Package 'cogmod'

September 12, 2026

Type Package

Title Cognitive Models for Subjective Scales and Decision Making Tasks

Version 0.3.0

Description Implements cognitive models for data from subjective (Likert or analog) scales and from decision making tasks with reaction times and choice data. Provides random generation, density functions, and custom response distributions for Bayesian estimation with 'brms', covering discreted-beta, ordered beta and choice-confidence models for subjective ratings, reaction-times families (Shifted Log-Normal, Shifted Wald), as well as sequential sampling models including the drift diffusion model (DDM), the racing diffusion model (RDM), the lognormal race model (LNR), and linear ballistic accumulator (LBA) model. The website provides examples and tutorials for using and interpreting the models. Methods are described in Ratcliff and McKoon (2008) <doi:10.1162/neco.2008.12-06-420>, Brown and Heathcote (2008) <doi:10.1016/j.cogpsych.2007.12.002>, Rouder et al. (2015) <doi:10.1007/s11336-013-9396-3>, Tillman et al. (2020) <doi:10.3758/s13423-020-01719-6>, Kubinec (2023) <doi:10.1017/pan.2022.20>, and Sciandra et al. (2024) <doi:10.1007/s10651-023-00592-5>.

URL https://github.com/DominiqueMakowski/cogmod, https://dominiquemakowski.github.io/cogmod/

BugReports https://github.com/DominiqueMakowski/cogmod/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports brms, insight, stats

Suggests testthat, cmdstanr, knitr, rmarkdown, loo, dplyr, ggplot2, ggrepel, easystats, datawizard, bayestestR, parameters, performance, modelbased, report, reformulas, lme4, RWiener, rtdists

Additional_repositories https://mc-stan.org/r-packages/

Config/Needs/website quarto

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Dominique Makowski [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5375-9967>)

Maintainer Dominique Makowski <D.Makowski@sussex.ac.uk>

Repository CRAN

Date/Publication 2026-09-12 08:10:02 UTC

Contents

badlm *Simulated Data Where Linear Models Fail*

Description

A simulated repeated-measures experiment in which the two conditions have, **by construction, the same mean reaction time** while differing radically in every other respect. Condition `A` has a short non-decision time (50 ms) and a wide, heavily right-skewed distribution; condition `B` has a long non-decision time (450 ms) and a narrow, nearly symmetric one. Twenty participants each contribute 25 trials per condition.

Usage

```
badlm
```

Format

A data frame with 1,000 rows and 3 variables:

Participant Participant identifier (factor, `S01-S20`).

Condition Experimental condition, `"A"` or `"B"`.

RT Reaction time, in seconds.

Details

The dataset exists to demonstrate the limits of the summary-statistics approach: a linear model - including a correctly specified linear *mixed* model with a random intercept per participant - finds no effect of `Condition`, because a difference in shift, in spread and in tail weight is invisible to a comparison of means. It is used in the *RT models* vignette and in the `cogmod` paper.

Participants differ in their overall speed through an *additive* offset (SD = 30 ms) applied identically to both conditions, so that each participant's true condition effect is exactly zero and a random intercept is the correctly specified model for the between-participant variation. The latent offsets are not included in the data.

Source

Simulated; see `data-raw/badlm.R` for the generating code.

Examples

```
data(badlm)

# The two conditions have the same mean...
tapply(badlm$RT, badlm$Condition, mean)

# ...but nothing else in common
tapply(badlm$RT, badlm$Condition, sd)
```

```
# A mixed model finds nothing
if (requireNamespace("lme4", quietly = TRUE)) {
  summary(lme4::lmer(RT ~ Condition + (1 | Participant), data = badlm))
}
```

`cogmod_inits` *Starting values that keep the sampler out of the flat regions*

Description

Builds an `init` argument for `brms::brm()`, for the custom families whose default starting point is a bad one.

This is not a tuning knob to reach for when sampling looks bad. For `cogmod_gamma()` and `cogmod_weibull()` the usual default is actively harmful, and the failure is silent: the chain does not error, it simply never moves.

Usage

```
cogmod_inits(formula, data, jitter = 0.25, ...)
```

Arguments

`formula`	The model formula, as passed to `brms::brm()`. Must carry the family, i.e. be built with `brms::bf(..., family = cogmod_gamma())`.
`data`	The data, as passed to `brms::brm()`.
`jitter`	SD of the noise added on the unconstrained scale, so that chains start at different points. Set to `0` for identical starts.
`...`	Passed to `brms::make_stancode()` and `brms::make_standata()`, for arguments such as `data2`.

Details

Two regions of the parameter space have no gradient to escape on, and both are easy to start inside.

`ndt` **too large.** `brms` initialises on the *unconstrained* scale, so `init = 0` puts `ndt` at `exp(0) = 1` second. For sub-second reaction times that is above nearly every observation, so every response is attributed to the outlier component, the decision parameters drop out of the density entirely, and their gradient is exactly zero. The chain is stuck where it started.

Shape below 1. For `cogmod_gamma()` and `cogmod_weibull()` the density is unbounded at `ndt` whenever the shape is below 1, and `init = 0` on a `softplus` link starts the shape at `log(2) = 0.69` - inside that region.

A single scalar `init` cannot avoid both, because they pull in opposite directions: `ndt = exp(c)` wants `c` around `-1.6`, while `shape = softplus(c)` wants `c` above `1.9`. That is why `init = 0` fails and why no other single number fixes it - the parameters have to be set separately, which is what this function does.

Value

A function of one argument, suitable for `brms::brm(init = )`. Each call returns a named list of starting values, one for every parameter the generated Stan program declares.

How it works

The Stan parameter declarations are read off `brms::make_stancode()` for the model you are actually fitting, and their dimensions off `brms::make_standata()`, rather than reconstructed from the formula. That is what makes it robust to `0 + Intercept`, interactions, group-level terms and smooths: whatever `brms` decided to call things, and however large it decided to make them, that is what is matched.

Every declared parameter is given a value, not only the ones this function has an opinion about. That is deliberate: CmdStan prints `Init values were only set for a subset of parameters` and lists the rest whenever the list is partial, which is noise for a list that is partial on purpose. The parameters with no family-specific target - regression slopes, standardized group-level effects, group-level SDs, spline coefficients - get generic values that are at least as good as Stan's own `U(-2, 2)`: slopes and standardized effects start at zero, positive parameters just above their lower bound, bounded ones at the midpoint, and Cholesky factors at the identity.

The family-specific values go to the intercept of each distributional parameter, and to any dpar left out of the formula (which `brms` declares as a plain auxiliary parameter). Values are set on whichever scale the parameter lives on: the link scale for an intercept - using the links on the family, so `cogmod_gamma(link_mu = "log")` is honoured - and the natural scale for an auxiliary parameter. Under `0 + Intercept` there is no separate intercept parameter, so the coefficient named `Intercept` inside the `b` vector gets it instead.

Each chain gets a different draw. The noise is added on the *unconstrained* scale - additive for a free parameter, multiplicative for a positive one, on the logit scale for a doubly bounded one - so a jittered value can never land outside its own bounds, and the chains still start dispersed enough for `Rhat` to mean something.

`ndt` starts deliberately **small** (0.1 s, a third of its prior median). The two errors are not symmetric: too small merely means the shift has to grow, which the gradient will do, whereas too large removes the gradient altogether.

Supported families

The family is read off `formula`, so build it with `brms::bf(..., family = cogmod_gamma())`. Every family built on the direct `ndt + poutlier` parameterization is covered - `cogmod_lognormal()`, `cogmod_logstudent()`, `cogmod_loggamma()`, `cogmod_invgaussian()`, `cogmod_exwald()`, `cogmod_bisa()`, `cogmod_gamma()`, `cogmod_invgamma()`, `cogmod_weibull()`, `cogmod_invweibull()`, `cogmod_logweibull()`, `cogmod_lba1()` and, for the choice-and-RT models, `cogmod_lnr()`, `cogmod_rdm()`, `cogmod_lba2()` and `cogmod_ddm()` - plus `cogmod_exgaussian()`, whose three parameters are all on the RT scale behind a `softplus` link and so are equally badly served by starting at `log(2)`.

