Using `nttagg`

Exact compound distributions with the nttagg kind and the number-theoretic transform

Stephen J. Mildenhall

2026-10-30

When the claim count N and the severity X take finitely many nonnegative integer values with rational probabilities, every probability of $S = X_1 + \cdots + X_N$ is rational. Write $\Pr(N = k) = a_k/d_N$ and $\Pr(X = j) = b_j/d_X$, with n the largest count. Then $\mathrm{P}z^S = \sum_k (a_k/d_N)\left(\sum_j b_j z^j/d_X\right)^k$ is a polynomial and $D = d_N d_X^n$ clears its denominators, so $\Pr(S = s) = H_s/D$ with integer H_s. The number-theoretic transform (NTT) is the FFT over $\mathbb{Z}/p\mathbb{Z}$ for primes p with $2^r \mid p-1$, where 2^r is the transform length. It computes each $H_s \bmod p$ exactly, and the Chinese remainder theorem recovers H_s once the product of the primes exceeds D.

The float FFT computes the same polynomial in double precision. Its error is roughly constant in absolute terms, near 10^{-17}, so the relative error in a probability of size 10^{-k} is of order 10^{k-17}: about fifteen figures in the body of the distribution and none below 10^{-17}. A comparison with the exact answer shows where the float answer loses accuracy.

The DecL kind `nttagg`[1] builds an `aggregate.ntt.AggregateNTT`, an ordinary `Aggregate` on the integer lattice, with bucket size 1, that also stores the exact answer. The program has a `dfreq` clause followed by a `dsev` clause, with no exposure, layers, or reinsurance. Outcomes are nonnegative integers, and probabilities are exact as written: `0.3` is 3/10, not the double nearest it. A probability may be an arithmetic expression, such as `(1-1/1000)`, evaluated exactly, and an omitted probability list gives equal exact weights.

```
nttagg Dice dfreq [1:6] dsev [1:6]
nttagg Book dfreq [0 1 2 3] [0.4 0.3 0.2 0.1] dsev [1 2 5] [1/2 1/3 1/6]
nttagg Cat dfreq [0:8] dsev [1 10] [(1-1/1000) 1/1000]
```

```
from fractions import Fraction
from aggregate import build, qd, __version__
from aggregate.ntt import engine_status
from aggregate.charts import build_chart_doc
from aggregate.plots import plot_chartdoc
from style import wprint, GT, refuse, figures
figures()
print(__version__)
```

```
1.43.1
```

[1] From aggregate 1.1.0

Exact inputs and exact probabilities

The `Book` example has 0 to 3 claims, each of size 1, 2, or 5. Its `pprogram` shows the program with the probabilities reduced to fractions, and `qd` prints the usual summary.

```
a = build('nttagg Book dfreq [0 1 2 3] [0.4 0.3 0.2 0.1] '
          'dsev [1 2 5] [1/2 1/3 1/6]')
print(a.pprogram)
qd(a)
```

```
nttagg Book
  dfreq [0 1 2 3] [2/5 3/10 1/5 1/10]
  dsev [1 2 5] [1/2 1/3 1/6]
Aggregate object: Book. Frequency distribution empirical. Severity
dhistogram, [1, 5], bounded; atoms [1 2 5]. Updated with bucket size 1
and log2 = 4. Validation: not unreasonable.

      Mean     SD      CV   Skew P01 Median P99
X
Freq     1      1       1    0.6
Sev      2 1.4142 0.70711 1.4142   1      1   5
Agg      2 2.4495  1.2247 1.4153   0      1  10
```

The attribute `p_exact` is a Series of reduced `Fractions` indexed by loss, here over the whole support 0 to 15. Losses of 13 and 14 are impossible, and the largest loss, 15, needs three claims of 5, with probability $(1/10)(1/6)^3 = 1/2160$.

```
wprint(' '.join(str(p) for p in a.p_exact))
```

```
2/5 3/20 3/20 19/240 17/360 1/15 1/27 5/144 1/60 1/180 1/180 1/240 1/360 0 0
1/2160
```

Any quantity computed from the exact probabilities in rational arithmetic is also exact. For `Book` the total probability, the mean, $\Pr(S \le 2)$, and the stop-loss premium $\mathsf{P}(S-5)^+$ are 1, 2, 7/10, and 23/90.

```
pe = {int(x): p for x, p in a.p_exact.items()}
print(sum(pe.values()), sum(x * p for x, p in pe.items()),
      sum(p for x, p in pe.items() if x <= 2),
      sum(max(x - 5, 0) * p for x, p in pe.items()))
```

```
1 2 7/10 23/90
```

