the shared count N_0.

`aggregate`'s `bivariate` construct has a different shape. It puts a **shared outer frequency** over both perils and couples the two per-claim severities with a copula. To ask it for Hesselager's model, thin the total count: with $\lambda = \lambda_0 + \lambda_1 + \lambda_2$ events, each event independently triggers peril X with probability $p_X = (\lambda_0 + \lambda_1)/\lambda$ and peril Y with $p_Y = (\lambda_0 + \lambda_2)/\lambda$, and the two triggers must co-occur with probability λ_0/λ. Table 30.4 is where that breaks down.

```
L0, L1, L2 = 7.0, 8.0, 9.0
L = L0 + L1 + L2
pX, pY = (L0 + L1) / L, (L0 + L2) / L
GT(pd.DataFrame([
    {'coupling of the two triggers': 'required by Hesselager',
     'P(both perils trigger)': L0 / L},
    {'coupling of the two triggers': 'independence copula',
     'P(both perils trigger)': pX * pY},
    {'coupling of the two triggers': 'comonotone, Frechet upper bound',
     'P(both perils trigger)': min(pX, pY)},
    {'coupling of the two triggers': 'countermonotone, Frechet lower bound',
     'P(both perils trigger)': max(0.0, pX + pY - 1)},
]).set_index('coupling of the two triggers'), float_format=6)
```

Table 30.4.: The co-occurrence probability Hesselager's model requires, against what any copula on the two triggers can deliver.

coupling of the two triggers	P(both perils trigger)
required by Hesselager	0.291667
independence copula	0.416667
comonotone, Frechet upper bound	0.625000
countermonotone, Frechet lower bound	0.291667

In Table 30.4 the required probability is exactly the Fréchet **lower** bound, since $p_X + p_Y - 1 = \lambda_0/\lambda$ identically. So there is a unique copula that gets the triggers right, and it is the countermonotone one. But the same copula then forces the two severities to be countermonotone whenever both

perils fire, and Hesselager needs them independent. One copula cannot be countermonotone on the triggers and independent on the severities, so the model is outside the construct. This is a structural mismatch, not a parameterization difficulty.

30.6. The joint law does reproduce, with a direct transform

The model is a two dimensional transform with a joint probability generating function. Since $\mathsf{P}_{N,W}(z_1, z_2) = \mathsf{P}_{N_0}(z_1 z_2)\,\mathsf{P}_{N_1}(z_1)\,\mathsf{P}_{N_2}(z_2)$, the joint characteristic function of (X, Y) is that pgf evaluated at the two severity transforms. `aggregate` supplies the severity discretization; the rest is a dozen lines. The target is Figure 30.1 and the result Table 30.5.

Table 7. Some values of $F_{X,Y}(x, y)$ calculated from the FFT and the proposed method.

DF	FFT	Approximation	DF	FFT	Approximation
$F_{X,Y}(52, 54)$	0.106406	0.104420	$F_{X,Y}(89, 85)$	0.708083	0.705525
$F_{X,Y}(62, 59)$	0.201156	0.201456	$F_{X,Y}(92, 93)$	0.807833	0.802099
$F_{X,Y}(67, 65)$	0.304023	0.307647	$F_{X,Y}(101, 103)$	0.905475	0.902043
$F_{X,Y}(74, 69)$	0.401565	0.404960	$F_{X,Y}(112, 110)$	0.950726	0.950726
$F_{X,Y}(75, 76)$	0.512051	0.515397	$F_{X,Y}(130, 127)$	0.990449	0.992031
$F_{X,Y}(81, 80)$	0.604866	0.604980	$F_{X,Y}(150, 150)$	0.999286	0.999013

Figure 30.1.: Jin, Provost and Ren (2016), Table 7.

```
def joint(pgf, sev_u, sev_v, h, n):
    """Joint CDF of (X, Y) = (sum_{N0+N1} U, sum_{N0+N2} V) on an h-lattice."""
    du = build(f'agg U dfreq [1] {sev_u}', log2=int(np.log2(n)), bs=h,
               normalize=False).agg_density
    dv = build(f'agg V dfreq [1] {sev_v}', log2=int(np.log2(n)), bs=h,
               normalize=False).agg_density
    zu, zv = np.fft.fft(du), np.fft.fft(dv)
    g = np.fft.ifft2(pgf(zu[:, None], zv[None, :])).real
    return np.cumsum(np.cumsum(g, axis=0), axis=1)

H, N = 0.0625, 4096
F1 = joint(lambda z1, z2: np.exp(L0 * (z1 * z2 - 1) + L1 * (z1 - 1)
                                 + L2 * (z2 - 1)),
           'sev 2 * gamma 2', 'sev 1.5 * gamma 3', H, N)
T7 = [(52, 54, .106406), (62, 59, .201156), (67, 65, .304023),
      (74, 69, .401565), (75, 76, .512051), (81, 80, .604866),
      (89, 85, .708083), (92, 93, .807833), (101, 103, .905475),
      (112, 110, .950726), (130, 127, .990449), (150, 150, .999286)]
t7 = pd.DataFrame([{'x': x, 'y': y,
                    'direct 2-D transform': F1[int(round(x / H)), int(round(y / H))],
                    'Table 7 FFT': f} for x, y, f in T7])
t7['diff'] = (t7['direct 2-D transform'] - t7['Table 7 FFT']).abs()
GT(t7.set_index(['x', 'y']),
   formatters={'direct 2-D transform': 6, 'Table 7 FFT': 6,
               'diff': '{x:.2e}'})
```

Table 30.5.: Table 7, the joint distribution function at twelve points under Poisson common shock counts.

x	y	direct 2-D transform	Table 7 FFT	diff
52	54	0.106478	0.106406	7.22e-05
62	59	0.201258	0.201156	1.02e-04
67	65	0.304145	0.304023	1.22e-04
74	69	0.401693	0.401565	1.28e-04
75	76	0.512181	0.512051	1.30e-04
81	80	0.604987	0.604866	1.21e-04
89	85	0.708186	0.708083	1.03e-04
92	93	0.807912	0.807833	7.91e-05
101	103	0.905521	0.905475	4.57e-05
112	110	0.950754	0.950726	2.79e-05
130	127	0.990455	0.990449	6.24e-06
150	150	0.999286	0.999286	4.75e-07

Bivariate Example 3 swaps Poisson for negative binomial counts, NB(6, 5), NB(7, 5), NB(8, 5), and needs only a different pgf. The target is Figure 30.2 and the result Table 30.6.

Table 11. Some values of $F_{X,Y}(x, y)$ calculated from the FFT and the proposed method.

DF	FFT	Approximation	DF	FFT	Approximation
$F_{X,Y}(223, 225)$	0.100787	0.101027	$F_{X,Y}(369, 373)$	0.703603	0.701101
$F_{X,Y}(253, 257)$	0.202233	0.203455	$F_{X,Y}(405, 403)$	0.802411	0.800898
$F_{X,Y}(280, 280)$	0.303393	0.304460	$F_{X,Y}(450, 452)$	0.903216	0.903584
$F_{X,Y}(304, 300)$	0.400091	0.400524	$F_{X,Y}(494, 491)$	0.950212	0.951278
$F_{X,Y}(321, 325)$	0.504903	0.504091	$F_{X,Y}(580, 580)$	0.990640	0.990754
$F_{X,Y}(348, 345)$	0.601492	0.600005	$F_{X,Y}(698, 698)$	0.999271	0.999075

Figure 30.2.: Jin, Provost and Ren (2016), Table 11.

```
R0, R1, R2, BETA = 6.0, 7.0, 8.0, 5.0
def nb_pgf(z1, z2):
    P = lambda z, r: (1 - BETA * (z - 1)) ** -r
    return P(z1 * z2, R0) * P(z1, R1) * P(z2, R2)

H3 = 0.25
F3 = joint(nb_pgf, 'sev 2 * gamma 2', 'sev 1.5 * gamma 3', H3, 4096)
T11 = [(223, 225, .100787), (253, 257, .202233), (280, 280, .303393),
       (304, 300, .400091), (321, 325, .504903), (348, 345, .601492),
       (369, 373, .703603), (405, 403, .802411), (450, 452, .903216),
       (494, 491, .950212), (580, 580, .990640), (698, 698, .999271)]
t11 = pd.DataFrame([{'x': x, 'y': y,
                     'direct 2-D transform':
                         F3[int(round(x / H3)), int(round(y / H3))],
                     'Table 11 FFT': f} for x, y, f in T11])
t11['diff'] = (t11['direct 2-D transform'] - t11['Table 11 FFT']).abs()
GT(t11.set_index(['x', 'y']),
   formatters={'direct 2-D transform': 6, 'Table 11 FFT': 6,
               'diff': '{x:.2e}'})
```

Table 30.6.: Table 11, the joint distribution function at twelve points under negative binomial common shock counts.

x	y	direct 2-D transform	Table 11 FFT	diff
223	225	0.100985	0.100787	1.98e-04
253	257	0.202520	0.202233	2.87e-04
280	280	0.303725	0.303393	3.32e-04
304	300	0.400436	0.400091	3.45e-04
321	325	0.505241	0.504903	3.38e-04
348	345	0.601802	0.601492	3.10e-04
369	373	0.703865	0.703603	2.62e-04
405	403	0.802608	0.802411	1.97e-04
450	452	0.903326	0.903216	1.10e-04
494	491	0.950274	0.950212	6.18e-05
580	580	0.990653	0.990640	1.31e-05
698	698	0.999273	0.999271	1.71e-06

Table 30.5 and Table 30.6 both land within 3.5×10^{-4}, with a residual that is uniformly positive, largest in the body of the distribution and vanishing in the tail. That is the signature of a half bucket offset rather than a disagreement about the model, and Table 30.7 confirms it by halving the bucket.

```
rows = []
for h, log2 in [(0.5, 9), (0.25, 10), (0.125, 11), (0.0625, 12)]:
    F = joint(lambda z1, z2: np.exp(L0 * (z1 * z2 - 1) + L1 * (z1 - 1)
                                    + L2 * (z2 - 1)),
              'sev 2 * gamma 2', 'sev 1.5 * gamma 3', h, 2 ** log2)
    rows.append({'bucket': h, 'grid': 2 ** log2, 'reach': h * 2 ** log2,
                 **{f'({x},{y})': F[int(round(x / h)), int(round(y / h))] - f
                    for x, y, f in [T7[0], T7[4], T7[8], T7[11]]}})
GT(pd.DataFrame(rows).set_index('bucket'),
   formatters={f'({x},{y})': '{x:.2e}'
               for x, y, f in [T7[0], T7[4], T7[8], T7[11]]})
```

Table 30.7.: Deviation from four cells of Table 7 as the bucket is halved four times.

bucket	grid	reach	(52,54)	(75,76)	(101,103)	(150,150)
0.50000	512	256	2.64e-03	4.60e-03	1.57e-03	1.69e-05
0.25000	1,024	256	1.16e-03	2.06e-03	7.20e-04	8.08e-06
0.12500	2,048	256	4.34e-04	7.77e-04	2.74e-04	3.11e-06
0.06250	4,096	256	7.22e-05	1.30e-04	4.57e-05	4.75e-07

In Table 30.7 the deviation falls monotonically at every halving, by a factor of thirty-five overall from `bs = 0.5` to `bs = 0.0625`, so the paper’s FFT column is the exact joint distribution function and the reproduction agrees with it.