See Also

`cogmod_priors()`, `cogmod_stanvars()`

Examples

```
d <- data.frame(RT = rcogmod_gamma(50, ndt = 0.3, poutlier = 0.02))
f <- brms::bf(RT ~ 1, sigma ~ 1, ndt ~ 1, poutlier ~ 1, family = cogmod_gamma())
inits <- cogmod_inits(f, d)
inits(1)


# Fitting needs cmdstanr, which lives outside CRAN - see the package website.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  m <- brms::brm(f,
    data = d, prior = cogmod_priors(f, d),
    stanvars = cogmod_stanvars(f), init = cogmod_inits(f, d),
    backend = "cmdstanr", chains = 1, iter = 500, refresh = 0
  )
}
```

`cogmod_priors` *Priors that make a cogmod posterior proper*

Description

Fills in weakly informative priors for the parameters `brms` would otherwise leave flat, for the model you are actually fitting.

For `cogmod_lognormal()` this is not a convenience. `brms` assigns a flat, improper prior to the intercept of any custom-family parameter it does not recognise, which there means both `ndt` and `poutlier`. The likelihood has two directions in which it is exactly flat: `poutlier` toward 1, where every response is attributed to the outlier component and `mu`, `sigma` and `ndt` drop out of the density altogether; and `ndt` toward 0, where the model reduces to an unshifted LogNormal and the gradient with respect to `log(ndt)` vanishes. Flat prior plus infinite flat region is an improper posterior. The fit does not fail loudly - it returns intercepts around `1e14` with `Rhat` near 2 and an effective sample size of about 5.

The second direction has nothing to do with the mixture; it is inherent to putting a positive shift on a log link, which is why a prior on `poutlier` alone is not enough.

Usage

```
cogmod_priors(formula, data, ...)
```

Arguments

`formula`	The model formula, as passed to `brms::brm()`. Must carry the family, i.e. be built with `brms::bf(..., family = cogmod_lognormal())`.
`data`	The data, as passed to `brms::brm()`.
`...`	Passed to `brms::get_prior()` and `brms::validate_prior()`, for arguments such as `data2` or `knots`.

Details

The function starts from `brms::get_prior()` for the model in hand, edits the rows it knows how to improve, and returns the result of `brms::validate_prior()`. Three things follow.

Every row comes from the model `brms` is going to build, so a prior matching no parameter is impossible by construction: `0 + Intercept` formulas, interactions, group-level terms and smooths are all handled because none of them are guessed at.

The return value is a `brmsprior` object: the same class returned by `brms::get_prior()` and `brms::prior()`. It prints as a table, including the defaults that `brms` will use, but can be combined with `c()` like any other `brms` prior specification.

Passing it through `brms::validate_prior()` means a malformed specification errors here, with the offending row in view, rather than deep inside `brm()`.

Value

A `brmsprior` object, to pass to `brms::brm(prior = )`.

Setting your own priors

Combine it with `brms::prior()` entries. Set `replace = TRUE` to replace one of these defaults, or omit it to add a prior for a different slot:

```
priors <- c(
  cogmod_priors(f, df),
  brms::prior(normal(-2, 0.1), class = "Intercept", dpar = "ndt"),
  replace = TRUE
)
```

Supported families

The family is read off `formula`, so build it with `brms::bf(..., family = cogmod_lognormal())`. Every family built on the direct `ndt` + `poutlier` parameterization is edited: `cogmod_lognormal()`, `cogmod_logstudent()`, `cogmod_loggamma()`, `cogmod_invgaussian()`, `cogmod_exwald()`, `cogmod_bisa()`, `cogmod_gamma()`, `cogmod_invgamma()`, `cogmod_weibull()`, `cogmod_invweibull()`, `cogmod_logweibull()` and, for the choice-and-RT models, `cogmod_lnr()`, `cogmod_rdm()`, `cogmod_lba2()` and `cogmod_ddm()`. `cogmod_exgaussian()` is edited too, although it is not built on that parameterization - see its own section below. Any other family, or a formula carrying none, is passed through: you get a message and `brms`'s own defaults, unchanged, so the call is always safe to leave in a script.

What gets set, on the link scale (`log` for `ndt`, `logit` for `poutlier`, `identity` for `shape`):

class	ndt	poutlier	shape
`Intercept`, or `b` on a coefficient named `Intercept`	`normal(-1.2, 0.2)`	`normal(-5, 1)`	`normal(0, 0.5)`
`b` (slopes)	`normal(0, 0.2)`	`normal(0, 0.2)`	`normal(0, 0.2)`
`sd`, `sds`	`exponential(1)`	`exponential(1)`	`exponential(1)`

`normal(-1.2, 0.2)` puts `ndt` at roughly 170 to 300 ms. Like everything else in these families it is stated in **seconds**, which is the unit the package expects throughout. `poutlier` is a proportion

and does not move: `normal(-5, 1)` is centred at about 0.7% and puts roughly 95% of its mass between 0.1% and 5%. That is where the empirical estimates sit - pooling across four lexical-decision megastudies, Miller (2024) puts the outlier proportion below 0.5% and argues that the 5-10% assumed in most simulation work is unrealistically large.

`shape` exists only for `cogmod_loggamma()`. `normal(0, 0.5)` is centred on the LogNormal shape and keeps the sampler clear of `sigma * shape >= 1`, where the decision density becomes unbounded at the shift and the likelihood with it.

A family may add rows of its own where its likelihood has a flat direction that `brms` would leave improper. Seven do:

- `cogmod_logstudent()`: `dof`. The LogNormal is only reached as `dof -> Inf`, and it is approached slowly - at `dof = 100` the density still differs from the LogNormal by 66% in the tail - so above about `dof = 30` the likelihood has almost stopped moving, and a `log` link puts that region at plus infinity. The prior fences the other end too: a Student-t is symmetric on the log scale, so a small `dof` piles mass just above `ndt` as well as in the slow tail, which is what `poutlier` is for. `normal(1.8, 0.7)` centres `dof` at 6 with 95% between 1.5 and 24.
- `cogmod_exwald()`: `tau`, the mean of the exponential residual stage. It is a length of time in seconds behind a `softplus` link, which `brms` has no way to know, and it shares a ridge with `ndt` - both delay the response, and only the shape of the leading edge tells them apart. `normal(-1.5, 0.7)` is the same statement `cogmod_exgaussian()` gets for the same quantity: roughly 55 to 630 ms.
- `cogmod_lba1()`: `sigmabias` and `boundary`. As the start-point range approaches zero the LBA converges to the recinormal and the likelihood stops depending on it, and a `softplus` link reaches zero only at minus infinity. Without those rows `sigmabias` runs off - `softplus(-10.4)`, `Rhat` 1.69.
- `cogmod_rdm()`: `sigmabias` and `boundary`, for exactly the reason `cogmod_lba1()` needs them - the two enter the threshold only through the sum `b = boundary + sigmabias` and trade off along a ridge worth a handful of log units, and the `softplus` link reaches zero only at minus infinity. The drift rates are left alone: a zero-drift accumulator still finishes, and still wins sometimes, so there is no plateau at the bottom of that direction the way there is for `cogmod_lnr()`'s `nuone`.
- `cogmod_lba2()`: `sigmazero`, `sigmaone`, `sigmabias` and `boundary`. The last two share the threshold ridge above; the two drift SDs are worse. The evidence scale of an LBA is arbitrary - multiply the drifts, their SDs, the start-point range and the threshold by any `c > 0` and every finishing time is unchanged - so the likelihood is *exactly* constant along that ray. These priors make the posterior proper; only fixing one SD in the formula (`sigmazero = 1`) identifies the scale. See `cogmod_lba2()`.
- `cogmod_ddm()`: `sigmadrift`, `sigmabias` and `sigmandt`, the three between-trial variability parameters. Each has a floor at zero that its link reaches only at minus infinity, and the likelihood stops changing well before then. They are also the parameters a DDM is least able to recover, so these priors are deliberately tighter than the rest. Fixing the ones a design cannot identify (`sigmadrift = 0` in `bf()`) is usually better than estimating them behind a prior.
- `cogmod_lnr()`: `nuone`, `sigmazero` and `sigmaone`. Push an accumulator's rate far enough down and it never finishes first, so the density depends on it only through a survival term that has already saturated at 1; past about `nuone = -6` the log-likelihood is exactly constant, and that accumulator's `sigma` is unidentified along with it. `mu` - which is `nuzero` - has the

mirror-image plateau, but it is the response's own intercept and brms already gives it a proper student_t default, so it is left alone. Model a rarely-chosen option and it is worth mirroring the nuone prior onto it.

