Pittarello, Luini and Marchione's GEMAct package

Reproducing Pittarello, Luini and Marchione (2024)

Stephen J. Mildenhall

2026-10-22 19:56 commit b5ca906+

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.

2 Paper solution and gold standard

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 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 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

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 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 1: Pittarello, Luini and Marchione (2024), Table 1.

Section 3.7, Figure 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 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.

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 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 3: Pittarello, Luini and Marchione (2024), Table 3.

3 The `aggregate` reproduction

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 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 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 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.

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 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 3 is reproduced.

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 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 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 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 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 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 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.

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 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 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 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}.

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 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 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 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 $(2.1 \times 10^9)^3$.

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 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 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 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.

4 Comments

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 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.

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.

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.

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).

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

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

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

Parodi, P. (2015). *Pricing in General Insurance*. CRC Press. file:///C:/Users/youhe/Downloads/kdoc/_o/_00042/_01.pdf

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