30.7. Why the marginals are not enough

The candidate assessment suggested falling back on marginals. That does not rescue the reproduction, because the marginals miss precisely the quantity Figure 30.1 and Figure 30.2 are reporting, as Table 30.8 shows.

```
mx = build('agg X 15 claims sev 2 * gamma 2 poisson', log2=18, bs=1 / 32,
           normalize=False)
my = build('agg Y 16 claims sev 1.5 * gamma 3 poisson', log2=18, bs=1 / 32,
           normalize=False)
gap = pd.DataFrame([
    {'x': x, 'y': y, 'joint F(x,y)': F1[int(round(x / H)), int(round(y / H))],
     'FX(x) FY(y) under independence': (mx.density_df.loc[x, 'F']
                                        * my.density_df.loc[y, 'F']),
     'Table 7': f} for x, y, f in T7])
gap['dependence gap'] = gap['joint F(x,y)'] - gap['FX(x) FY(y) under independence']
GT(gap.set_index(['x', 'y']), float_format=6)
```

Table 30.8.: The joint distribution function against the product of its own marginals, which is what dropping the common shock would give.

x	*y*	joint F(x,y)	FX(x) FY(y) under independence	Table 7	dependence gap
52	54	0.106478	0.071004	0.106406	0.035474
62	59	0.201258	0.158877	0.201156	0.042381
67	65	0.304145	0.259349	0.304023	0.044796
74	69	0.401693	0.363693	0.401565	0.038000
75	76	0.512181	0.474976	0.512051	0.037204
81	80	0.604987	0.577291	0.604866	0.027695
89	85	0.708186	0.691658	0.708083	0.016528
92	93	0.807912	0.796772	0.807833	0.011140
101	103	0.905521	0.901455	0.905475	0.004066
112	110	0.950754	0.949539	0.950726	0.001215
130	127	0.990455	0.990366	0.990449	0.000089
150	150	0.999286	0.999284	0.999286	0.000003

The margins are right: $N = N_0 + N_1 \sim \text{Poisson}(15)$ and $W = N_0 + N_2 \sim \text{Poisson}(16)$, both one-line builds. But the common shock moves the joint distribution function up to 4.5 points of probability away from the independent product, so a marginals-only reproduction would be reporting a number the paper never computed.

30.8. What would make this a reproduction

One feature: a common shock count mode for `bivariate`, in which the two components share a frequency component N_0 and carry independent private components N_1, N_2, with severities independent throughout. That is the Hesselager model, it is standard in bivariate aggregate work, and the machinery to compute it is the twelve line `joint` function above sitting on the existing 2-D grid and severity discretization. It would turn 96 published reference values across eight tables into a regression suite, and it would give `aggregate` a count-driven dependence to sit beside the copula-driven one it already has.

Two smaller notes for whoever picks it up.

The paper has a specification defect in bivariate Example 6. The text reads "Let $N_0 \sim \mathrm{NB}(5)$, $N_1 \sim \mathrm{NB}(6)$, $N_2 \sim \mathrm{NB}(7)$", giving one parameter to a two parameter family. The surrounding examples all use $\beta = 5$, so $\mathrm{NB}(5, 5)$, $\mathrm{NB}(6, 5)$, $\mathrm{NB}(7, 5)$ is the obvious reading, but it is a guess and Tables 18 to 20 would decide it.

Tables 8, 10, 12, 14, 16, 17, 19, 20, 22 and 24 are simulation output and share the univariate half's problem: they compare an empirical distribution from 5,000 simulated draws against the approximation fitted to that sample's moments. Only the odd-numbered tables carry the FFT column that is worth anything as a reference.

Part VIII.

Severity curves

31. Provenance of the named severity curves

What each published curve in the shipped library is, and where it came from

31.1. Introduction

`src/aggregate/agg/library.agg` ships eleven reusable severity objects named `<Citation-Key>.<Curve>`, harvested from the candidate assessment in `plan-examples.md` while the reproductions in this folder were being written. Each carries its citation key in `note{}`, and little else. This page is the rest of the record: which paper, which table or exhibit the parameters came out of, what the paper uses the curve for, and every transcription decision that stands between the printed parameters and the DecL.

A twelfth library severity carries the `role:published` tag, `CommAutoMixedExponentialSev`, a five component commercial auto mixed exponential at the 2008 cost level. It came out of my own *Actuarial Review* article rather than out of the assessment, so it has no provenance question to answer and is not covered here.

The curves are working definitions, not reproductions. Where a reproduction exists it is cited, and it is the authority; where none exists the entry says so. Table 31.1 is what actually ships.

```
%xmode minimal
import warnings

import numpy as np
import pandas as pd
from bookstyle import GT, build

rows = []
with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    # `role:published` is the library's tag for a curve whose parameters were
    # printed somewhere. Twelve entries carry it; the dot in the name is what
    # separates the eleven `<CitationKey>.<Curve>` harvests this page documents
    # from `CommAutoMixedExponentialSev`, which came from elsewhere.
    for name in build.discover(kind='sev', tags='role:published').index:
        if '.' not in name:
            continue
        fz = build(f'agg T 1 claim sev.{name} fixed', update=False).sevs[0].fz
        # `moments=` by keyword: a tabulated curve's `fz` is an unfrozen
        # rv_histogram, which reads a positional 'mv' as a shape parameter.
        m, v = fz.stats(moments='mv')
        dist = getattr(fz, 'dist', None)
        rows.append({'curve': name,
                     'scipy': getattr(dist, 'name', None) or type(fz).__name__,
                     'shape': ', '.join(f'{a:g}' for a in getattr(fz, 'args', ())),
                     'scale': float(getattr(fz, 'kwds', {}).get('scale', np.nan)),
```

```
                    'mean': float(m),
                    'cv': float(np.sqrt(v) / m) if np.isfinite(v) else np.nan})

GT(pd.DataFrame(rows).set_index('curve'), float_format=',.4g')
```

```
Exception reporting mode: Minimal
```

Table 31.1.: Every per-paper severity curve in the shipped library, its `scipy` family, parameters and moments.

curve	scipy	shape	scale	mean	cv
Berens1997.Lognorm	lognorm	0.246221	1.455e+05	1.5e+05	0.25
Berens1997.Pareto	pareto	1.5	5e+04	1.5e+05	—
Bodoff2017.Pareto	lomax	1.5	5e+04	1e+05	—
Jin2016.Gamma	gamma	2.99948	2	6	0.5774
Jin2016.Pareto	lomax	20	100	5.263	1.054
Kang2019.Pareto	lomax	3.5	3.5	1.4	1.528
Mack2003.AmericanPareto	lomax	1.5	1e+04	2e+04	—
Mack2003.Lognorm	lognorm	1.79998	1,979	1e+04	4.953
Mack2003.RiebesellPareto	lomax	0.737	1e+04	—	—
Mata2005.Lognorm	lognorm	2.29	1.105e+04	1.521e+05	13.73
Venter1983.Piecewise	rv_histogram		—	1.88e+04	2.734

A blank coefficient of variation means the second moment does not exist, and an infinite mean means the first does not either. A blank shape and scale means the curve is a tabulation rather than a parametric family, which is why `Venter1983.Piecewise` has neither, and why its mean and CV here are the *unlimited* ones and so do not match the paper: that curve is only itself inside the layer described below. Table 31.1 is built from the library, so it cannot drift from what actually ships.

31.2. Summary: paper to curve

Table 31.2 maps each assessed paper to the curves it contributed.

Table 31.2.: Each assessed paper, the curves it contributed to the library, and why.

Paper	Curves	Why
Berens (1997)	`Berens1997.Lognorm`, `Berens1997.Pareto`	the severities of the paper's two simulation models
Mack and Fackler (2003)	`Mack2003.Lognorm`, `Mack2003.AmericanPareto`, `Mack2003.RiebesellPareto`	two columns of Table 1, plus the Pareto tail index Riebesell's rule implies
Mata and Verheyen (2005)	`Mata2005.Lognorm`	the one severity behind every table in the paper
Bodoff (2017)	`Bodoff2017.Pareto`	Exhibit 2
Jin et al. (2016)	`Jin2016.Gamma`, `Jin2016.Pareto`	the univariate test severities
Kang et al. (2019)	`Kang2019.Pareto`	the running heavy tailed illustration

Paper	Curves	Why
Venter (1983)	`Venter1983.Piecewise`	the claim size law behind Exhibit 3, a tabulated CDF rather than a named family
Bruno et al. (2006)	none	a 14 point discrete severity, likewise not a named family
Clark (2005)	none	three lognormal lines specified by premium and CV, nothing worth naming
Lau (1984)	none	moments only. The severity form is never stated and has to be reverse fitted
Braithwaite and Ware (1997)	none	never assessed. It surfaced only as the conceptual pair to Bodoff (2017)
Goffard and Laub (2020), Nadarajah et al. (2016), Vernic (1999), Bakar et al. (2022), Provost (2005), Hilbe (2014), Lindsay et al. (2000)	none	Tier 3 in the assessment: no reproducible parameters, no compounding, or no insurance content

Bruno et al. (2006) has no curve and is a full reproduction anyway. A severity does not have to be a named family to be worth carrying: `Venter1983.Piecewise` is a tabulated CDF, and Bruno's 14 point law would sit in the library just as well as a `dsev` if anything besides its own chapter needed it.

The rare event and numerical accuracy set (`Asmussen2016a`, `BenAmar2023`, `BenRached2024`, `Liu2026`, `Wilson2016a`, `Wilson2017`, `Homer2003`) contributed no named curves. Those papers are about estimator accuracy in the far tail, and their test severities are unit scale lognormals, Paretos and Lévy laws chosen to stress the estimator rather than published as models of anything.

31.3. The curves

31.3.1. Berens1997

Paper. Berens (1997), *Reinsurance contracts with a multi-year aggregate limit*, CAS Forum.

What the paper does with them. It is a simulation study of the surcharge a reinsurer should charge for letting an aggregate limit run over three years instead of one, priced across excess layers of width \$150,000 attaching at \$150,000, \$300,000 and \$450,000. Two frequency and severity pairs are run, deliberately chosen to disagree. The first is Poisson with 30 expected claims and a lognormal severity of mean \$150,000 and standard deviation \$37,500 (Exhibit I, upper). The second is negative binomial with 30 expected claims and $p = 0.10$, and a single parameter Pareto severity with $q = 1.5$ and mean \$150,000 (Exhibit III). Exhibit V plots claim size percentiles from both, and it is the paper's explanation of why the surcharge ranking flips as the attachment rises: under the Pareto far more claims fall below a \$150,000 retention, but at high attachments so many are large enough to exhaust the layer that the layer loss is actually *less* volatile.

`Berens1997.Lognorm` = `lognorm 150000 cv 0.25`. The mean and standard deviation are stated, and \$37,500 / \$150,000 = 0.25 is the CV. The paper then writes "the distribution of the severity was

thus $\Lambda(11.92, .25)$", which is the shortcut $\mu = \ln(\text{mean})$, $\sigma = \text{cv}$, and is not consistent with its own moments: $\Lambda(11.92, 0.25)$ has mean 155,011 and CV 0.2540. The library object follows the stated moments, giving $\sigma = 0.246221$ and $\mu = 11.8881$. Nothing in the paper turns on the difference, but a reproduction that quoted the printed (μ, σ) would run 3.3% high on the severity mean, and further out on excess layers, where a shift in the body of a tight lognormal is amplified.