All of them are the same failure as ndt and poutlier: an infinite flat region under a flat prior. See cogmod_lba1(), cogmod_rdm(), cogmod_lba2(), cogmod_ddm() and cogmod_lnr().

Note that no prior is set on the shape of cogmod_weibull() or cogmod_gamma(), although a shape below 2 makes their ndt gradient unbounded. That region is reached because the *likelihood* prefers it, by around 100 log units on the data in vignette("rt_models"), so a prior weak enough to be a sensible default cannot move the posterior out of it - only bias it. ?rcogmod_weibull sets out what to do instead.

The ex-Gaussian

cogmod_exgaussian() has neither ndt nor poutlier, but all three of its parameters are lengths of time in **seconds**, and brms has no way to know that. sigma and tau sit behind a softplus link; mu is on identity. All three intercepts are set.

class	sigma	tau
Intercept, or b on a coefficient named Intercept	normal(-2.3, 0.7)	normal(-1.5, 0.7)
b (slopes)	normal(0, 0.5)	normal(0, 0.5)
sd, sds	exponential(1)	exponential(1)

On the softplus scale normal(-2.3, 0.7) puts sigma - the SD of the Gaussian component - between roughly 25 and 330 ms with a median of 96 ms, and normal(-1.5, 0.7) puts tau - the mean of the exponential tail - between roughly 55 and 630 ms with a median of 201 ms. Both cover the range these parameters occupy across the usual simple- and choice-RT tasks and are wide enough not to fight data that disagrees.

tau arrives from brms flat, so it is filled like any other unrecognised custom dpar. sigma arrives with a **non-empty** default, student_t(3, 0, 2.5), because brms recognises the name from its own families - centred on softplus(0) = 0.69 s, a Gaussian SD wider than most whole RT distributions. That one is overridden rather than filled, the same treatment shape and an omitted ndt get above.

mu gets normal(0.4, 0.25) on its own intercept - 95% of the mass between -0.09 and 0.89 s. It used to be left to brms, whose student_t(3, 0, 2.5) is a fair statement about a location on a softplus link (median 0.69 s) but not on the identity link mu now uses, where it is centred on zero seconds and puts a Gaussian centre of -2 s on a par with one of +2 s. The prior does **not** exclude negative values: mu is a location, and for fast heavily-tailed data the Gaussian component genuinely belongs near or below zero with tau carrying the mass. Only the intercept is set - the response's slopes are the effects being estimated, and are left to brms. Note that mu is the centre of the Gaussian component **alone**, so the mean of the distribution it implies is mu + tau; see cogmod_exgaussian().

Parameters left out of the formula

Writing ndt ~ 1 and omitting ndt entirely are not the same thing to brms, and the difference matters here. A dpar that appears in bf() - even as ~ 1 - gets a linear predictor, so it is estimated on the **link** scale and reported under *Regression Coefficients* as ndt_Intercept. A dpar left out is declared as

a plain auxiliary parameter instead, the same mechanism as `sigma` for `gaussian()`, estimated on the **natural** scale with no link and reported under *Further Distributional Parameters*.

Both forms are filled, with priors on whichever scale the parameter actually lives on:

dpar	in `bf()` (link scale)	omitted (natural scale)
`ndt`	`normal(-1.2, 0.2)`	`lognormal(-1.2, 0.2)`
`poutlier`	`normal(-5, 1)`	`exponential(100)`
`shape`	`normal(0, 0.5)`	`normal(0, 0.5)`
`sigmabias`, `boundary` (`cogmod_lba1()`, `cogmod_rdm()`, `cogmod_lba2()`)	`normal(0, 1)`	`lognormal(-0.7, 0.75)`
`sigmazero`, `sigmaone` (`cogmod_lba2()`)	`normal(0, 1)`	`lognormal(-0.7, 0.75)`
`sigmadrift` (`cogmod_ddm()`)	`normal(0, 1)`	`lognormal(-1, 0.75)`
`sigmabias` (`cogmod_ddm()`)	`normal(-2, 1)`	`beta(1, 5)`
`sigmandt` (`cogmod_ddm()`)	`normal(-3, 1)`	`lognormal(-3, 1)`
`nuone` (`cogmod_lnr()`)	`normal(0.7, 1.5)`	`normal(0.7, 1.5)`
`sigmazero`, `sigmaone` (`cogmod_lnr()`)	`normal(0, 1)`	`lognormal(-0.7, 0.75)`
`dof` (`cogmod_logstudent()`)	`normal(1.8, 0.7)`	`lognormal(1.8, 0.7)`
`tau` (`cogmod_exwald()`)	`normal(-1.5, 0.7)`	`lognormal(-1.5, 0.7)`
`mu` (`cogmod_exgaussian()`)	`normal(0.4, 0.25)`	- (always modelled)
`sigma` (`cogmod_exgaussian()`)	`normal(-2.3, 0.7)`	`lognormal(-2.3, 0.7)`
`tau` (`cogmod_exgaussian()`)	`normal(-1.5, 0.7)`	`lognormal(-1.5, 0.7)`

The `ndt` pair describes the same belief twice: `lognormal` is just `normal` on the log scale, written for the untransformed parameter. The `cogmod_exgaussian()` pairs do the same to within a rounding error, because `softplus(x)` and `exp(x)` agree to three figures for the `x` below `-2` that both parameters live at: `lognormal(-2.3, 0.7)` has median 0.100 against `softplus(-2.3)` = `0.096`.

If the data were trimmed before fitting, tighten this rather than removing it: `normal(-7, 0.5)` asserts essentially no contamination while keeping the density positive below `ndt`. Fixing `poutlier` = `0` outright reinstates the hard min-RT boundary that the component exists to remove. See the *Trimmed data* section of `cogmod_lognormal()`.

`poutlier` is deliberately *not* the same belief twice. Leaving it out of the formula is itself information - you either trimmed the data already or do not expect outliers - so the omitted form puts its **mode at zero**, which a logit-scale prior cannot do at any location. The centre is unchanged: `exponential(100)` has median 0.0069 against `plogis(-5)` = `0.0067`. It is still a prior rather than a constraint, so a genuine spike of fast responses will still pull the rate up; to switch the parameter off entirely, trim the data and it will simply sit near zero.

Two rows override a **non-empty** `brms` default rather than filling an empty one. `brms` recognises the name `ndt` from its own shifted families and supplies `uniform(0, min_Y)` - precisely the min-RT bound that `cogmod_lognormal()`'s parameterization exists to remove, reimposed silently and with a warning about an upper bound on an unbounded parameter. It recognises `sigma` too, and gives `cogmod_exgaussian()`'s a half `student_t(3, 0, 2.5)`, whose median of 1.9 s is a Gaussian SD wider than most whole RT distributions. An omitted `poutlier` is left flat over `[0, 1]` by `brms`, which is proper but puts half its mass above 0.5, and an omitted `shape` or `tau` is flat over the whole real line, which is not proper at all.

Slope and group-level priors are deliberately narrow. On a log or a logit link a flat slope prior is not as harmless as it looks, and a group-level SD with no prior can wander far enough for individual groups to reach the flat regions above even when the population intercept is well behaved.

