Using `stable` names and arithmetic

Named stable and tweedie laws, and arithmetic on stable references

Stephen J. Mildenhall

2026-10-30

A sum of independent stable laws with a common index α is a stable law, and so is a scaled or shifted stable law. Using `stable` describes the four parameters and the two parameterizations, and states the rules for the parameters of a sum, a multiple, and a shift. The class `aggregate.stable.Stable` computes these combinations in Python, and DecL reaches them through names. A `stable` statement with a name[1] stores the law in the recipe base, and the reference `stable.NAME` returns it. References take arithmetic[2], which is used to combine the stable limits of several units. A `tweedie` statement takes the same name and reference but no arithmetic, because a sum of independent Tweedie laws is a Tweedie law only when they have the same gamma severity.

```
stable x0 1.5 0                          # a name, then any form of the statement
stable x1 1.5 0.85
stable Book as "Pareto book" 1000 claims sev 100 * lomax 1.5
tweedie t0 1.5 100 2

3 * stable.x0                            # an anonymous law, not stored
stable.x0 + 2 * stable.x1 + 3
stable z stable.x0 + 2 * stable.x1 + 3   # the same combination, stored as stable.z
3 @ stable.x0                            # the sum of three independent copies
```

```
from aggregate import build, __version__
from style import wprint, refuse, figures
figures()
print(__version__)
```

```
1.43.1
```

Named stable and tweedie laws

A named law is a `stable` or `tweedie` statement with a name after the keyword, followed by an optional `as` label and any form of the statement, with optional `note`, `tags`, and `hints`. Printed, it starts with the name, and each form after the colon is valid DecL.

```
wprint(build('stable x0 1.5 0'))
wprint(build('stable x1 1.5 0.85'))
wprint(build('stable x2 1.5 -0.5'))
wprint(build('stable Book as "Pareto book" 1000 claims sev 100 * lomax 1.5'))
wprint(build('tweedie t0 1.5 100 2'))
```

[1] From aggregate 1.42.0
[2] From aggregate 1.43.0

```
x0: stable 1.5 0 1 0 = stable 1.5 0 1 0 s0, mean 0, Pr(X>mean) 0.5, Pr(X>x) ~
0.199471 x^-1.5, Pr(X<-x) ~ 0.199471 x^-1.5
x1: stable 1.5 0.85 1 0 = stable 1.5 0.85 1 -0.85 s0, mean 0, Pr(X>mean)
0.350502, Pr(X>x) ~ 0.369022 x^-1.5, Pr(X<-x) ~ 0.0299207 x^-1.5
x2: stable 1.5 -0.5 1 0 = stable 1.5 -0.5 1 0.5 s0, mean 0, Pr(X>mean) 0.598389,
Pr(X>x) ~ 0.0997356 x^-1.5, Pr(X<-x) ~ 0.299207 x^-1.5
Pareto book (Book): stable 1.5 1 18452.7 200000 = stable 1.5 1 18452.7 181547
s0, mean 200000, Pr(X>mean) 0.333333, Pr(X>x) ~ 1e+06 x^-1.5
t0: tweedie 1.5 100 2 = tweedie 1.5 100 cv 0.447214 = tweedie 10 claims sev
gamma 10 cv 1, Pr(X=0) 4.53999e-05
```

The recipe base stores numbers whichever form was written: the four S1 parameters for `stable`, and *p*, the mean, and the dispersion for `tweedie`. The program is stored with the entry, so the limit form of `Book` remains readable while its stored law is four numbers.

```
book = build.recipe('Book')
wprint(book.program)
print({k: float(book.spec[k]) for k in ('alpha', 'beta', 'scale', 'loc')})
```

```
stable Book as "Pareto book" 1000 claims sev 100 * lomax 1.5
{'alpha': 1.5, 'beta': 1.0, 'scale': 18452.701486440274, 'loc': 200000.0}
```

A later statement refers to a named law as `stable.x0` or `tweedie.t0`, the same dotted form as `agg.X` and `sev.X`. Either reference alone returns the stored law, with its name and label. The bare name does too when it is unique across kinds.

```
print(repr(build('stable.x0')))
print(repr(build('x0')))
print(build('tweedie.t0').dispersion)
```

```
Stable(alpha=1.5, beta=0, scale=1, loc=0, name='x0')
Stable(alpha=1.5, beta=0, scale=1, loc=0, name='x0')
2.0
```