`Berens1997.Pareto = 50000 * pareto 1.5`. "Single parameter Pareto" is the European or Type I form, supported on $[\theta, \infty)$, DecL `pareto`. Its mean is $q\theta/(q-1) = 3\theta$, so the stated mean of \$150,000 pins $\theta = 50{,}000$. The paper's second Pareto run has mean \$300,000, hence $\theta = 100{,}000$; that variant is not carried. With $q = 1.5$ the variance is infinite, and because the support starts at 50,000 rather than 0 `aggregate` logs `Pareto not shifted to x>0 range` and falls back to numeric moments. That message is expected here, not a defect.

Status. No reproduction. The single year layer pieces are native, but the rolling multi-year aggregate limit is path dependent and needs a wrapper loop over independent annual builds with cumulative capping.

31.3.2. Mack2003

Paper. Mack and Fackler (2003), *Exposure-rating in liability reinsurance*, Blätter der DGVFM. Reproduced in `Mack2003.qmd`.

What the paper does with them. It asks whether Riebesell's 1936 increased limits rule, that doubling the sum insured multiplies the net premium by $1 + z$, is consistent with the collective risk model. A power law premium forces a Pareto claim size law with $\alpha = 1 - \text{ld}(1 + z)$, which is below one and so has an infinite mean. Theorem 2 rescues the rule: for any continuous F with a finite mean there is a threshold u such that grafting a Pareto tail onto F above u makes Riebesell exact on $[u, \infty)$. Table 1 tabulates the function that locates that threshold,

$$\alpha_F(u) = 1 - \frac{u\, S(u)}{\mathsf{E}[X \wedge u]},$$

for four severities at seven multiples of the mean. Three of the four columns are here.

`Mack2003.Lognorm = lognorm 10000 cv 4.953`. Table 1's "Lognormal $\sigma = 1.8$" column, and also the claim size distribution of the paper's own worked example in §7. The CV is $\sqrt{e^{\sigma^2} - 1} = 4.953153$ at $\sigma = 1.8$; the library rounds to four figures, which recovers $\sigma = 1.79998$. Table 1 is indexed by $u/\mathsf{E}[X]$ and so is scale free. The 10,000 mean is a placeholder, and the paper remarks only that $\mathsf{E}[X]$ in liability is usually below €50,000.

`Mack2003.AmericanPareto = 10000 * lomax 1.5`. Table 1's "American Pareto $\alpha = 1.5$" column. The paper does not define the term, and the reading that it means Type II, that is Lomax with $S(x) = (\theta/(\theta + x))^\alpha$ and support from zero, is an inference. `Mack2003.qmd` confirms it: all 26 cells of Table 1 come back within 8e-4 under that reading.

`Mack2003.RiebesellPareto = 10000 * lomax 0.737`. Not a curve the paper tabulates. It is the tail index Riebesell implies at the customary $z = 20\%$: $\alpha = 1 - \ln(1.2)/\ln 2 = 0.736966$, which the paper prints as 0.737. **This object carries the exponent, not Mack and Fackler's construction.** Their tail is spliced onto the lognormal above $u \approx 5\,\mathsf{E}[X]$ and is conditional on exceeding u; this is a ground up Lomax that merely has the right index. It has an infinite mean, so use it only inside a finite layer. Unbounded it does not raise, it silently builds on a degenerate two bucket grid and reports a mean of 0.67. For the actual splice, and the refinement that the threshold is $u^* = 5.186\,\mathsf{E}[X]$ where the spliced curve gives $b(2v)/b(v) = 1.200000$ exactly, see `Mack2003.qmd`.

31.3.3. Mata2005

Paper. Mata and Verheyen (2005), *An Improved Method for Experience Rating Excess of Loss Treaties using Exposure Rating Techniques*, CAS Forum. Reproduced in `Mata2005.qmd`.

What the paper does with it. One severity drives the whole worked example: layer by layer experience and exposure rating of five excess layers over a limits profile running from \$250,000 to \$5,000,000, with an 8% ground up severity trend, a 60% expected gross loss ratio, and the trend then decomposed into frequency and severity components layer by layer. Tables 1 to 5 and 9 to 14 are all limited expected values or ratios of them off this curve.

`Mata2005.Lognorm = 11048 * lognorm 2.29`. The 2005 level lognormal, $\mu = 9.31$ and $\sigma = 2.29$, written in scipy's scale form with scale $= e^{9.31} = 11{,}047.95$ rounded to 11,048. Mean 152,064, CV 13.73. The 2000 level curve, $\mu' = \mu - \ln(1.08^5) = 8.925195$, is not carried; scale it by 1.08^{-5}. The paper prints μ' rounded to 8.93, and using the rounded value moves the limited expected values by about half a percent, more than the effects the paper is measuring, so derive it rather than transcribe it.

Germane finding. `Mata2005.qmd` reproduces Tables 2, 3, 4, 5, 11, 13 and 14 to the last printed digit when driven off the paper's own Table 1, but Table 1's limited expected values are themselves not the exact lognormal ones. They run 0.04% high at \$250,000 drifting to 0.24% low at \$5M, in both years, while `aggregate` matches the closed form to eight figures. The paper's conclusions are unaffected, but a mismatch against Table 1 is the paper's, not the library's.

31.3.4. Bodoff2017

Paper. Bodoff (2017), *An Actuarial Model of Excess of Policy Limits Losses*, Variance. Reproduced in `Bodoff2017.qmd`. Not the Bodoff already in the docs, which is Bodoff (2007) on percentile layer capital allocation.

What the paper does with it. Exposure rating for excess of policy limits losses, the ones where the insurer pays past the limit. The paper's move is to treat the *effectiveness* of the policy limit as Bernoulli: with probability p the limit caps the loss and with probability $1 - p$ it does not, so the policy limited expected value is a mixture $p\,\mathsf{E}[X \wedge k] + (1-p)\,\mathsf{E}[X]$ of two ordinary limited expected values off the same curve. Exhibit 2 is the severity, Exhibit 1 a nine point limits profile, Exhibit 3 sets $p = 99\%$, and Exhibit 4 is the six layer exposure rating that results.

`Bodoff2017.Pareto = 50000 * lomax 1.5`. Exhibit 2 says only "Pareto, $\theta = 50{,}000$, $\alpha = 1.50$". That it means the two parameter Pareto, $S(x) = (\theta/(\theta + x))^{\alpha}$, is an inference: it is the form in the Clark reinsurance study note the paper cites, and `Bodoff2017.qmd` confirms it by fit. Mean 100,000, infinite variance, so it needs a limit. The XPL model conditions this severity *before* the policy limit and then mixes over the limits profile.

31.3.5. Jin2016

Paper. Jin et al. (2016), *Moment-based density approximations for aggregate losses*, Scandinavian Actuarial Journal. Assessed in `Jin2016.qmd`, which is **not** a reproduction.

What the paper does with them. They are test severities for a moment based approximation, not models of anything. The method approximates the density of $S = X \mid X > 0$ by a gamma base density times a polynomial adjustment, and needs only moments, so it works from sample moments when frequency and severity are not separately known. The paper's own ground truth is the FFT.

`Jin2016.Gamma = gamma 6 cv 0.5774`. Example 1's $U \sim \gamma(3, 2)$, in the shape and scale reading, so mean 6 and CV $1/\sqrt{3}$. The convention is not stated in the paper but is pinned by arithmetic: Example 1 reports its fitted base gamma as (2.684349, 7.056877), and only the scale reading reproduces those, which `Jin2016.qmd` does to six figures. Transcription wart: 0.5774 is a four figure rounding of $1/\sqrt{3} = 0.5773502691896258$ and yields shape 2.99948 rather than 3. Write `sev 2 * gamma 3` when exactness matters.

`Jin2016.Pareto = 100 * lomax 20`. Examples 5 and 6 use $U \sim \text{Pareto}(\gamma, \delta) = \text{Pareto}(20, 100)$ with density $\gamma\delta^\gamma/(u+\delta)^{\gamma+1}$ on $u > 0$, which is Lomax with shape γ and scale δ. Mean $100/19 = 5.2632$. The paper's figures also show a Pareto(5, 100) variant, which is *heavier* rather than lighter; it is not carried.

Why there is no reproduction. The univariate half of the paper publishes figures and Kolmogorov-Smirnov statistics and nothing else, and those statistics are computed between a simulated empirical distribution and an approximation fitted to that sample's own moments, with no seed given. They are properties of a draw, not of the model. Every genuine reference value in the paper, 96 of them, is in the FFT columns of the bivariate tables, whose dependence is Hesselager common shock *counts* that `aggregate`'s copula coupled `bivariate` cannot express. So these two curves are a parameter transcription that happens to be checkable, not a reproduction target.

31.3.6. Kang2019

Paper. Kang et al. (2019), *Moment-Based Density Approximation Techniques as Applied to Heavy-tailed Distributions*, International Journal of Statistics and Probability.

What the paper does with it. Pareto II(3.5, 3.5) is the running illustration for approximating a heavy tailed density by exponential tilting: tilt the density so it becomes exponentially bounded, approximate the tilted density by a base density times a polynomial adjustment, then untilt. Figure 1 plots the tilted density at $\theta = 1.5$, Table 1 gives integrated squared errors over a grid of tilting parameters and polynomial degrees, and Tables 2 and 3 give VaR and TVaR for the exact distribution beside three approximants. The exact column is a clean external check on a heavy tail: $\text{VaR}_{0.99} = 9.54658$, $\text{TVaR}_{0.99} = 14.7652$, $\text{TVaR}_{0.999} = 31.7646$.

`Kang2019.Pareto = 3.5 * lomax 3.5`. The paper defines Pareto II(α, β) explicitly, density $\alpha\beta^\alpha/(x+\beta)^{\alpha+1}$ on $x \geq 0$, and names it Lomax, so the mapping is unambiguous: shape $\alpha = 3.5$, scale $\beta = 3.5$. Mean 1.4, CV 1.5275, skewness 11.78. Moments exist up to order 3 only, so there is no kurtosis. Watch the parameters: §6.3.1 of the paper switches to a simulated sample from Pareto II(3.5, **35**), rescaled, and the VaR and TVaR figures there are not comparable. The library object is the analytic Pareto II(3.5, 3.5) of §6.1.

Status. No reproduction. The paper's "aggregate" is severity only, with no compounding, so it exercises the quantile machinery and nothing else. It is a good heavy tail VaR and TVaR accuracy benchmark, which is why the curve was kept.

31.3.7. Venter1983

Paper. Venter (1983), *Transformed beta and gamma distributions and aggregate losses*, PCAS. Reproduced in `Venter1983.qmd` and in the highlights chapter.

What the paper does with it. It is the claim size law of Exhibit 3, the paper's test of the transformed gamma as a three moment approximation to the aggregate. Exhibit 3 Part 2 states it as a piecewise linear CDF from \$0 to a \$250,000 policy limit, tabulated at 23 points, with the residual 2.41% of

probability sitting as a point mass at the limit. Frequency is Poisson with $\lambda = 13.7376$, and the compound distribution has mean \$250,000, CV 0.7667 and skewness 1.0744.