References

- Miller, J. (2024). Estimating the proportions and latencies of reaction time outliers: A pooling method and case study of lexical decision tasks. *Behavior Research Methods*, *56*(7), 7280-7306. doi:10.3758/s1342802402419y

See Also

`cogmod_inits()`, `cogmod_stanvars()`

Examples

```
d <- data.frame(RT = rcogmod_lognormal(50, ndt = 0.3, poutlier = 0.02))
f <- brms::bf(RT ~ 1, sigma ~ 1, ndt ~ 1, poutlier ~ 1,
  family = cogmod_lognormal()
)
cogmod_priors(f, d)

# Replace a default, or append a prior for another parameter.
priors <- c(
  cogmod_priors(f, d),
  brms::prior(normal(-2, 0.1), class = "Intercept", dpar = "ndt"),
  replace = TRUE
)
```

`cogmod_stanvars` *The Stan code a cogmod family needs, read off the model*

Description

Returns the `stanvars` argument for `brms::brm()`, for whichever cogmod family the model uses. It is a front end to the per-family `<family>_stanvars()` functions - `cogmod_lognormal_stanvars()`, `cogmod_choco_stanvars()`, `cogmod_ddm_stanvars()` and the rest - which remain available and unchanged.

Usage

```
cogmod_stanvars(formula, ...)
```

Arguments

`formula`	A `brms::bf()` formula carrying the family, a cogmod family object, or a fitted `brmsfit`.
`...`	Passed to the family's own `<family>_stanvars()` function.

Details

brms needs the custom likelihood injected into the generated Stan program, and every cogmod family ships one. Calling this instead of the family's own function means the family is named once, in bf(), rather than twice:

```
f <- brms::bf(RT ~ Condition, ndt ~ Condition,
              family = cogmod_lognormal())

brms::brm(f, data = df,
          prior    = cogmod_priors(f, df),
          init     = cogmod_inits(f, df),
          stanvars = cogmod_stanvars(f))
```

Value

A stanvars object, to pass to brms::brm(stanvars =).

A warning it may emit

cogmod_lba1() and cogmod_lba2() have a likelihood that is *exactly* constant along the ray that multiplies the drift rates, their SDs, the start-point range and the threshold offset by a common factor. If the formula pins none of them to a constant, this warns: the RT distribution is still identified, but the individual parameters are not, and the fit will converge to whatever the priors say about that direction rather than fail. Fixing any one member in bf() - conventionally sigmazero = 1 - silences it. Leaving a parameter *out* of bf() does not count: brms estimates it anyway.

What it accepts

A brms::bf() formula carrying the family, the family object itself, or a fitted brmsfit (useful for recompiling or for update()).

See Also

cogmod_priors(), cogmod_inits()

Examples

```
f <- brms::bf(RT ~ 1, ndt ~ 1, family = cogmod_lognormal())
cogmod_stanvars(f)

# Equivalent to naming the family a second time:
cogmod_lognormal_stanvars()
```

p_outlier *Per-trial outlier probabilities*

Description

Posterior probability that each response was generated by the outlier component rather than by the decision process, for a model fitted with any family built on the outlier mixture - `cogmod_lognormal()`, `cogmod_loggamma()`, `cogmod_lnr()` and the rest listed in the *Supported families* section of `cogmod_priors()`. This is the mixture *responsibility* `poutlier * g(rt) / (poutlier * g(rt) + (1 - poutlier) * f(rt - ndt))`, averaged over posterior draws.

Responses faster than `ndt` come out at 1, responses in the heart of the distribution near 0, and responses in either tail somewhere in between - the model discriminates by evidence rather than by a cutoff. A response in the middle can still be an outlier; a low probability means the data cannot tell, not that the trial is clean.

Averaging the responsibility over draws gives `P(trial i came from the outlier component | data)` directly, so the posterior mean *is* the quantity of interest and there is no interval to report alongside it. Pass `summary = FALSE` for the raw draws if you need the spread.

Usage

```
p_outlier(object, summary = TRUE)
```

Arguments

`object`	A `brmsfit` fitted with `cogmod_lognormal()`, `cogmod_loggamma()` or any other family built on the outlier mixture - see the *Supported families* section of `cogmod_priors()`.
`summary`	Logical; if `TRUE` (default) returns a data frame with one row per observation. If `FALSE`, returns the full draws x observations matrix.

Value

A data frame with columns `rt` and `p_outlier`, in the order the observations appear in the model frame, or a draws x observations matrix if `summary = FALSE`.

Examples

```
# Fitting needs cmdstanr, which lives outside CRAN - see the package website.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  df <- data.frame(RT = rcogmod_lognormal(200, ndt = 0.3, poutlier = 0.05))
  f <- brms::bf(RT ~ 1, ndt ~ 1, poutlier ~ 1, family = cogmod_lognormal())
  m <- brms::brm(f,
    data = df, stanvars = cogmod_stanvars(f),
    prior = cogmod_priors(f, df), init = cogmod_inits(f, df),
    backend = "cmdstanr", chains = 1, iter = 500, refresh = 0
  )
  head(p_outlier(m))
```

```
}
```

`rcogmod_betadiscrete` *Discrete Beta Model*

Description

The Discrete Beta (DBT) distribution models ordinal rating data on a fixed integer scale $R \in \{1, \ldots, k\}$ by discretizing an underlying continuous Beta distribution at $k-1$ evenly-spaced thresholds $\gamma_j = j/k$. Unlike proportional-odds style models, which fix the underlying distribution and estimate the thresholds, the Discrete Beta fixes the thresholds and estimates the two shape parameters of the underlying Beta distribution instead. This keeps the model parsimonious (only 2 parameters) while remaining flexible enough to reproduce "U" and "J" (non-monotonic convex) shapes that are common in rating data and that proportional-odds models cannot capture (Sciandra et al., 2024).

Usage

```
rcogmod_betadiscrete(n, mu = 0.5, phi = 3, k = 5, pzero = 0)

dcogmod_betadiscrete(x, mu = 0.5, phi = 3, k = 5, pzero = 0, log = FALSE)

pcogmod_betadiscrete(
  q,
  mu = 0.5,
  phi = 3,
  k = 5,
  pzero = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

qcogmod_betadiscrete(
  p,
  mu = 0.5,
  phi = 3,
  k = 5,
  pzero = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

cogmod_betadiscrete_lpmf_expose()

cogmod_betadiscrete_stanvars()
```

```
cogmod_betadiscrete(link_mu = "logit", link_phi = "log", link_pzero = "logit")

log_lik_cogmod_betadiscrete(i, prep)

posterior_predict_cogmod_betadiscrete(i, prep, ...)

posterior_epred_cogmod_betadiscrete(prep)
```

Arguments

n	Number of simulated values.
mu	Mean of the underlying Beta distribution (`0 < mu < 1`).
phi	Precision parameter of the underlying Beta distribution (must be strictly positive). Can be conceptualized as an "agreement" indicator: higher `phi` means less dispersion (more agreement) among ratings, holding `mu` fixed. Note: In many implementations, `phi` is parametrized differently, and correspond to the double of our `phi` argument (cogmod's `phi` = standard's `phi` * 2). Our parametrization Makes it `phi = 1` corresponds to uniform when `mu = 0.5`, which makes setting priors more convenient (e.g., on the logit scale)
k	Number of rating categories (a positive integer, `k >= 1`), i.e. the response scale runs from 1 to `k`.
pzero	Probability of an additional "hurdle" point mass at 0, on top of the `1:k` rating scale. Defaults to `0`, in which case the distribution reduces to the pure Discrete Beta model. Useful for rating scales that include an extra "zero" category (e.g., "not applicable" or a genuine zero response) that is not part of the underlying `1:k` continuum.
x, q	Vector of quantiles (integer ratings between 1 and `k`, or 0 if `pzero > 0`).
log, log.p	Logical; if `TRUE`, probabilities/densities are returned on the log scale.
lower.tail	Logical; if `TRUE` (default), probabilities are $P(R \leq q)$, otherwise $P(R > q)$.
p	Vector of probabilities.
link_mu, link_phi, link_pzero	Link functions for the parameters. `pzero` defaults to a `"logit"` link. By default (i.e., if `pzero` is not included in the `brms::bf()` formula), it is estimated as a single, intercept-only value shared across all observations (as is done for `pmid` in `cogmod_choco()`); it can instead be given predictors to let it vary (`pzero ~ x`), or fixed to a constant – e.g., `pzero = 0`, recovering the pure Discrete Beta model – directly in `brms::bf()` (as is done for `pmid` in `cogmod_choco()`).
i, prep	For brms' functions to run: index of the observation and a `brms` preparation object.
...	Additional arguments.