Arithmetic on stable references

Stable references take `+` and `-` between laws, `+`, `-`, `*`, and `/` with a number, unary minus, and `@`. An expression on its own returns an anonymous `Stable` and stores nothing. On the numeric side anything a DecL expression accepts works, and a `+`, `-`, or `*` between numbers needs parentheses, as it does everywhere in DecL.

```
print(repr(build('3 * stable.x0')))
print(repr(build('stable.x1 + stable.x2')))
print(repr(build('stable.x0 + 2 * stable.x1 + 3')))
print(repr(build('(1 + 2) * stable.x0 / 6 - 1')))
```

```
Stable(alpha=1.5, beta=0, scale=3, loc=0)
Stable(alpha=1.5, beta=0.175, scale=1.5874010519681994, loc=0)
Stable(alpha=1.5, beta=0.6279767062808198, scale=2.4472608147714756, loc=3)
Stable(alpha=1.5, beta=0, scale=0.5, loc=-1)
```

A stored combination is written `stable z <expression>`. A reference resolves when the statement is parsed, so `z` holds four numbers and redefining `x0` afterward leaves `z` unchanged.

```
z = build('stable z stable.x0 + 2 * stable.x1 + 3')
wprint(z)
build('stable x0 1.5 0 9')
print(build('stable.z').scale == z.scale)
build('stable x0 1.5 0');
```

```
z: stable 1.5 0.627977 2.44726 3 = stable 1.5 0.627977 2.44726 1.46318 s0, mean
3, Pr(X>mean) 0.381008, Pr(X>x) ~ 1.24322 x^-1.5, Pr(X<-x) ~ 0.2841 x^-1.5
True
```

Independent copies of a stable law

The arithmetic operations act on laws, not on random variables, and each occurrence of a reference is an independent copy. As a result `stable.x0 + stable.x0` is different from `2 * stable.x0`. At $\alpha = 3/2$ the first has scale $2^{2/3} = 1.5874$ and the second has scale 2, because the scale of a sum of independent copies grows more slowly than the number of copies. Likewise `stable.x0 - stable.x0` is a symmetric stable law, not zero.

```
print(repr(build('stable.x0 + stable.x0')))
print(repr(build('2 * stable.x0')))
print(repr(build('stable.x0 - stable.x0')))
```

```
Stable(alpha=1.5, beta=0, scale=1.5874010519681994, loc=0)
Stable(alpha=1.5, beta=0, scale=2, loc=0)
Stable(alpha=1.5, beta=0, scale=1.5874010519681994, loc=0)
```

The form `t @ stable.x` is the sum of t independent copies, with scale $t^{1/\alpha}\gamma$ and location $t\delta$. Its characteristic function is φ^t, so t need not be an integer, and the result is the law at time t of the Lévy process with `stable.x` at time 1. The operator `@` is the same as in `2 @ agg.X`, which doubles a claim count. The last line below adds two laws at time 1/2, which returns `stable.x1`.

```
print(repr(build('2 @ stable.x1')))
print(repr(build('stable.x1 + stable.x1')))
print(repr(build('0.5 @ stable.x1 + 0.5 @ stable.x1')))
```

```
Stable(alpha=1.5, beta=0.85, scale=1.5874010519681994, loc=0)
Stable(alpha=1.5, beta=0.85, scale=1.5874010519681994, loc=0)
Stable(alpha=1.5, beta=0.85, scale=1, loc=0)
```

Limit laws of two units

Two units with Pareto severities and a common tail index combine through their stable limits. The sum has the sum of the means. Its right tail constant is the sum of the two values of $\lambda\theta^\alpha$, which gives a closed-form check. The difference has both tails, with the larger on the side of the unit with the larger $\lambda\theta^\alpha$.

```
build('stable A 1000 claims sev 100 * lomax 1.5')
build('stable B 400 claims sev 250 * lomax 1.5')
total = build('stable.A + stable.B')
wprint(total)
print(f'{total.tail_constant:,.2f} {1000 * 100 ** 1.5 + 400 * 250 ** 1.5:,.2f}')
wprint(build('stable.A - stable.B'))
```

```
stable 1.5 1 34721.7 400000 = stable 1.5 1 34721.7 365278 s0, mean 400000,
Pr(X>mean) 0.333333, Pr(X>x) ~ 2.58114e+06 x^-1.5
2,581,138.83 2,581,138.83
stable 1.5 -0.225148 34721.7 0 = stable 1.5 -0.225148 34721.7 7817.53 s0, mean
0, Pr(X>mean) 0.546994, Pr(X>x) ~ 1e+06 x^-1.5, Pr(X<-x) ~ 1.58114e+06 x^-1.5
```