`Venter1983.Piecewise` = `chistogram xps [...] [...]`. The 23 knots are Venter's own, and the 23 probabilities are the successive CDF increments with the limit mass appended. **The top bin is not part of the paper.** The atom at the limit is easier to obtain than to declare, so the limit mass is carried in one wide bin from \$250,000 to \$300,000 and a `250000 xs 0` layer then folds it back onto the limit. That makes the curve correct only inside that layer; used unlimited it spreads 2.41% of probability uniformly over a \$50,000 interval that Venter never wrote. Like `Mack2003.RiebesellPareto`, this one carries a usage constraint rather than a free standing distribution.

The transcription has an acceptance test built in, which is why the curve can be trusted: read the knots wrongly and the severity mean, CV and skewness (18,198.195, 2.6600, 3.6746) stop matching Exhibit 3 Part 2. They match, as the moments table in the Venter1983 chapter records.

What is not carried. Exhibit 3 Part 2 also prints Venter's own \$500 discrete PMF, whose first five rows could not be decoded. Nothing depends on it, since `aggregate` discretizes the piecewise linear CDF itself.

31.4. Reference curves with no paper attached

Alongside the eleven published curves the library carries 18 single claim aggregates named `Curve*`, listed by `build.discover(kind='agg', tags='topic:severity role:reference')`. These are vocabulary, not reproductions. Each is the simplest possible carrier, one fixed claim so the aggregate *is* the severity, and the shapes are illustrative with a common scale near 10,000 so they can be read against each other. No paper prescribes any of these parameters.

What they *do* record is a selection: scipy ships more than a hundred continuous distributions and actuaries use a small, well trodden subset. The picks and the literature behind them are:

- **Light and moderate tails**, `CurveExponential`, `CurveGamma`, `CurveWeibull`, `CurveInverseGaussian`. The standard working severities. Appendix A of Klugman et al. (2012) is the canonical actuarial table and the parameterizations follow its families; Parodi (2015) and Parodi et al. (2024) use lognormal, gamma and Weibull as the everyday attritional curves, and Parodi and Watson (2019) derives them from a property graph model of fire and explosion losses.
- **Lognormal and the GB2 family**, `CurveLognormal`, `CurveGenGamma`, `CurveBetaPrime`, `CurveBurr12`, `CurveBurrIII`, `CurveLogLogistic`. The flexible three and four parameter curves that nest most of the others. Venter (1983) introduced the transformed beta and gamma to actuaries, McDonald and Richards (1987) studied the hazard rates of the generalized beta family, and Cummins et al. (1990) is the insurance loss application. The GB2 umbrella covers the log-t, generalized gamma, B2, Burr III and XII, Lomax, Fisk, gamma, Weibull, and the lognormal as a limit.
- **Heavy and power law tails**, `CurvePareto`, `CurveLomax`, `CurveGPD`, `CurveInverseGamma`. Large loss and reinsurance work, plus the peaks over threshold tail of extreme value theory. The two power laws both use shape 2.5, chosen so the variance stays finite: the auto sizer refuses an infinite variance without an explicit grid.
- **Extreme value laws**, `CurveGumbel`, `CurveFrechet`, `CurveGEV`. Block maxima, for a largest event rather than a ground up attritional severity.
- **Bounded severity**, `CurveBeta`. A loss as a fraction of a known sum insured, and the gateway to exposure curve models.

`CurveBeta` also carries a deliberate absence. The Swiss Re standard for property per risk exposure rating is the two parameter MBBEFD class of Bernegger (1997) on the destruction ratio $[0, 1]$. It is an exposure *curve* family rather than a scipy continuous random variable, so there is no `sev mbbefd` form and the bounded beta is the closest stand in currently available.

31.5. Using the curves

`library.agg` is the default database, so the curves resolve from a bare `build` with no explicit load, as Table 31.3 shows:

```
a = build('agg XPL 1 claim 1000000 xs 0 sev.Bodoff2017.Pareto fixed')
b = build('agg J 3 claims sev.Jin2016.Gamma poisson')

GT(pd.DataFrame([{'program': 'Bodoff2017.Pareto limited to 1M', 'mean': a.est_m},
                 {'program': 'Poisson(3) x Jin2016.Gamma', 'mean': b.est_m}]
                ).set_index('program'), float_format=',.2f')
```

Table 31.3.: Two programs resolving a named curve straight out of the shipped library.

program	mean
Bodoff2017.Pareto limited to 1M	78,178.21
Poisson(3) x Jin2016.Gamma	18.00

Four of the eleven have no finite variance and need a limit or an explicit grid. `Bodoff2017.Pareto` and `Mack2003.AmericanPareto` have $\alpha = 1.5$ and raise `InfiniteVarianceError` unbounded, which is the behavior you want. The other two fail less cleanly, and both are worth knowing about:

- `Berens1997.Pareto`, also $\alpha = 1.5$, is a Type I Pareto supported from 50,000 up, so it takes the numeric moment path and never reaches the infinite variance check. It raises `ValueError: Inadmissible value passed to round_bucket, inf` instead.
- `Mack2003.RiebesellPareto` at $\alpha = 0.737$ has no mean either, and it fails silently. The auto sizer cannot size a grid from an infinite mean, falls back to a two bucket grid, and returns 0.67 with no error at all. Always give this one a layer.

Part IX.

Appendix

32. Every program in the book

The DecL declarations behind the reproductions, 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 paper they reproduce 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. A line wider than the page is folded at its spaces onto indented continuation lines, which parse exactly as the unfolded line does. One declaration, 28,000 characters of exact rational probabilities, is cut short where marked.

Three things follow from generating it this way. A chapter that builds nothing does not appear. Programs are collected by declaration rather than by object, so a chapter that rebuilds one model at ten bucket sizes contributes one entry and not ten, 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 rather than listed in full. A chapter that values the same severity at thirty limits writes thirty distinct declarations, and printing all thirty teaches a reader nothing the first two do not. Two of each shape are shown, followed by a count of the rest, where a shape is the declaration with its numbers taken out. Nothing is hidden by this that is not an arithmetic variant of something on the page beside it.

Every program below except the one cut short runs as it stands, in the aggregate Loss Lab (aLL) with no installation, or through `build` in a Python session.

The same declarations are also available as one loadable file, `reproductions.agg` in the project root, written from the same store by `make_agg.py`. Each one there carries a `note{}` naming the paper it comes from and saying in a line what it is, and each is renamed by citation key so that nothing in the file shadows anything else: `build.load('reproductions.agg')` puts all of them in the recipe base, after which `build('Bear1990.BN.TreatyVI')` rebuilds that treaty by name. Two things differ from the listing below. The shipped severity curves appear in full rather than sampled, each being a curve in its own right rather than an arithmetic variant of the one above it, and a model that more than one chapter builds appears once.

Five papers, five programs

From Venter (1983), Bear and Nemlick (1990), Homer and Clark (2003), Bruno et al. (2006) and Denuit, Dhaene, et al. (2025), reproduced in Chapter 1.

```
agg Venter
  13.7376 claims
  250000 xs 0
  sev chistogram xps [0 1000 5000 6000 7000 8000 9000 10000 12500 15000 17500 20000
    25000 35000 50000 75000 100000 125000 150000 175000 200000 225000 250000 300000]
    [0.38935 0.38935 0.00568 0.00543 0.00517 0.00495 0.00473 0.01098 0.00989 0.00896
    0.00815 0.01426 0.02237 0.02353 0.02459 0.01517 0.01021 0.00732 0.00547 0.00423
    0.00337 0.00274 0.0241]
  poisson
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  poisson
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  poisson
  aggregate net of
    360 xs 0
```

```
agg HC
  5 claims
  dsev [200000 400000 600000 800000 1000000] [0.378 0.235 0.146 0.091 0.15]
  occurrence net of
    400000 xs 600000
  mixed gamma 0.2
```

```
agg B7
  91000 claims
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.0103301 0.030799 0.0293511
    0.0103301 0.0730414 0.0111568 0.0264554 0.1002133 0.0815418 0.0252146 0.0212857
    0.0254214 0.0991756 0.4556837]
  poisson
```

```
port Denuit.Pool
  agg Agent1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent2
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent3
    1 claim
    sev 2 * gamma [1 8] wts [0.504541349633 0.495458650367] splice [0 0] [30 30]
    fixed
```

Mack and Fackler's test of Riebesell's rule

From Mack and Fackler (2003), reproduced in Chapter 3.

```
agg S
  1 claim
  0.01 xs 0
  sev expon
  fixed
```

```
agg S
  1 claim
  0.1 xs 0
  sev expon
  fixed
```

and 5 more of this shape, differing only in their numbers.

```
agg S
  1 claim
  sev expon
  fixed
```

```
agg S
  1 claim
  0.01 xs 0
  sev 10 * gamma 0.1
  fixed
```

```
agg S
  1 claim
  0.1 xs 0
  sev 10 * gamma 0.1
  fixed
```

and 5 more of this shape, differing only in their numbers.

```
agg S
  1 claim
  sev 10 * gamma 0.1
  fixed
```

```
agg S
  1 claim
  0.01 xs 0
  sev lognorm 1 cv 4.95315270785704
  fixed
```

```
agg S
  1 claim
  0.1 xs 0
  sev lognorm 1 cv 4.95315270785704
  fixed
```

and 5 more of this shape, differing only in their numbers.

```
agg S
  1 claim
  sev lognorm 1 cv 4.95315270785704
  fixed
```

```
agg S
  1 claim
  0.01 xs 0
  sev 0.5 * lomax 1.5
  fixed
```

```
agg S
  1 claim
  0.1 xs 0
  sev 0.5 * lomax 1.5
  fixed
```

and 5 more of this shape, differing only in their numbers.

```
agg S
  1 claim
  sev 0.5 * lomax 1.5
  fixed
```

```
agg Riebesell
  1 claim
  5000000 xs 0
  sev [9894.934954 259286.6924] * [lognorm pareto] [1.8 0.7369655942] wts
    [0.965191548087 0.034808451913] splice [0 259286.6924 inf]
  fixed
```

```
agg Riebesell
  1 claim
  50000 xs 0
  sev [9894.934954 259286.6924] * [lognorm pareto] [1.8 0.7369655942] wts
    [0.965191548087 0.034808451913] splice [0 259286.6924 inf]
  fixed
```

and 11 more of this shape, differing only in their numbers.

Mata and Verheyen on trend by layer

From Mata and Verheyen (2005), reproduced in Chapter 4.

```
agg M
  1 claim
  250000 xs 0
  sev lognorm 152062.97146657703 cv 13.727535749987513
  fixed
```

```
agg M
  1 claim
  500000 xs 0
  sev lognorm 152062.97146657703 cv 13.727535749987513
  fixed
```

and 8 more of this shape, differing only in their numbers.

```
agg M
  1 claim
  sev lognorm 152062.97146657703 cv 13.727535749987513
  fixed
```

```
agg M
  1 claim
  sev lognorm 103491.50327117534 cv 13.727535749987513
  fixed
```

Bodoff and excess of policy limits losses

From Bodoff (2017), reproduced in Chapter 5.

```
agg B
  1 claim
  50000 xs 0
  sev 50000 * lomax 1.5
  fixed
```

```
agg B
  1 claim
  100000 xs 0
  sev 50000 * lomax 1.5
  fixed
```

and 8 more of this shape, differing only in their numbers.