The exact moments are a column of `stats_df`, computed in rational arithmetic and rounded to float at the end. Validation compares the FFT moments, in the `empirical` column, with the exact moments. For `Book` the two agree to the six figures shown.

```
GT(a.stats_df.loc['agg', ['exact', 'empirical']], float_format='.6g')
```

measure	exact	empirical
ex1	2	2
ex2	10	10
ex3	64.8	64.8
mean	2	2
cv	1.22474	1.22474
skew	1.41526	1.41526

Exact arithmetic in a probability list

A probability in an `nttagg` list may be an arithmetic expression, and it is evaluated exactly. In the `Cat` example the expression `(1-1/1000)` is exactly 999/1000: a claim is 1 with that probability and 10 otherwise, and there are 0 to 8 claims, each count with probability 1/9. A loss of $10k$ needs exactly k claims, all large, so $\Pr(S = 10k) = 1/(9 \cdot 1000^k)$. A loss of 9 is impossible, and a loss of 8 needs eight small claims.

```
c = build('nttagg Cat dfreq [0:8] dsev [1 10] [(1-1/1000) 1/1000]')
print(c.pprogram)
print(c.p_exact_at([9, 10, 20, 80]))
print(c.p_exact_at(8), c.p_exact_at(8) == Fraction(999, 1000) ** 8 / 9)
print(all(c.p_exact_at(10 * k) == Fraction(1, 9 * 1000 ** k)
          for k in range(1, 9)))
```

```
nttagg Cat
  dfreq [0 1 2 3 4 5 6 7 8]
  dsev [1 10] [999/1000 1/1000]
loss
9.0                                0
10.0                            1/9000
20.0                         1/9000000
80.0    1/900000000000000000000000
Name: p_exact, dtype: object
1102253271188826697768 89/100000000000000000000000 True
True
```

Accuracy of the float FFT in the tail

For the `Cat` example, the frame `accuracy_df` lists the exact probability, correctly rounded to a float, beside the raw FFT answer, with the relative error computed exactly. The FFT error is never more than about 2×10^{-17}, so its relative error grows as the probability falls, reaching about 9% at 10^{-16}. At smaller probabilities the FFT answer is noise, and it is negative at a loss of 60.

```
df = c.accuracy_df.loc[10:80:10, ['p_exact_float', 'p_fft', 'rel_err']]
GT(df.rename(index=int), float_format='.3e')
```

loss	p_exact_float	p_fft	rel_err
10	1.111e-04	1.111e-04	-3.98e-14
20	1.111e-07	1.111e-07	7.77e-11
30	1.111e-10	1.111e-10	3.22e-08
40	1.111e-13	1.111e-13	-2.89e-05
50	1.111e-16	1.209e-16	8.77e-02
60	1.111e-19	-1.298e-18	-1.27e+01
70	1.111e-22	1.735e-18	1.56e+04
80	1.111e-25	1.605e-17	1.44e+08

The attribute `ntt_audit` summarizes the comparison. The noise floor is the largest absolute FFT value on a bucket whose exact probability is zero, and the maximum relative error is taken over the buckets whose exact probability is above it. The unresolved buckets have a positive exact probability at or below the floor.

```
for k in ['noise_floor', 'max_rel_err', 'rel_err_buckets',
          'unresolved_buckets']:
    print(f'{k:20s} {c.ntt_audit[k]:.4g}')
```

```
noise_floor          2.286e-17
max_rel_err          0.08771
rel_err_buckets      39
unresolved_buckets   6
```

Probabilities below the smallest double

The `Deep` example has the severity of `Cat`, 1 or 10, and a claim count equally likely to be any of 0 to 120. Its largest loss has probability $\Pr(S = 1200) = 1/(121 \cdot 10^{360})$, far below the smallest double, about 5×10^{-324}. The column `p_exact_float` underflows to zero at losses of 1,190 and 1,200, while `p_exact_log10`, computed from the integers, stays finite. The float answer `p_total` is zero at all four losses.