Details

Writing $\alpha = \mu\phi$ and $\beta = (1 - \mu)\phi$ for the shape parameters of the underlying Beta distribution, the probability mass function is (Sciandra et al., 2024, eq. 2)

$$P(R = j) = F_B(j/k; \alpha, \beta) - F_B((j-1)/k; \alpha, \beta), \quad j = 1, \ldots, k$$

where F_B is the Beta CDF.

`rcogmod_betadiscrete()` uses the equivalent, faster generative representation: draw a continuous $X \sim Beta(\alpha, \beta)$ and set $R = \lceil kX \rceil$, clipped to `[1, k]`.

When `pzero > 0`, a hurdle is added at 0: with probability `pzero` the response is 0, and with probability `1 - pzero` it is generated from the Discrete Beta distribution described above, i.e.

$$P(R = 0) = \texttt{pzero}, \quad P(R = j) = (1 - \texttt{pzero}) \times [F_B(j/k) - F_B((j-1)/k)], \quad j = 1, \ldots, k$$

Special cases:

- `mu = 0.5`, `phi = 1` (i.e. `alpha = beta = 1`): reduces to the discrete Uniform distribution on `1:k`.
- `alpha, beta < 1`: "U"/"J"-shaped, with mass concentrated in the tails.
- `alpha, beta > 1`: concave, with mass concentrated around the middle category.
- `phi -> Inf` (with `mu` fixed): mass concentrates on a single category.
- `pzero = 0`: reduces to the pure Discrete Beta model (no hurdle).

Note that `y = 0` is always handled by `pzero` alone, and `k` always refers to the number of categories of the *non-zero* `1:k` part of the scale. What *does* require some care is deciding what `k` should be and whether to estimate or fix `pzero`, depending on how the zero in your data arose:

- Scale is `0:N` and 0 is *not* a hurdle (just the lowest of `N + 1` ordinary ordinal categories, e.g., a 0-10 rating scale with no excess of zeros): recode the data to `1:(N + 1)` (add 1 to every response), use `vint(N + 1)`, and fix `pzero = 0` as shown below.
- Scale is `0:N` and 0 *is* a hurdle (e.g., a mix of a genuine/excess "zero" response with an ordinal `1:N` scale): keep the data as-is, use `vint(N)` (i.e., the total number of categories *minus* the hurdle category), and let `pzero` be estimated (optionally with predictors, `pzero ~ x`).
- Scale is `1:N` with excess responses piling up at the low end (e.g., a floor effect at the lowest category): recode by subtracting 1 (`1:N` becomes `0:(N - 1)`), then proceed as in the previous bullet, i.e., `vint(N - 1)` and estimate `pzero`.

Value

`dcogmod_betadiscrete()` returns the probability mass; `pcogmod_betadiscrete()` returns the cumulative probability; `qcogmod_betadiscrete()` returns the quantile (an integer between 0 and `k`); `rcogmod_betadiscrete()` returns simulated ratings. All are numeric vectors, vectorized over `x`/`q`/`p`, `mu`, `phi`, `pzero` and `k`. `cogmod_betadiscrete()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_betadiscrete_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_betadiscrete_lpmf_expose()` compiles that Stan code and returns it as an R function, for checking the mass function outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_betadiscrete()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, `posterior_predict_cogmod_betadiscrete()` a draws x 1 matrix of ratings simulated for observation `i`, and `posterior_epred_cogmod_betadiscrete()` a draws x observations matrix of expected ratings.

References

- Sciandra, M., Fasola, S., Albano, A., Di Maria, C., & Plaia, A. (2024). Discrete Beta and Shifted Beta-Binomial models for rating and ranking data. Environmental and Ecological Statistics, 31, 317-338. doi:10.1007/s10651023005925

Examples

```
x <- 1:10
probs <- dcogmod_betadiscrete(x, mu = 0.66, phi = 3.51, k = 10)
barplot(probs, names.arg = x)

y <- rcogmod_betadiscrete(1000, mu = 0.66, phi = 3.51, k = 10)
hist(y, breaks = 0:10)

# discrete Uniform special case
dcogmod_betadiscrete(1:5, mu = 0.5, phi = 1, k = 5)

# hurdle at zero: 20% chance of a 0, otherwise pure Discrete Beta
dcogmod_betadiscrete(0:5, mu = 0.66, phi = 3.51, k = 5, pzero = 0.2)


# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpmf <- cogmod_betadiscrete_lpmf_expose()
  lpmf(y = 7, mu = 0.66, phi = 3.51, pzero = 0, k = 10)
}


# Fitting with brms. Because `k` is fixed data rather than a distributional
# parameter, it is passed through the brms::vint() addition term. Put the
# family on the formula, and cogmod_stanvars() supplies the Stan code for it.
f <- brms::bf(rating | vint(k) ~ predictor, family = cogmod_betadiscrete())
cogmod_stanvars(f)

# To also model the hurdle probability (e.g., proportion of zero ratings):
brms::bf(rating | vint(k) ~ predictor, pzero ~ predictor,
  family = cogmod_betadiscrete()
)

# To fix pzero at exactly 0, e.g. because your scale has no hurdle:
brms::bf(rating | vint(k) ~ predictor, pzero = 0,
  family = cogmod_betadiscrete()
)
```

`rcogmod_betagate` *Beta-Gate Model*

Description

The Beta-Gate model represents subjective ratings as a mixture of a continuous Beta distribution with additional point masses at the extremes (0 and 1). This structure effectively captures common patterns in subjective rating data where respondents often select extreme values at higher rates than would be expected from a Beta distribution alone.

The Beta-Gate model corresponds to a reparametrized ordered beta model (Kubinec, 2023, doi:10.1017/pan.2022.20). In the ordered Beta model, the extreme values (0 and 1) arise from censoring an underlying latent process based on cutpoints ("gates"). Values falling past the gates are considered extremes (zeros and ones). The difference from the Ordered Beta is the way the cutpoints are defined, as well as the scale of the precision parameter phi.

It differs from the Zero-One-Inflated Beta (ZOIB) model in that the ZOIB model has `zoi` and `coi` parameters, directly controlling the likelihood of extreme values. Instead, Beta-Gate uses `pex` and `bex` to define "cutpoints" after which extreme values become likely. In an ordered beta framework, the boundary probabilities arise through a single underlying ordering process (the location of the cutpoints on the latent scale). In a ZOIB framework, the boundaries are more like additional mass points inserted into a beta distribution. In Beta-gate models, extreme values arise naturally from thresholding a single latent process.