Homer and Clark's bivariate FFT

From Homer and Clark (2003), reproduced in Chapter 6.

```
bivariate A
  dfreq [2]
  dbvsev [0 1 2] [0 1 2] [[0.4 0 0] [0.3 0.3 0] [0 0 0]]
```

```
agg HC
  5 claims
  dsev [200000 400000 600000 800000 1000000] [0.378 0.235 0.146 0.091 0.15]
  occurrence net of
    400000 xs 600000
  mixed gamma 0.2
```

```
bivariate HC3
  4 claims
  dbvsev [0 200000 400000 600000 800000 1000000] [0 200000 400000 600000 800000 1000000
    1200000] [[0.08389999999999327 0.014699999999998822 0.0012999999999998956
    9.9999999999992e-05 0 0 0] [0.27979999999997757 0.13289999999998933
    0.03159999999999747 0.004999999999999599 0.0005999999999999518 9.9999999999992e-05
    0] [0.04149999999999668 0.03209999999999742 0.012499999999998998
    0.003199999999999744 0.0005999999999999518 9.9999999999992e-05 0]
    [0.030699999999997542 0.03299999999999736 0.01769999999999858 0.006399999999999488
    0.0016999999999998637 0.000399999999999968 9.9999999999992e-05]
    [0.022799999999998172 0.03129999999999749 0.021499999999998274 0.009899999999999208
    0.003999999999997275 0.0008999999999999279 0.000199999999999984]
    [0.033756260434097296 0.05659432387309546 0.047378964941596206 0.02644407345579788
    0.011118530884799109 0.003706176961599703 0.0010016694490799197]]
  mixed gamma 0.5
```

```
bivariate HC4
  10 claims
  dbvsev [0 200000 400000 600000 800000] [0 1 2 3 4 5] [[0 0.10000000000000002 0 0 0 0]
    [0.43800000000000006 0 0 0 0 0] [0.24600000000000002 0 0 0 0 0] [0.13800000000000004
    0 0 0 0 0] [0.07800000000000001 0 0 0 0 0]]
  mixed gamma 0.31622776601683794
```

```
bivariate HC4b
  10 claims
  dbvsev [0 200000 400000 600000 800000] [0 1 2 3 4 5] [[0 0.10000000000000002 0 0 0 0]
    [0.43800000000000006 0 0 0 0 0] [0.24600000000000002 0 0 0 0 0] [0.13800000000000004
    0 0 0 0 0] [0.07800000000000001 0 0 0 0 0]]
  mixed gamma 0.31622776601683794
```

Bear and Nemlick on adjustable treaty features

From Bear and Nemlick (1990), reproduced in Chapter 7.

```
port BearNemlick.One
  agg Class1
    9000 premium at 0.04 lr
    160 xs 0
    sev 40 * pareto 0.9 + -40
    mixed gamma 0.07879574790219362
  agg Class2
    3000 premium at 0.03 lr
    160 xs 0
    sev 40 * pareto 0.95 + -40
    mixed gamma 0.22335691114062584
```

```
port BearNemlick.Two
  agg Class1
    2000 premium at 0.1 lr
    700 xs 0
    sev 300 * pareto 1.5 + -300
    mixed gamma 0.09022811434760547
  agg Class2
    2000 premium at 0.14 lr
    700 xs 0
    sev 300 * pareto 1.3 + -300
    mixed gamma 0.09869275424396479
  agg Class3
    2000 premium at 0.21 lr
    700 xs 0
    sev 300 * pareto 1.1 + -300
    mixed gamma 0.12408498159738195
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  poisson
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  poisson
  aggregate net of
    360 xs 0
```

```
xpnl BN.TreatyII
  1172 premium
  less
    agg BN.Two
      [2000 2000 2000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      occurrence net of
        700 xs 0 deposit 1172 reinstatements [0 0 0] as Layer
      poisson
```

```
agg BN.TwoAlt
  [2000 2000 2000] premium at [0.1 0.14 0.21] lr
  700 xs 0
  sev 300 * pareto [1.5 1.3 1.1] + -300
  poisson
  aggregate ceded to
    2800 xs 0
```

```
xpnl BN.TreatyIII
  10000 premium
  less
    agg BN.Three
      [4500 4500 1000] premium at [0.032 0.038 0.035] lr
      400 xs 0
      sev 100 * pareto 1.1 + -100
      poisson
      aggregate net of
        10000 xs 0 deposit 350 corridor 1 po 1 xs 1 as Corridor
```

```
xpnl BN.TreatyIIIb
  10000 premium
  less
    agg BN.Three
      [4500 4500 1000] premium at [0.032 0.038 0.035] lr
      400 xs 0
      sev 100 * pareto 1.1 + -100
      poisson
      aggregate net of
        10000 xs 0 deposit 366.9658642415209 corridor 1 po 0.9537671868292504 xs
          0.9537671868292504 as Corridor
```

```
pnl BN.TreatyIV
  retro basic 0 lcm 1.3333333333 min 360 max 1200 premium
  less
    agg BN.Four
      [9000 3000] premium at [0.04 0.03] lr
      160 xs 0
      sev 40 * pareto [0.9 0.95] + -40
      poisson
```

```
xpnl BN.TreatyV
  18000 premium
  less
    agg BN.Five
      [6000 6000 6000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      poisson
      aggregate net of
        80% po 100000 xs 0 deposit 5625 pc 0.25 after 0.2 as "PC layer"
```

```
xpnl BN.TreatyVI
  25000 premium
  less
```

```
    agg BN.Six
      25000 premium at 0.1 lr
      900 xs 0
      sev 100 * pareto 0.98 + -100
      poisson
      aggregate net of
        100000 xs 0 deposit 5000 slide 0.4 at 0.35 and 0.25 at 0.55 and 0.2 at 0.65 as
          Slide
```

```
xpnl BN.TreatyVI
  25000 premium
  less
    agg BN.Six
      25000 premium at 0.1 lr
      900 xs 0
      sev 100 * pareto 0.95 + -100
      poisson
      aggregate net of
        100000 xs 0 deposit 5000 slide 0.4 at 0.35 and 0.25 at 0.55 and 0.2 at 0.65 as
          Slide
```

and 4 more of this shape, differing only in their numbers.

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  mixed gamma 0.22360679774997896
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  mixed gamma 0.31622776601683794
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  mixed gamma 0.22360679774997896
  aggregate net of
    360 xs 0
```

```
agg BN.One
  [9000 3000] premium at [0.04 0.03] lr
  160 xs 0
  sev 40 * pareto [0.9 0.95] + -40
  mixed gamma 0.31622776601683794
  aggregate net of
    360 xs 0
```

```
xpnl BN.TreatyII
  1172 premium
  less
    agg BN.Two
      [2000 2000 2000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      occurrence net of
        700 xs 0 deposit 1172 reinstatements [0 0 0] as Layer
      mixed gamma 0.22360679774997896
```

```
xpnl BN.TreatyII
  1172 premium
  less
    agg BN.Two
      [2000 2000 2000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      occurrence net of
        700 xs 0 deposit 1172 reinstatements [0 0 0] as Layer
      mixed gamma 0.31622776601683794
```

```
xpnl BN.TreatyIII
  10000 premium
  less
    agg BN.Three
      [4500 4500 1000] premium at [0.032 0.038 0.035] lr
      400 xs 0
      sev 100 * pareto 1.1 + -100
      mixed gamma 0.22360679774997896
      aggregate net of
        10000 xs 0 deposit 350 corridor 1 po 1 xs 1 as Corridor
```

```
xpnl BN.TreatyIII
  10000 premium
  less
    agg BN.Three
      [4500 4500 1000] premium at [0.032 0.038 0.035] lr
      400 xs 0
      sev 100 * pareto 1.1 + -100
      mixed gamma 0.31622776601683794
      aggregate net of
        10000 xs 0 deposit 350 corridor 1 po 1 xs 1 as Corridor
```

```
pnl BN.TreatyIV
  retro basic 0 lcm 1.3333333333 min 360 max 1200 premium
  less
    agg BN.Four
      [9000 3000] premium at [0.04 0.03] lr
      160 xs 0
      sev 40 * pareto [0.9 0.95] + -40
      mixed gamma 0.22360679774997896
```

```
pnl BN.TreatyIV
  retro basic 0 lcm 1.3333333333 min 360 max 1200 premium
  less
    agg BN.Four
      [9000 3000] premium at [0.04 0.03] lr
      160 xs 0
      sev 40 * pareto [0.9 0.95] + -40
      mixed gamma 0.31622776601683794
```

```
xpnl BN.TreatyV
  18000 premium
  less
    agg BN.Five
      [6000 6000 6000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      mixed gamma 0.22360679774997896
      aggregate net of
        80% po 100000 xs 0 deposit 5625 pc 0.25 after 0.2 as "PC layer"
```

```
xpnl BN.TreatyV
  18000 premium
  less
    agg BN.Five
      [6000 6000 6000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      mixed gamma 0.31622776601683794
      aggregate net of
        80% po 100000 xs 0 deposit 5625 pc 0.25 after 0.2 as "PC layer"
```

```
xpnl BN.TreatyVI
  25000 premium
  less
    agg BN.Six
      25000 premium at 0.1 lr
      900 xs 0
      sev 100 * pareto 0.98 + -100
      mixed gamma 0.22360679774997896
      aggregate net of
        100000 xs 0 deposit 5000 slide 0.4 at 0.35 and 0.25 at 0.55 and 0.2 at 0.65 as
          Slide
```

```
xpnl BN.TreatyVI
  25000 premium
  less
    agg BN.Six
      25000 premium at 0.1 lr
      900 xs 0
      sev 100 * pareto 0.98 + -100
      mixed gamma 0.31622776601683794
      aggregate net of
        100000 xs 0 deposit 5000 slide 0.4 at 0.35 and 0.25 at 0.55 and 0.2 at 0.65 as
          Slide
```

```
xpnl BN.TreatyIIp
  1172 premium
  less
    agg BN.Two
      [2000 2000 2000] premium at [0.1 0.14 0.21] lr
      700 xs 0
      sev 300 * pareto [1.5 1.3 1.1] + -300
      occurrence net of
        700 xs 0 deposit 1172 reinstatements [1 1 1] as Layer
      poisson
```

Wang and risk loads by layer

From Wang (2000), reproduced in Chapter 8.

```
agg Wang.Layers
  1 claim
  10000000 xs 0
  sev 2000 * lomax 1.2
  occurrence ceded to
    50000 xs 0 and
    50000 xs 50000 and
    100000 xs 100000 and
    100000 xs 200000 and
    100000 xs 300000 and
    100000 xs 400000 and
    500000 xs 500000 and
    1000000 xs 1000000 and
    3000000 xs 2000000 and
    5000000 xs 5000000
  fixed
```

```
distortion wang_a wang 0.1
```

```
distortion ph_r ph 0.9245
```

```
distortion tmp ph 0.9
```

```
distortion tmp ph 0.95
```

and 4 more of this shape, differing only in their numbers.

Wang and the pricing of weather options

From Wang (2002), reproduced in Chapter 9.