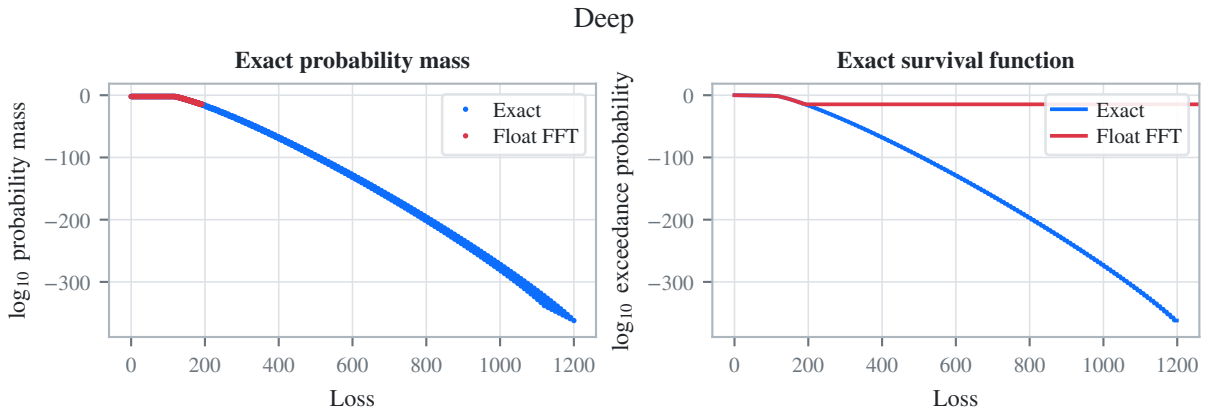
```
d = build('nttagg Deep dfreq [0:120] dsev [1 10] [(1-1/1000) 1/1000]')
print(d.p_exact_at(1200) == Fraction(1, 121 * 10 ** 360))
GT(d.density_df.loc[[300, 600, 1190, 1200],
                    ['p_total', 'p_exact_float', 'p_exact_log10']],
   float_format='.4g')
```

```
True
```

loss	p_total	p_exact_float	p_exact_log10
300	0	2.205e-40	-39.66
600	0	1.031e-131	-131
1,190	0	0	-359.1
1,200	0	0	-362.1

The `exact_tail` chart shows the exact mass and survival function on a base-10 logarithmic scale beside the float FFT answer. The float answer stops near 10^{-15}, and the exact answer continues to 10^{-362}.

```
fig = plot_chartdoc(build_chart_doc(d, 'exact_tail'))
```



The optional compiled package **`aggregate-ntt`**

The optional compiled package `aggregate-ntt` speeds up the exact calculation in large cases, and the library uses it when it is installed. The function `engine_status()` returns a dictionary showing whether it is used and, if not, why. Passing `ntt_engine='python'` to `build` forces the pure-Python calculation, and for `Deep` the two give identical exact probabilities.

```
wprint(engine_status())
dp = build('nttagg DeepPython dfreq [0:120] dsev [1 10] [(1-1/1000) 1/1000]',
           ntt_engine='python')
print(d.ntt_audit['engine'], dp.ntt_audit['engine'],
      dp.p_exact.equals(d.p_exact))
```

```
{'engine': 'rust', 'installed': True, 'engine_version': '1.0.0',
'engine_protocol': 2, 'library_protocol': 2, 'reason': None}
rust python True
```

Refused programs

An `nttagg` program is refused when its probabilities do not sum to exactly 1, when an outcome is not a nonnegative integer, or when a probability raises a number to a fractional power, whose result need not be rational. The support must also fit the grid, which `build` limits to 2^{16} buckets by default. An `nttagg` cannot be a unit of a portfolio.

```
refuse('nttagg X dfreq [0 1] [1/2 1/3] dsev [1]',
       'nttagg X dfreq [0 1] dsev [1.5 2]',
       'nttagg X dfreq [0 1] dsev [1 2] [(1/2)**(1/2) 1/2]',
       'nttagg X dfreq [1:100] dsev [1:100000]',
       'port P nttagg X dfreq [1] dsev [1]')
```

```
nttagg frequency: probabilities must sum to exactly 1; shortfall is 1/6
nttagg outcomes must be nonnegative integers, got '1.5'; clear denominators and
rescale the lattice yourself
nttagg exponent must be a whole number, got 1/2
X: the exact support 0..10000000 needs log2 >= 24, got 16. Pass build(...,
log2=24) or add hints{log2=24} to the declaration.
DecL parse error at line 1, column 8: Unexpected 'nttagg'. Did you mean: agg,
tags?
```