Usage

```
rcogmod_betagate(n, mu = 0.5, phi = 3, pex = 0.1, bex = 0.5)

dcogmod_betagate(x, mu = 0.5, phi = 3, pex = 0.1, bex = 0.5, log = FALSE)

cogmod_betagate_lpdf_expose()

cogmod_betagate_stanvars()

cogmod_betagate(
  link_mu = "logit",
  link_phi = "softplus",
  link_pex = "logit",
  link_bex = "logit"
)

log_lik_cogmod_betagate(i, prep)

posterior_predict_cogmod_betagate(i, prep, ...)

posterior_epred_cogmod_betagate(prep)
```

Arguments

`n`	Number of simulated values.
`mu`	Mean of the underlying Beta distribution (`0 < mu < 1`).
`phi`	Precision parameter of the underlying Beta distribution (must be strictly positive). Can be conceptualized as an "agreement" indicator: higher `phi` means less dispersion (more agreement) among ratings, holding `mu` fixed. Note: In many implementations, `phi` is parametrized differently, and correspond to the double of our `phi` argument (cogmod's `phi` = standard's `phi * 2`). Our parametrization Makes it `phi = 1` corresponds to uniform when `mu = 0.5`, which makes setting priors more convenient (e.g., on the logit scale)

pex	Controls the location of the lower and upper boundary gates (0 <= pex <= 1). It defines the total probability mass allocated to the extremes (0 or 1). Higher pex increases the probability of extreme values (0 or 1).
bex	Balances the extreme probability mass pex between 0 and 1 (0 <= bex <= 1). A balance of 0.5 means that the 'gates' are symmetrically placed around the center of the distribution, and values higher or lower than 0.5 will shift the relative "ease" of crossing the gates towards 1 or 0, respectively.
x	Vector of quantiles (values at which to evaluate the density). Must be between 0 and 1, inclusive.
log	Logical; if TRUE, returns the log-density.
link_mu, link_phi, link_pex, link_bex	Link functions for the parameters.
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

Special cases:

- When pex = 0: Pure Beta distribution with mean mu and precision phi * 2.
- When pex = 1: Pure Bernoulli distribution with P(1) = bex, P(0) = 1-bex.
- When bex = 0 and pex = 1: All mass at 0.
- When bex = 1 and pex = 1: All mass at 1.

Psychological Interpretation:

- mu: Can be interpreted as the underlying average tendency or preference strength, disregarding extreme "all-or-nothing" responses.
- phi: Reflects the certainty or consistency of the non-extreme responses. Higher phi indicates responses tightly clustered around mu (more certainty), while lower phi (especially phi = 1) suggests more uniform or uncertain responses.
- pex: Represents the overall tendency towards extreme responding (choosing 0 or 1). This could reflect individual response styles (e.g., acquiescence, yea-saying/nay-saying) or properties of the item itself (e.g., polarizing questions).
- bex: Indicates the *direction* of the extreme response bias. bex > 0.5 suggests a bias for producing ones more easily, while bex < 0.5 suggests a bias towards zero.

Value

rcogmod_betagate() returns a numeric vector of n simulated ratings on the unit interval [0, 1], including the exact 0s and 1s produced by the gates. dcogmod_betagate() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument; at 0 and 1 it is the probability mass rather than a density. cogmod_betagate() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_betagate_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(),

and cogmod_betagate_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_betagate() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, posterior_predict_cogmod_betagate() a draws x 1 matrix of ratings simulated for observation i, and posterior_epred_cogmod_betagate() a draws x observations matrix of expected ratings.

References

- Kubinec, R. (2023). Ordered beta regression: a parsimonious, well-fitting model for continuous data with lower and upper bounds. Political Analysis, 31(4), 519-536.

Examples

```
# Symmetric gates (c0=0.05, c1=0.95), pex=0.1, bex=0.5
x1 <- rcogmod_betagate(10000, mu = 0.5, phi = 3, pex = 0.1, bex = 0.5)
hist(x1, breaks=50, main="rcogmod_betagate: Symmetric Cutpoints (pex=0.1)")

# Asymmetric gates (c0=0.15, c1=0.95), pex=0.2, bex=0.25
x2 <- rcogmod_betagate(10000, mu = 0.5, phi = 3, pex = 0.2, bex = 0.25)
hist(x2, breaks=50, main="rcogmod_betagate: Asymmetric Cutpoints (pex=0.2, bex=0.25)")

# No gating (pure Beta)
x3 <- rcogmod_betagate(10000, mu = 0.7, phi = 5, pex = 0, bex = 0.5)
hist(x3, breaks=50, main="rcogmod_betagate: No Extreme Values (pex=0)")

x <- seq(0, 1, length.out = 1001)
densities <- dcogmod_betagate(x, mu = 0.5, phi = 5, pex = 0.2, bex = 0.5)
plot(x, densities, type = "l", main = "Density Function", xlab = "y", ylab = "Density")

# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpdf <- cogmod_betagate_lpdf_expose()
  lpdf(y = 0.5, mu = 0.6, phi = 10, pex = 0.2, bex = 0.5)
}
```

---

rcogmod_bisa *Shifted Birnbaum-Saunders (Fatigue Life) Model*

Description

Density, random generation, and brms custom family for the shifted Birnbaum-Saunders distribution, also known as the fatigue life distribution: a **first-passage-time model in which evidence arrives in discrete cycles and only ever towards the boundary**. The decision time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_bisa()`: Simulates random draws.
- `dcogmod_bisa()`: Computes the density (likelihood).
- `cogmod_bisa()`: Creates a `brms::custom_family()`.
- `cogmod_bisa_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_bisa(n, mu = 3, boundary = 0.5, ndt = 0.2, poutlier = 0)

dcogmod_bisa(x, mu = 3, boundary = 0.5, ndt = 0.2, poutlier = 0, log = FALSE)

cogmod_bisa(
  link_mu = "softplus",
  link_boundary = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_bisa_lpdf_expose()

cogmod_bisa_stanvars()

log_lik_cogmod_bisa(i, prep)

posterior_predict_cogmod_bisa(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_bisa(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mu`	Drift rate: the average size of the per-cycle evidence increment, whose SD is fixed at 1. Must be positive.
`boundary`	Decision threshold: the evidence needed to respond. Must be positive.
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
`poutlier`	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
`x`	Vector of quantiles (observed reaction times).
`log`	Logical; if TRUE, probabilities p are given as log(p).

link_mu, link_boundary, link_ndt, link_poutlier
: Link functions for the parameters.

predict_outliers
: Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().

i, prep
: For brms' functions to run: index of the observation and a brms preparation object.

...
: Additional arguments.

Details

The Birnbaum-Saunders distribution is the near neighbour of the Wald (cogmod_invgaussian()), and this family is deliberately parameterized so that the two can be compared directly: mu is a drift rate and boundary a decision threshold in both, meaning the same thing, so the *only* thing that differs between them is how the evidence arrives.

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds, what the outlier component is for, and why its scale is a constant rather than a dpar.

Value

rcogmod_bisa() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_bisa() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_bisa() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_bisa_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_bisa_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_bisa() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_bisa() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_bisa() returns a draws x observations matrix of expected reaction times.

Where the distribution comes from

A Wald time is the first crossing of a diffusion: evidence moves continuously and can move *either way* at any instant. Here it instead accumulates in discrete cycles, and every cycle pushes towards the boundary - what is random is the **size** of each increment, never its sign. If the increments have average size mu and SD 1, then after n cycles the accumulated evidence is Normal(n * mu, n) by the central limit theorem, so

```
P(T <= n) = P(evidence >= boundary) = Phi((mu * n - boundary) / sqrt(n))
```

and treating the cycle count n as continuous turns that into a first-crossing time. It is one-directional accumulation in discrete chunks, with a Gaussian approximation standing in for the exact hitting-time calculation - which is exactly the fatigue-crack process Birnbaum and Saunders (1969) derived it for.

Fixing the per-cycle SD at 1 is not an arbitrary choice: it is the same convention that fixes the Wald's diffusion coefficient, and it is what keeps mu and boundary on a common scale across the two families. It also means there is no third parameter and there cannot be one - the shape is pinned by mu * boundary, just as the Wald's shape is pinned at boundary^2.

The tidy consequence is that

```
(mu * t - boundary) / sqrt(t)
```

is **exactly** standard normal. In the distribution's usual (a, b) parameters that transform is written (1 / a) * (sqrt(t / b) - sqrt(b / t)), with scale b = boundary / mu and shape a = 1 / sqrt(mu * boundary). The map between the two parameterizations is a bijection - boundary = sqrt(b) / a and mu = 1 / (a * sqrt(b)) - so nothing is given up by stating it mechanistically. Every quantity of the family is elementary as a result: the CDF is a normal CDF, the quantile function a closed form, and rcogmod_bisa() is one normal draw per observation with no rejection step.