```
agg HDD
  1 claim
  dsev [972.5 1147 1244 901 1573 1055 1488 1065.5 1018.5 1155 1474.5 1129.5 1077.5
    1129.5 1090.5 938.5 1199.5 1156 1040 940.5 1090.5 1517.5]
  fixed
```

```
distortion wang25 wang 0.25
```

Denuit and conditional mean risk sharing

From Denuit (2019), reproduced in Chapter 10.

```
port D61a
  agg P1
    0.08 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P2
    0.08 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
  agg P3
    0.1 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P4
    0.1 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
```

```
port D61b
  agg P1
    0.08 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P2
    0.08 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
  agg P3
    0.1 claims
    dsev [1 2 3 4] [0.1 0.2 0.3 0.4]
    poisson
  agg P4
    0.1 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
```

```
port Denuit61
  agg P1
    0.08 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
```

```
  agg P2
    0.08 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
  agg P3
    0.1 claims
    dsev [1 2 3 4] [0.1 0.2 0.4 0.3]
    poisson
  agg P4
    0.1 claims
    dsev [1 2 3 4] [0.15 0.25 0.3 0.3]
    poisson
```

Denuit, Hieber and Robert on mortality credits

From Denuit et al. (2022), reproduced in Chapter 11.

```
port Fund100
  agg Low
    6 claims
    dsev [1]
    binomial 0.1
  agg High
    8 claims
    dsev [3]
    binomial 0.2
```

```
port Fund500
  agg Low
    30 claims
    dsev [1]
    binomial 0.1
  agg High
    40 claims
    dsev [3]
    binomial 0.2
```

and 1 more of this shape, differing only in their numbers.

Denuit et al. on comonotonic allocations

From Denuit, Dhaene, et al. (2025), reproduced in Chapter 12.

```
port Denuit.Pool
  hints{log2=16; bs=0.002; normalize=False; padding=1}
  agg Agent1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg Agent2
    1 claim
    sev 2 * expon splice [0 10]
```

```
    fixed
  agg Agent3
    1 claim
    sev 2 * gamma [1 8] wts [0.504541349633 0.495458650367] splice [0 0] [30 30]
    fixed
```

```
port Denuit.PoolAlt
  hints{log2=16; bs=0.002; normalize=False; padding=1}
  agg A1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg A2
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg A3
    1 claim
    sev 2 * gamma [1 8] wts=2
    fixed
```

```
port Denuit.PoolB
  hints{log2=16; bs=0.002; normalize=False; padding=1}
  agg B1
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg B2
    1 claim
    sev 2 * expon splice [0 10]
    fixed
  agg B3
    1 claim
    sev 2 * gamma [1 8] wts=2 splice [0 0] [30 30]
    fixed
```

Denuit, Ortega-Jiménez and Robert on conditional expectations given the sum

From Denuit, Ortega-Jiménez, et al. (2025), reproduced in Chapter 13.

```
port K
  hints{log2=20; bs=0.01; normalize=False; padding=1}
  agg X
    1 claim
    sev pareto 3.5
    fixed
  agg Y
    1 claim
    sev pareto 2.5
    fixed
```

```
port K
  hints{log2=20; bs=0.01; normalize=False; padding=1}
  agg X
    1 claim
    sev pareto 2.5
    fixed
  agg Y
    1 claim
    sev pareto 1.5
    fixed
```

and 1 more of this shape, differing only in their numbers.

```
port K
  hints{log2=18; bs=0.002; normalize=False; padding=1}
  agg X
    1 claim
    sev lomax 3.5
    fixed
  agg Y
    1 claim
    sev lomax 3
    fixed
```

Gerber and compound Poisson stop loss premiums

From Gerber (1982), reproduced in Chapter 14.

```
agg Gerber
  1 claim
  sev 2 * uniform + 1
  poisson
  hints{log2=18; bs=0.0009765625}
```

```
agg Gerber
  10 claims
  sev 2 * uniform + 1
  poisson
  hints{log2=18; bs=0.0078125}
```

```
agg Gerber
  100 claims
  sev 2 * uniform + 1
  poisson
```

Robertson and the sum of uniform claims

From Robertson (1992), reproduced in Chapter 15.

```
agg Robertson
  5 claims
  sev chistogram xps [0 0.2 0.4 0.6 0.8 1] [0.2 0.2 0.2 0.2 0.2]
  fixed
```

```
agg Robertson2
  5 claims
  sev uniform
  fixed
```

Embrechts, Grübel and Pitts on discretization error

From Embrechts et al. (1993), reproduced in Chapter 16.

```
agg EGP
  20 claims
  sev expon 1
  poisson
```

Grübel and Hermesmeier on aliasing and exponential tilting

From Grübel and Hermesmeier (1999), reproduced in Chapter 17.

```
agg Levy
  20 claims
  sev levy
  poisson
```

Pittarello, Luini and Marchione's GEMAct package

From Pittarello et al. (2024), reproduced in Chapter 18.

```
agg PLM.XL
  4 claims
  20 xs 5
  sev gamma 5 !
  poisson
```

```
agg PLM.Layer1
  0.5 claims
  100 xs 100
  sev 83.34 * genpareto 0.834 !
  poisson
```

```
agg PLM.Layer3
  0.5 claims
  100 xs 100
  sev 83.34 * genpareto 0.834 !
  poisson
  aggregate ceded to
    200 xs 0
```

```
xpnl PLM.Layer2
  1 premium
  less
    agg PLM.Layer2Gross
      0.5 claims
      sev 83.34 * genpareto 0.834
      occurrence net of
        200 xs 100 deposit 1 reinstatements [0.6 0.6] as Layer
      poisson
```

```
agg PLM.RS
  0.5 claims
  100 xs 0
  sev 100 * pareto 1.2 + -100 !
  poisson
```

```
agg PLM.Layer2Net
  0.5 claims
  200 xs 100
  sev 83.34 * genpareto 0.834 !
  poisson
```

```
agg PLM.Table1
  3 claims
  inf xs 10000
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
```

```
agg PLM.Gross
  3 claims
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
```

```
agg PLM.XL1M
  3 claims
  1000000 xs 10000
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
```

```
agg PLM.SL
  3 claims
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
  aggregate ceded to
    1000000 xs 50000
```

```
agg PLM.XLAgg
  3 claims
  1000000 xs 10000
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
  aggregate ceded to
    1000000 xs 50000
```

```
agg PLM.Cut
  3 claims
  819150 xs 10000
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
```

```
agg PLM.Cut
  3 claims
  1638300 xs 10000
  sev lognorm 84541.68 cv 2.1022565437545127
  poisson
```

and 10 more of this shape, differing only in their numbers.

Embrechts and Frei's three discretization designs

From Embrechts and Frei (2009), reproduced in Chapter 19.

```
agg EF
  50 claims
  sev expon
  poisson
```

```
agg EF1
  20 claims
  sev 3 * pareto 4 + -3
  poisson
```

Schaller and Temnov on a severity with no mean

From Schaller and Temnov (2008), reproduced in Chapter 20.

```
agg SchallerTemnov
  18 claims
  sev 12000 * genpareto 1 + 7000
  poisson
  hints{log2=20; bs=4.76837158203125; padding=2; normalize=False}
```

Bruno et al. on thresholded direct convolution

From Bruno et al. (2006), reproduced in Chapter 21.

```
agg B
  504.814259 claims
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.0103301 0.030799 0.0293511
    0.0103301 0.0730414 0.0111568 0.0264554 0.1002133 0.0815418 0.0252146 0.0212857
    0.0254214 0.0991756 0.4556837]
  poisson
```

```
agg B8
  dfreq [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17] [0.79870839980732
    0.168928395537711 0.0275621536510322 0.00411140723243176 0.000591554176846273
    8.41033531369783e-05 1.19716064327213e-05 1.71994460232325e-06 2.50751602571129e-07
    3.72405307143863e-08 5.65056685224704e-09 8.77932283380845e-10 1.39934149169577e-10
    2.29163277016947e-11 3.86088331748775e-12 6.69927670844654e-13 1.19834141403275e-13
    2.211580538677e-14]
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.0103301 0.030799 0.0293511
    0.0103301 0.0730414 0.0111568 0.0264554 0.1002133 0.0815418 0.0252146 0.0212857
    0.0254214 0.0991756 0.4556837]
```

```
agg B7
  91000 claims
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.0103301 0.030799 0.0293511
    0.0103301 0.0730414 0.0111568 0.0264554 0.1002133 0.0815418 0.0252146 0.0212857
    0.0254214 0.0991756 0.4556837]
  poisson
```

Gollini and Rougier on a catastrophe model event loss table

From Gollini and Rougier (2016), reproduced in Chapter 22.

```
agg Hurricane
  2.974183 claims
  dsev [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] [0.58889746 0.21928748
    0.0672295 0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063 0.00941256
    0.00198404 0.00452247 7.436e-05 0.00025122 0.00253892 0.00435067 0.0007371 1.486e-05
    4.95e-06 4.95e-06]
  poisson
```

```
agg Hurricane.Capped
  2.974183 claims
  5 xs 0
  dsev [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] [0.58889746 0.21928748
    0.0672295 0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063 0.00941256
    0.00198404 0.00452247 7.436e-05 0.00025122 0.00253892 0.00435067 0.0007371 1.486e-05
    4.95e-06 4.95e-06]
  poisson
```

```
agg Hurricane.Gamma
  2.974183 claims
  sev gamma [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 24] cv 0.5 wts [0.58889746
    0.21928748 0.0672295 0.05374453 0.01379374 0.01411 0.00846236 0.0061182 0.00446063
    0.00941256 0.00198404 0.00452247 7.436e-05 0.00025122 0.00253892 0.00435067
    0.0007371 1.486e-05 4.95e-06 4.95e-06]
  poisson
```

Wilson and Keich on accurate FFT convolution

From Wilson and Keich (2016), reproduced in Chapter 23.

```
agg Toy
  dfreq [2]
  dsev [1 2 3] [0.99999 1e-05 1e-20]
```

```
nttagg Wilson
  dfreq [2]
  dsev [1 2 3] [9999899999999999999/10000000000000000000 1/100000
    1/10000000000000000000]
```

Wilson and Keich's sisFFT for small tail probabilities

From Wilson and Keich (2017), reproduced in Chapter 24.