Stable laws in the recipe library

The recipe library included with aggregate (see Using recipes) holds four stable laws under their classical names: `stable.Cauchy` ($\alpha = 1, \beta = 0$), `stable.Levy` ($\alpha = 1/2, \beta = 1$), `stable.Landau` ($\alpha = 1, \beta = 1$), and `stable.Holtsmark` ($\alpha = 3/2, \beta = 0$). They are available in any session and to a client that resolves library entries only, such as a spreadsheet, which has no access to a law declared in another cell.

```
for name in ('Cauchy', 'Levy', 'Landau', 'Holtsmark'):
    wprint(build(f'stable.{name}'))
print(repr(build('stable.Holtsmark + 2 * stable.Holtsmark')))
```

```
Cauchy: stable 1 0 1 0 = stable 1 0 1 0 s0, mean undefined, Pr(X>loc) 0.5,
Pr(X>x) ~ 0.31831 x^-1, Pr(X<-x) ~ 0.31831 x^-1
Levy: stable 0.5 1 1 0 = stable 0.5 1 1 1 s0, mean infinite, Pr(X>loc) 1,
Pr(X>x) ~ 0.797885 x^-0.5
Landau: stable 1 1 1 0 = stable 1 1 1 0 s0, mean infinite, Pr(X>x) ~ 0.63662
x^-1
Holtsmark: stable 1.5 0 1 0 = stable 1.5 0 1 0 s0, mean 0, Pr(X>mean) 0.5,
Pr(X>x) ~ 0.199471 x^-1.5, Pr(X<-x) ~ 0.199471 x^-1.5
Stable(alpha=1.5, beta=0, scale=2.4472608147714756, loc=0)
```

Refused programs

A law must be named to enter arithmetic, because `stable 1.5 -0.5` would otherwise have two readings, and DecL has no variables, so `3 * stable 1.5 0` and `x0 = stable 1.5 0` are parse errors. A name may contain a minus sign, so a subtraction needs a space before the minus, and the message for a missing name of that shape shows the form with spaces. The laws in a sum must have a common α. Scaling by zero, division by zero, and zero copies are refused. A `tweedie` reference takes no arithmetic, and a `chfagg` takes the parameters of a stable law but not a reference to one. A name held under two kinds must be written with its prefix.

```
build('agg x0 dfreq [1] dsev [1]')
refuse('3 * stable 1.5 0',
       'x0 = stable 1.5 0',
       'stable.x1-stable.x2',
       'stable.x0 + stable.Cauchy',
       '0 * stable.x1',
       'stable.x1 / 0',
       '0 @ stable.x1',
       '2 * tweedie.t0',
       'chfagg C stable.x1',
       'x0')
```

```
DecL parse error at line 1, column 5: Unexpected 'stable'.
DecL parse error at line 1, column 1: Unexpected 'x0'.
no recipe named 'x1-stable.x2' of kind 'stable'. A name may contain a minus
sign, so stable.x1-stable.x2 is one name. To subtract, write spaces: stable.x1 -
stable.x2
```

```
Stable: only laws with a common alpha add to a stable law, got 1.5 and 1.0
Stable: scaling by zero gives a point mass, not a stable law
Stable: division by zero does not give a stable law
Stable: the number of independent copies in t @ X must be positive and finite,
got 0.0
DecL parse error at line 1, column 5: Unexpected 'tweedie.t0'.
DecL parse error at line 1, column 10: Unexpected 'stable.x1'. Did you mean:
stable?
'x0' is ambiguous across kinds (agg, stable); write agg.x0 or stable.x0, or pass
kind= to choose. Shipped library names are unique across kinds.
```