Relation to the Wald

In these parameters the density is the Wald's own, tilted:

```
f_BS(t) = f_Wald(t; mu, boundary) * (mu * t + boundary) / (2 * boundary)
```

One sign is all that separates them - the exponent carries (mu * t - boundary), the prefactor (mu * t + boundary). The tilt factor is what makes the Birnbaum-Saunders an **equal mixture of an inverse Gaussian and a reciprocal inverse Gaussian**: half the mass is the Wald with the same mu and boundary, half is that Wald's length-biased version, which weights long crossings in proportion to their length.

So at the same (mu, boundary) this family is both slower and more spread out than the Wald. At mu = 3, boundary = 0.5 its mean is 0.222 s against the Wald's 0.167, and its SD 0.184 against 0.136. In general

```
E[T]   = boundary / mu   + 1 / (2 * mu^2)    # the Wald's mean, plus a term
Var[T] = boundary / mu^3 + 5 / (4 * mu^4)    # the Wald's variance, plus one
```

both always finite, so posterior_epred() always has a number to return - unlike cogmod_invgaussian() once its drift varies. The median is exactly boundary / mu, which is the Wald's *mean*: do not read the two families' parameters as describing the same central tendency.

The right tail decays like exp(-mu^2 * t / 2), the same exponential order as the Wald's and much lighter than a LogNormal's, and the density vanishes at the shift with all its derivatives. ndt is therefore as well behaved here as it is for the Wald, with no unbounded-likelihood boundary of the kind cogmod_gamma() and cogmod_weibull() have at shape < 1.

There is no sigmadrift. Across-trial drift variability is what cogmod_invgaussian() is for; here the extra dispersion comes from the mixture structure instead, at no cost in parameters.

References

- Birnbaum, Z. W., & Saunders, S. C. (1969). A new family of life distributions. *Journal of Applied Probability*, *6*(2), 319-327. doi:10.2307/3212003
- Desmond, A. F. (1986). On the relationship between two fatigue-life models. *IEEE Transactions on Reliability*, *R-35*(2), 167-169. doi:10.1109/TR.1986.4335400

Examples

```
rts <- rcogmod_bisa(1000, mu = 3, boundary = 0.5, ndt = 0.2, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# The mean is the Wald's, boundary / mu, plus 1 / (2 * mu^2).
mean(rcogmod_bisa(1e5, mu = 3, boundary = 0.5, ndt = 0.2))
0.2 + 0.5 / 3 + 1 / (2 * 3^2)

# The median is exactly ndt + boundary / mu.
median(rcogmod_bisa(1e5, mu = 3, boundary = 0.5, ndt = 0.2))
```

---

rcogmod_choco *Choice-Confidence (CHOCO) Model*

Description

Simulates data from the Choice-Confidence (CHOCO) model. This model is useful for subjective ratings (e.g., Likert-type scales) where responses represent a choice between two underlying categories (e.g., "disagree" vs. "agree") along with a degree of confidence or intensity.

The CHOCO model divides the response scale at a middle-value. Responses above and below the middle are modeled by two rescaled (and mirrored for the left side) Beta-Gate distributions. In Beta-Gate distributions, extreme values (0 or 1) are generated by the "lumping" of values that crossed a threshold (or "gate"). The location of these gates from the center of the distribution is controlled by the pex and bex parameters, influecing the ease of crossing the gate (and thus the probability of extreme values).

Usage

```
rcogmod_choco(
  n,
  p = 0.5,
  confright = 0.5,
  precright = 4,
  confleft = 0.5,
  precleft = 4,
  pex = 0.1,
  bex = 0.5,
  pmid = 0,
  mid = 0.5
```

```
)

dcogmod_choco(
  x,
  p = 0.5,
  confright = 0.5,
  precright = 4,
  confleft = 0.5,
  precleft = 4,
  pex = 0.1,
  bex = 0.5,
  pmid = 0,
  mid = 0.5,
  log = FALSE
)

cogmod_choco_lpdf_expose()

cogmod_choco_stanvars()

cogmod_choco(
  link_mu = "logit",
  link_confright = "logit",
  link_precright = "softplus",
  link_confleft = "logit",
  link_precleft = "softplus",
  link_pex = "logit",
  link_bex = "logit",
  link_pmid = "logit"
)

log_lik_cogmod_choco(i, prep)

posterior_predict_cogmod_choco(i, prep, ...)

posterior_epred_cogmod_choco(prep)
```

Arguments

`n`	Number of simulated trials.
`p`	Proportion parameter determining the balance between the left and right sides *after excluding* the probability mass at the middle (`pmid`). `P(Right Side | Not Middle) = p`.
`confright, confleft`	Mean parameter (`mu`) for the underlying Beta-Gate distribution for the *right* side and *left* side, respectively. Represents confidence towards 1. `0 < confright < 1`.
`precright, precleft`	Precision parameter (`phi`) for the underlying Beta-Gate distribution for the *right* side and *left* side, respectively. Must be positive. Higher values indicate more

concentrated distributions, and a value of 1 corresponds to a uniform distribution.

`pex` Controls the location of the lower and upper boundary gates (`0 <= pex <= 1`). It defines the total probability mass allocated to the extremes (0 or 1). Higher `pex` increases the probability of extreme values (0 or 1).

`bex` Balances the extreme probability mass `pex` between 0 and 1 (`0 <= bex <= 1`). A balance of `0.5` means that the 'gates' are symmetrically placed around the center of the distribution, and values higher or lower than `0.5` will shift the relative "ease" of crossing the gates towards 1 or 0, respectively.

`pmid` Probability mass exactly at the `mid`. This determines the proportion of trials where the output is directly assigned the value of `mid`, bypassing the left or right components.

`mid` The point dividing the scale (`0 < mid < 1`). Typically set to 0.5. Note that in the Stan implementation, `mid` is fixed at 0.5 and not available as a parameter.

`x` Vector of quantiles (values at which to evaluate the density). Must be between 0 and 1, inclusive.

`log` Logical; if TRUE, returns the log-density.

`link_mu`, `link_confright`, `link_precright`, `link_confleft`, `link_precleft`, `link_pex`, `link_bex`, `link_pmid`
Link functions for the parameters.

`i`, `prep` For brms' functions to run: index of the observation and a `brms` preparation object.

`...` Additional arguments.

Details

Psychological Interpretation:

- `p`: Represents the overall tendency to choose the "right" category (e.g., "agree") over the "left" category (e.g., "disagree"), given that a choice is made (i.e., not responding exactly at `mid`).
- `confright` and `confleft`: Average confidence level when choosing the "right" or "left" category. Higher values (closer to 1) indicate stronger confidence or agreement towards the extreme end of the scale.
- `precright` and `precleft`: Certainty or consistency of the confidence ratings for the right and left choices, respectively. Higher values indicate less variability in confidence ratings around their respective means (`confright`, `confleft`).
- `pex`: Represents the overall tendency towards extreme responding (choosing 0 or 1). This could reflect individual response styles (e.g., acquiescence, yea-saying/nay-saying) or properties of the item itself (e.g., polarizing questions).
- `bex`: Indicates the *direction* of the extreme response bias. `bex > 0.5` suggests a bias for producing ones more easily, while `bex < 0.5` suggests a bias towards zero.

Value

rcogmod_choco() returns a numeric vector of n simulated ratings on the unit interval [0, 1], including exact 0s and 1s from the extreme-response gates and exact mid values. dcogmod_choco() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument; at 0, 1 and mid it is the probability mass rather than a density. cogmod_choco() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_choco_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_choco_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_choco() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, posterior_predict_cogmod_choco() a draws x 1 matrix of ratings simulated for observation i, and posterior_epred_cogmod_choco() a draws x observations matrix of expected ratings.