```
agg WK
  dfreq [2]
  dsev [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
    30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
    58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
    86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
    110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127]
    [0.05698071412216734 0.06620214484224507 0.07439431506585613 0.0808594823064372
    0.08500523661668595 0.08643386270782825 0.08500523661668595 0.0808594823064372
    0.07439431506585613 0.06620214484224507 0.05698071412216734 0.0474359096066097
    0.03819531491351482 0.02974654116430425 0.022407137097008607 0.016325247924520128
    0.011504207774114473 0.007841103119752254 0.005169173762132145 0.003296010711360087
    0.0020327296165158883 0.0012125345785902368 0.0006995715951974206
    0.0003903853710257675 0.00021070691529164931 0.00010999865526234663
    5.554173985207159e-05 2.7125334425354087e-05 1.2813100714451682e-05
    5.854056311225143e-06 2.586920418962627e-06 1.105688414827683e-06
    4.570944696450248e-07 1.827690656905043e-07 7.068429552096851e-08
    2.644031917226084e-08 9.566079154710707e-09 3.347532207218618e-09
    1.1330239115747578e-09 3.709171412794566e-10 1.1744600532204117e-10
    3.59685651045085e-11 1.0654461224833036e-11 3.0525542488755614e-12
    8.458995729331485e-13 2.2672412675735592e-13 5.8776021715289e-14
    1.473757926702887e-14 3.574173561367906e-15 8.38394921987302e-16
    1.902151734895072e-16 4.174122530025176e-17 8.859490495024046e-18
    1.818761770055943e-18 3.611323587025658e-19 6.935543648882527e-20
    1.2883033361832961e-20 2.3146177051578186e-21 4.0222024620549385e-22
    6.760395460348606e-23 1.0990154370000577e-23 1.7280607304882195e-24
    2.628074830109522e-25 3.865805235535093e-26 5.500038508200373e-27
```

```
    7.568590490625302e-28 1.0073671919033027e-28 1.2968331832718913e-29
    1.6147449836629747e-30 1.9446761623098232e-31 2.2652396490885474e-32
    2.5521402435349255e-33 2.781111713376742e-34 2.931270180098265e-35
    2.9882490005560692e-36 2.9464646373490424e-37 2.81001863961723e-38
    2.5920339497460392e-39 2.3125742607080913e-40 1.9956032700263374e-41
    1.6656212138890558e-42 1.3446269064639106e-43 1.0499070900667768e-44
    7.929092000468263e-46 5.791880098166922e-47 4.092033548360507e-48
    2.796290748421717e-49 1.8482000741056655e-50 1.1815147781965363e-51
    7.305548867286579e-53 4.3690806287813747e-54 2.5272650181927225e-55
    1.4139531828455055e-56 7.651433198651694e-58 4.004737637015077e-59
    2.0273506236978068e-60 9.92675213472881e-62 4.701203037982131e-63
    2.1534478334709612e-64 9.540765472727624e-66 4.088420924459984e-67
    1.6945387385131509e-68 6.79314552283295e-70 2.6339883633345665e-71
    9.878256594055475e-73 3.583193580753452e-74 1.2571403239902966e-75
    4.2659991042454884e-77 1.4001716502139405e-78 4.4449340074571594e-80
    1.364812125339333e-81 4.0532551594801475e-83 1.1642829388275267e-84
    3.234719599335948e-86 8.692371190637349e-88 2.2592453900859697e-89
    5.679525621992663e-91 1.3809699827592993e-92 3.247730108931048e-94
    7.387528029916211e-96 1.625331188442783e-97 3.4586619315136957e-99
    7.118653852801502e-101 1.4171347738069365e-102 2.7286507445183277e-104
    5.081691135606408e-106 9.153603493842637e-108 1.5947750937110134e-109]
```

```
nttagg WKExact
  dfreq [2]
  dsev [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
    30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
    58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
    86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
    110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127] [1205358117
    64446499302627767141300693656836408314036477899945172967332002636542964093996091960 6
    86262214058546805318733319372 8/2115379100128795384141269704787538749829391031397325 1
    56479702597911102712337040158720475206605482703083788986571601650839700 3 14004263358
    29845571578239240903718494240992908282963197796760858189765888474030421110930534260 9
    94701597220333948174021427200/2115379100128795384141269704787538749829391031397325 15
    64797025979111027123370401587204752066054827030837889865716016508397003 157372179258
    70883236071749785133813154663534520204378281828859226984346822449726357173903251053 1
    63235598066422342555522826 24/2115379100128795384141269704787538749829391031397325 156
    47970259791110271233704015872047520660548270308378898657160165083970 03 1710484589182
    71398473385047278500960604216834855898300044105363232283467012828959445278442826109 2
    75294416681751591619710156 8/2115379100128795384141269704787538749829391031397325 1564
    79702597911102712337040158720475206605482703083788986571601650839700 3 17981830094044
    04637114502989356361510338483795743649982320353691105560694753443765327880796704603 2
    60059644846490769095131136/2115379100128795384141269704787538749829391031397325 15647
    97025979111027123370401587204752066054827030837889865716016508397003 182840386715541
    59003520233158334254305025384189384888087133797473601071364849438392445578265847175 8
    95788387215054494907760 64/2115379100128795384141269704787538749829391031397325 156479
    70259791110271233704015872047520660548270308378898657160165083970 03 ...
```

The declaration runs to 28,081 characters and is cut short here; `reproductions.agg` contains it in full.

```
nttagg DiceOfDice
  dfreq [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
    30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
    58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
    86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
    110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130
    131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
    152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172
```

```
    173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193
    194 195 196 197 198 199 200]
  dsev [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
    59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86
    87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110
    111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
    132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152
    153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173
    174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194
    195 196 197 198 199 200]
```

Asmussen, Jensen and Rojas-Nandayapa on the lognormal left tail

From Asmussen et al. (2016), reproduced in Chapter 25.

```
agg S
  dfreq [4]
  sev lognorm 0.25
```

```
agg S
  dfreq [64]
  sev lognorm 0.25
```

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

Ben Amar et al. on state-dependent importance sampling

From Ben Amar et al. (2023), reproduced in Chapter 26.

```
agg S10
  dfreq [10]
  sev lognorm 0.6907755278982137
```

```
agg S9
  dfreq [9]
  sev lognorm 0.6907755278982137
```

and 4 more of this shape, differing only in their numbers.

Ben Rached, Hoel and Meo on the FFT accuracy floor

From Ben Rached et al. (2024), reproduced in Chapter 27.

```
agg S
  dfreq [16]
  sev lognorm 0.125
```

```
agg L
  dfreq [16]
  sev 0.1 * levy
```

Liu, Wu and Yao on k-moment tilting

From Liu et al. (2026), reproduced in Chapter 28.

```
agg T
  dfreq [1]
  sev pareto 2.5
```

```
agg T
  dfreq [5]
  sev pareto 2.5
```

```
agg T
  dfreq [1]
  sev 1.1 * pareto 2.6
```

```
agg T
  dfreq [1]
  sev 1.2 * pareto 2.7
```

```
port Paretos
  hints{log2=22;bs=1/8;padding=1;normalize=False}
  agg P0
    dfreq [1]
    sev pareto 2.5
  agg P1
    dfreq [1]
    sev 1.1 * pareto 2.6
  agg P2
    dfreq [1]
    sev 1.2 * pareto 2.7
  agg P3
    dfreq [1]
    sev 0.9 * pareto 2.8
  agg P4
    dfreq [1]
    sev 0.8 * pareto 2.9
```

```
port P
  agg P0
    dfreq [1]
    sev pareto 2.5
  agg P1
    dfreq [1]
    sev 1.1 * pareto 2.6
  agg P2
    dfreq [1]
    sev 1.2 * pareto 2.7
  agg P3
    dfreq [1]
    sev 0.9 * pareto 2.8
  agg P4
    dfreq [1]
    sev 0.8 * pareto 2.9
```

```
port LT
  hints{log2=18;bs=1/1024;padding=1;normalize=False}
  agg C0
    dfreq [1]
    sev expon + 1
  agg C1
    dfreq [1]
    sev 0.8333333333333334 * expon + 1.1
  agg C2
    dfreq [1]
    sev 0.7142857142857143 * expon + 1.2
  agg C3
    dfreq [1]
    sev 0.6666666666666666 * expon + 0.9
  agg C4
    dfreq [1]
    sev 0.5555555555555556 * expon + 0.8
```

Jin, Provost and Ren's moment-based density approximations

From Jin et al. (2016), reproduced in Chapter 30.

```
agg J
  3 claims
  sev 2 * gamma 3
  poisson
```

```
agg J
  15 claims
  sev 2 * invgauss 1.5
  poisson
```

```
agg J
  50 claims
  sev 2 * gamma 3
  mixed gamma 0.3162277660168379
```

```
agg J
  50 claims
  sev 2 * invgauss 1.5
  mixed gamma 0.3162277660168379
```

```
agg J
  3 claims
  sev 100 * lomax 20
  poisson
```

```
agg J
  6 claims
  sev 100 * lomax 20
  mixed gamma 0.5773502691896257
```

```
agg NB
  50 claims
  dsev [1]
  mixed gamma 0.31622776601683794
```

```
agg U
  dfreq [1]
  sev 2 * gamma 2
```

```
agg V
  dfreq [1]
  sev 1.5 * gamma 3
```

```
agg X
  15 claims
  sev 2 * gamma 2
  poisson
```

```
agg Y
  16 claims
  sev 1.5 * gamma 3
  poisson
```

Provenance of the named severity curves

From the papers cited in Chapter 31.

```
agg T
  1 claim
  sev lognorm 150000 cv 0.25
  fixed
```

```
agg T
  1 claim
  sev lognorm 10000 cv 4.953
  fixed
```

```
agg T
  1 claim
  sev 50000 * pareto 1.5
  fixed
```

```
agg T
  1 claim
  sev 50000 * lomax 1.5
  fixed
```

```
agg T
  1 claim
  sev 100 * lomax 20
  fixed
```

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

```
agg T
  1 claim
  sev gamma 6 cv 0.5774
  fixed
```

```
agg T
  1 claim
  sev 11048 * lognorm 2.29
  fixed
```

```
agg T
  1 claim
  sev chistogram xps [0 1000 5000 6000 7000 8000 9000 10000 12500 15000 17500 20000
    25000 35000 50000 75000 100000 125000 150000 175000 200000 225000 250000 300000]
    [0.38935 0.38935 0.00568 0.00543 0.00517 0.00495 0.00473 0.01098 0.00989 0.00896
    0.00815 0.01426 0.02237 0.02353 0.02459 0.01517 0.01021 0.00732 0.00547 0.00423
    0.00337 0.00274 0.0241]
  fixed
```

```
agg XPL
  1 claim
  1000000 xs 0
  sev 50000 * lomax 1.5
  fixed
```

```
agg J
  3 claims
  sev gamma 6 cv 0.5774
  poisson
```

29 chapters build 221 distinct programs between them, of which 144 are printed above and 77 are further declarations of a shape already shown.

References

Arbenz, P., Embrechts, P., & Puccetti, G. (2011). The AEP algorithm for the fast computation of the distribution of the sum of dependent random variables. *Bernoulli*, *17*(2), 562–591. https://doi.org/10.3150/10-BEJ284

Asmussen, S., Jensen, J. L., & Rojas-Nandayapa, L. (2016). Exponential Family Techniques for the Lognormal Left Tail. *Scandinavian Journal of Statistics*, *43*(3), 774–787. https://doi.org/10.1111/sjos.12203

Bahadur, R. R., & Rao, R. R. (1960). On deviations of the sample mean. *The Annals of Mathematical Statistics*, *31*(4), 1015–1027.

Bakar, S. A. A., Nadarajah, S., & Ngataman, N. (2022). A family of density-hazard distributions for insurance losses. *Communications in Statistics - Simulation and Computation*. https://doi.org/10.1080/03610918.2020.1784430

Bear, R. A., & Nemlick, K. J. (1990). Pricing the impact of adjustable features and loss sharing provisions of reinsurance treaties. *Proceedings of the Casualty Actuarial Society*, *77*(147), 86–87. https://doi.org/10.1016/0167-6687(93)91078-9

Ben Amar, E., Ben Rached, N., Haji-Ali, A.-L., & Tempone, R. (2023). State-dependent importance sampling for estimating expectations of functionals of sums of independent random variables. *Statistics and Computing*. https://doi.org/10.1007/s11222-022-10202-2

Ben Rached, N., Hoel, H., & Meo, J. V. (2024). A fast and accurate numerical method for the left tail of sums of independent random variables. *Statistics and Computing*. https://doi.org/10.1007/s11222-024-10514-5

Berens, R. M. (1997). Reinsurance contracts with a multi-year aggregate limit. *Casualty Actuarial Society Forum*.

Bernegger, S. (1997). The Swiss Re exposure curves and the MBBEFD distribution class. *ASTIN Bulletin*, *27*(1), 99–111.

Bodoff, N. M. (2017). An Actuarial Model of Excess of Policy Limits Losses. *Variance*, *11*. https://doi.org/10.66573/001c.116059

Braithwaite, P., & Ware, B. C. (1997). Pricing extra-contractual obligations and excess of policy limits exposures in Clash Reinsurance Treaties. *Casualty Actuarial Society Forum*. https://doi.org/10.1016/s0167-6687(97)89196-2

Brown, L. D., Cai, T. T., & DasGupta, A. (2001). Interval Estimation for a Binomial Proportion. *Statistical Science*. https://doi.org/10.1214/ss/1009213286

Bruno, M. G., Camerini, E., Manna, A., & Tomassetti, A. (2006). A new method for evaluating the distribution of aggregate claims. *Applied Mathematics and Computation*. https://doi.org/10.1016/j.amc.2005.09.038

Clark, D. R. (2005). Reinsurance Applications for the RMK Framework, 353–366.

Cummins, J. D., Dionne, G., McDonald, J. B., & Pritchett, B. M. (1990). Applications of the GB2 family of distributions in modeling insurance loss processes. *Insurance: Mathematics and Economics*, *9*(4), 257–272. https://doi.org/10.1016/0167-6687(90)90003-V

Denuit, M. M. (2019). Size-biased transform and conditional mean risk sharing, with application to p2p insurance and tontines. *ASTIN Bulletin*, *49*(3), 591–617. https://doi.org/10.1017/asb.2019.24

Denuit, M. M. (2020). Investing in your own and peers' risks: the simple analytics of P2P insurance. *European Actuarial Journal*. https://doi.org/10.1007/s13385-020-00238-x

Denuit, M. M., Dhaene, J., Ghossoub, M., & Robert, C. Y. (2025). Comonotonicity and Pareto optimality, with application to collaborative insurance. *Insurance: Mathematics and Economics*, *120*,

1–16. https://doi.org/10.1016/j.insmatheco.2024.11.001
Denuit, M. M., Hieber, P., & Robert, C. Y. (2022). Mortality Credits Within Large Survivor Funds. *ASTIN Bulletin*, *52*(3), 813–834. https://doi.org/10.1017/asb.2022.13
Denuit, M. M., Ortega-Jiménez, P., & Robert, C.-Y. (2025). Conditional expectations given the sum of independent random variables with regularly varying densities. *ASTIN Bulletin*. https://doi.org/10.1017/asb.2025.11
Embrechts, P., & Frei, M. (2009). Panjer recursion versus FFT for compound distributions. *Mathematical Methods of Operations Research*, *69*(3), 497–508. https://doi.org/10.1007/s00186-008-0249-2
Embrechts, P., Grübel, R., & Pitts, S. M. (1993). Some applications of the fast Fourier transform algorithm in insurance mathematics. *Statistica Neerlandica*, *47*(1), 59–75. https://doi.org/10.1111/j.1467-9574.1993.tb01406.x
Gerber, H. U. (1982). On the numerical evaluation of the distribution of aggregate claims and its stop-loss premiums. *Insurance Mathematics and Economics*, *1*(1), 13–18. https://doi.org/10.1016/0167-6687(82)90016-6
Goffard, P.-O., & Laub, P. J. (2020). Orthogonal polynomial expansions to evaluate stop-loss premiums. *Journal of Computational and Applied Mathematics*, *370*. https://doi.org/10.1016/j.cam.2019.112648
Gollini, I., & Rougier, J. (2016). Rapidly bounding the exceedance probabilities of high aggregate losses. *The Journal of Operational Risk*, 1–33. https://doi.org/10.21314/JOP.2016.179
Grübel, R., & Hermesmeier, R. (1999). Computation of Compound Distributions I: Aliasing Errors and Exponential Tilting. *Astin Bulletin*, *29*(2), 197–214. https://doi.org/10.2143/AST.29.2.504611
Grübel, R., & Hermesmeier, R. (2000). Computation of Compound Distributions II: Discretization Errors and Richardson Extrapolation. *ASTIN Bulletin*, *30*(2), 309–332. https://doi.org/10.2143/AST.30.2.504638
Hilbe, J. M. (2014). *Modeling count data*. Cambridge University Press.
Homer, D. L., & Clark, D. R. (2003). Insurance Applications of Bivariate Distributions. *Proceedings of the Casualty Actuarial Society*, *90*(iid), 274–307. http://www.casact.org/pubs/proceed/proceed03/03274.pdf
Jin, T., Provost, S. B., & Ren, J. (2016). Moment-based density approximations for aggregate losses. *Scandinavian Actuarial Journal*, *2016*(3), 216–245. https://doi.org/10.1080/03461238.2014.921640
Kang, J. S. J., Provost, S. B., & Ren, J. (2019). Moment-Based Density Approximation Techniques as Applied to Heavy-tailed Distributions. *International Journal of Statistics and Probability*. https://doi.org/10.5539/ijsp.v8n3p1
Keich, U. (2005). sFFT: A Faster Accurate Computation of the p-Value of the Entropy Score. *Journal of Computational Biology*, *12*(4), 416–430.
Klugman, S. A., Panjer, H. H., & Willmot, G. E. (2012). *Loss Models: from data to decisions* (Fourth.). John Wiley & Sons, Inc.
Lau, H.-S. (1984). An Effective Approach for Estimating the Aggregate Loss of an Insurance Portfolio. *The Journal of Risk and Insurance*. https://doi.org/10.2307/252798
Lindsay, B. G., Pilla, R. S., & Basak, P. (2000). Moment-based approximations of distributions using mixtures: Theory and applications. *Annals of the Institute of Statistical Mathematics*, *52*(2), 215–230.
Liu, Z., Wu, J., & Yao, J. (2026). Estimating tail probabilities of heavy-tailed distributions using k-moment tilting. *Operations Research Letters*. https://doi.org/10.1016/j.orl.2026.107509
Mack, T., & Fackler, M. (2003). Exposure-rating in liability reinsurance. *Blätter der DGVFM*, *26*(2), 229–238.
Mata, A. J., & Verheyen, M. A. (2005). An Improved Method for Experience Rating Excess of Loss Treaties using Exposure Rating Techniques. *CAS Forum*.

McDonald, J. B., & Richards, D. O. (1987). Hazard Rates and Generalized Beta Distributions. *IEEE Transactions on Reliability*, *R36*(4), 463—–466.

Mildenhall, S. J. (2005). Correlation and Aggregate Loss Distributions With An Emphasis On The Iman-Conover Method. *Casualty Actuarial Society Forum*, *Winter*. https://www.casact.org/sites/default/files/database/forum/_06wforum/_06w107.pdf

Mildenhall, S. J., & Major, J. A. (2022). *Pricing Insurance Risk: Theory and Practice*. John Wiley & Sons, Inc. https://doi.org/10.1002/9781119756538

Nadarajah, S., Chu, J., & Jiang, X. (2016). On moment based density approximations for aggregate losses. *Journal of Computational and Applied Mathematics*. https://doi.org/10.1016/j.cam.2015.11.048

Parodi, P. (2015). *Pricing in General Insurance*. CRC Press. file:///C:/Users/youhe/Downloads/kdoc/_o/_00042/_01.pdf

Parodi, P., Thrumble, D., Watson, P., Ji, Z., Wang, A., Bhatia, I., et al. (2024). Loss modelling from first principles. *British Actuarial Journal*, *6*, 1–39. https://doi.org/10.1017/s1357321724000163

Parodi, P., & Watson, P. (2019). Property Graphs - a Statistical Model for Fire and Explosion Losses Based on Graph Theory. *ASTIN Bulletin*, *49*(2), 263–297. https://doi.org/10.1017/asb.2019.4

Philips, T. K., & Nelson, R. (1995). The Moment Bound is Tighter than Chernoff's Bound for Positive Tail Probabilities. *The American Statistician*. https://doi.org/10.1080/00031305.1995.10476137

Pittarello, G., Luini, E., & Marchione, M. M. (2024). GEMAct: a Python package for non-life (re) insurance modeling. *Annals of Actuarial Science*, 1–37. https://doi.org/10.1017/S1748499524000022

Provost, S. B. (2005). Moment-based density approximants. *Mathematica Journal*, *9*(4), 727–756.

Robertson, J. P. (1992). The computation of aggregate loss distributions. *Proceedings of the Casualty Actuarial Society*, *79*(150), 57–133.

Schaller, P., & Temnov, G. (2008). Efficient and Precise Computation of Convolutions: Applying FFTs to Heavy Tailed Distributions. *Computational Methods in Applied Mathematics*, *8*(2), 187–200.

Venter, G. G. (1983). Transformed beta and gamma distributions and aggregate losses. *Proceedings of the Casualty Actuarial Society*, *70*(133), 289–308.

Venter, G. G. (1991). Premium Calculation Implications of Reinsurance Without Arbitrage. *ASTIN Bulletin*, *21*(02), 223–230. https://doi.org/10.2143/ast.21.2.2005365

Vernic, R. (1999). Recursive Evaluation of Some Bivariate Compound Distributions. *ASTIN Bulletin*, *29*(2), 315–325. https://doi.org/10.2143/ast.29.2.504618

Wang, S. S. (1995). Insurance pricing and increased limits ratemaking by proportional hazards transforms. *Insurance: Mathematics and Economics*, *17*(1), 43–54. https://doi.org/10.1016/0167-6687(95)00010-P

Wang, S. S. (1998). Aggregation of correlated risk portfolios: models and algorithms. *Proceedings of the Casualty Actuarial society*, 848–939. http://www.casact.com/pubs/proceed/proceed98/98048.pdf

Wang, S. S. (2000). A class of distorion operators for pricing financial and insurance risks. *Journal of Risk and Insurance*, *67*(1), 15–36.

Wang, S. S. (2002). A Universal Framework for Pricing Financial and Insurance Risks. *ASTIN Bulletin*, *32*(2), 213–234. https://doi.org/10.2143/AST.32.2.1027

Wilson, H., & Keich, U. (2016). Accurate pairwise convolutions of non-negative vectors via FFT. *Computational Statistics and Data Analysis*, *101*, 300–315. https://doi.org/10.1016/j.csda.2016.03.010

Wilson, H., & Keich, U. (2017). Accurate small tail probabilities of sums of iid lattice-valued random variables via FFT. *Journal of Computational and Graphical Statistics*, *26*(1), 223–229.