References

- Kubinec, R. (2023). Ordered beta regression: a parsimonious, well-fitting model for continuous data with lower and upper bounds. Political Analysis, 31(4), 519-536. (Describes the underlying ordered beta model)

See Also

rcogmod_betagate

Examples

```
# Simulate data with different parameterizations
# 10% at mid, 50/50 split otherwise, symmetric confidence/precision
x1 <- rcogmod_choco(
  n = 5000, p = 0.5, confright = 0.5, precright = 4,
  confleft = 0.5, precleft = 4, pex = 0.1, bex = 0.5, pmid = 0, mid = 0.5
)
hist(x1, breaks = 50, main = "CHOCO: Symmetric Confidence/Precision", xlab = "y")

# No mid mass, 70% probability on right, higher confidence left (closer to 0)
x2 <- rcogmod_choco(
  n = 5000, p = 0.7, confright = 0.5, precright = 3,
  confleft = 0.8, precleft = 5, pex = 0.15, bex = 0.7, pmid = 0, mid = 0.5
)
hist(x2, breaks = 50, main = "CHOCO: Asymmetric p, Higher Conf Left", xlab = "y")

# Lower confidence overall (closer to mid), high probability in the middle
x3 <- rcogmod_choco(
  n = 5000, p = 0.5, confright = 0.2, precright = 3,
  confleft = 0.2, precleft = 3, pex = 0, bex = 0.5, pmid = 0.05, mid = 0.5
)
hist(x3, breaks = 50, main = "CHOCO: Low confidence overall", xlab = "y")
cogmod_choco()

# Example usage in a brms formula:
brms::bf(y ~ x1 + (1 | group),
```

```
  confright ~ x3,
  confleft ~ x3,
  precright ~ 1,
  precleft ~ 1,
  pex ~ age,
  bex ~ 1,
  pmid ~ 1,
  family = cogmod_choco()
)
```

`rcogmod_ddm` *Drift Diffusion Model (DDM)*

Description

The Drift Diffusion Model (DDM) describes a two-choice decision as noisy evidence accumulating between two boundaries until one of them is reached. The boundary reached is the choice and the time taken is the decision time. The observed RT is that decision time shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the diffusion.

Functions:

- `rcogmod_ddm()`: Simulates random draws from the DDM.
- `dcogmod_ddm()`: Computes the density (likelihood).
- `pcogmod_ddm()`: Computes the cumulative distribution function.
- `cogmod_ddm()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_ddm_stanvars()`: Generates the `stanvars` to pass to `brm()`.
- `p_outlier()`: Per-trial posterior probability of being an outlier.

`pcogmod_ddm()` is the cumulative distribution function: the probability that a response has been made by time `q`. With `response = NULL` it is the RT distribution marginally over the choice; with `response` set it is the *defective* CDF that boundary carries, rising to the probability of that boundary rather than to one.

Usage

```
rcogmod_ddm(
  n,
  drift = 0,
  boundary = 1,
  bias = 0.5,
  ndt = 0.2,
  sigmadrift = 0,
  sigmabias = 0,
  sigmandt = 0,
  poutlier = 0
```

```
)

dcogmod_ddm(
  x,
  drift = 0,
  boundary = 1,
  bias = 0.5,
  ndt = 0.2,
  response,
  sigmadrift = 0,
  sigmabias = 0,
  sigmandt = 0,
  poutlier = 0,
  log = FALSE
)

pcogmod_ddm(
  q,
  drift = 0,
  boundary = 1,
  bias = 0.5,
  ndt = 0.2,
  response = NULL,
  poutlier = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

cogmod_ddm(
  link_mu = "identity",
  link_boundary = "softplus",
  link_bias = "logit",
  link_sigmadrift = "softplus",
  link_sigmabias = "logit",
  link_sigmandt = "log",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_ddm_lpdf_expose()

cogmod_ddm_stanvars()

log_lik_cogmod_ddm(i, prep)

posterior_predict_cogmod_ddm(i, prep, predict_outliers = NULL, ...)
```

```
posterior_epred_cogmod_ddm(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of simulated trials. If `length(n) > 1`, the length is taken to be the number required.
`drift`	Drift rate. Any real value; positive pushes the accumulator towards the boundary coded `1`.
`boundary`	Boundary separation. Must be positive.
`bias`	Starting point, as a proportion of the boundary separation *measured from the boundary coded* `0`. Must be in `(0, 1)`.
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. With `sigmandt > 0` it is the **lower bound** of the between-trial distribution rather than its midpoint.
`sigmadrift`	Between-trial SD of the drift rate (`sv`). Must be non-negative. Default `0`.
`sigmabias`	Between-trial start-point range, as a fraction in `[0, 1)` of the widest range that keeps the start point inside the boundaries: `sw = sigmabias * min(2 * bias, 2 * (1 - bias))`. Default `0`.
`sigmandt`	Between-trial range of the non-decision time (`st0`), in the same unit as the data, with `ndt` its lower bound. Default `0`. Formerly `sigmatau`, which was a fraction of `minrt`.
`poutlier`	Proportion of responses generated by the outlier process rather than by the diffusion. Range: `[0, 1]`.
`x`	The observed reaction time (RT).
`response`	The boundary reached: `1` for the upper boundary, `0` for the lower one. This gives the *defective* density that boundary carries, mixed with the outlier component.
`log`	Logical; if TRUE, returns the log-density. Default: FALSE.
`q`	Vector of quantiles (reaction times, in seconds).
`lower.tail`	Logical; if TRUE (default) the probability is `P(RT <= q)`, otherwise `P(RT > q)`. With a `response`, both are defective • see Details.
`log.p`	Logical; if TRUE, probabilities are returned on the log scale.
`link_mu`, `link_boundary`, `link_bias`	Link functions for the drift rate, the boundary separation and the starting point. `mu` is the drift: `brms` requires the first distributional parameter of a custom family to be called `mu`.
`link_sigmadrift`, `link_sigmabias`, `link_sigmandt`	Link functions for the between-trial variability parameters. Fix them in the `brms::bf()` formula (e.g. `sigmadrift = 0`) to recover the classic 4-parameter DDM.
`link_ndt`, `link_poutlier`	Link functions for the non-decision time and the outlier rate.

predict_outliers
: Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the diffusion alone; the likelihood is always the full mixture either way. On the prediction methods themselves the default is NULL, which defers to the flag carried on the model - see with_outliers(). See Details.

i, prep
: For brms' functions to run: index of the observation and a brms preparation object.

...
: Additional arguments.

Value

rcogmod_ddm() returns a data frame with n rows and two columns:

rt
: The simulated reaction time.

response
: The boundary reached, 1 for upper and 0 for lower, matching the dec() coding used by the brms families.

dcogmod_ddm() returns the defective density at each element of x - the log density if log = TRUE - and pcogmod_ddm() the defective cumulative probability at each element of q, both for the response given in response and recycled to the length of the longest argument. cogmod_ddm() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_ddm_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_ddm_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_ddm() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, posterior_predict_cogmod_ddm() a draws x 2 matrix of reaction times and choices simulated for observation i, and posterior_epred_cogmod_ddm() a draws x observations matrix of expected reaction times (marginal over the two responses, and only approximate once the between-trial variability parameters are non-zero).

Response coding

The response coded 1 (response here, dec() in a brms formula) is the **upper** boundary and the response coded 0 is the **lower** one, following brms's own wiener() family. Two consequences are worth keeping in mind when reading a fitted model:

- bias is measured *from the lower boundary*, so bias > 0.5 places the starting point closer to the response coded 1, and bias < 0.5 closer to the response coded 0. Since first-passage times are shorter for the nearer boundary, bias > 0.5 makes responses coded 1 *faster* than responses coded 0, and bias < 0.5 makes them slower. At exactly bias = 0.5 the two conditional RT distributions are identical, whatever the drift rate.
- A positive drift pushes the accumulator towards the response coded 1. When 0 codes correct responses and 1 codes errors (a common choice when the task has no natural stimulus-to-boundary mapping), good performance therefore corresponds to a *negative* drift rate.

Parameterization

`ndt` is expressed **directly, in seconds** (through a log link in the `brms` family). Nothing about it is taken from the data: it is not bounded by the fastest observed response, so a non-decision time that varies by condition or by participant can exceed the sample minimum wherever the data support it.

This replaces the earlier `tau` / `minrt` pair, in which `ndt = tau * minrt` with `minrt` set to the fastest observed RT. That capped the non-decision time at an order statistic of the sample, so any condition or participant whose true `ndt` exceeded the fastest observed response was inexpressible, and the misfit surfaced as spurious effects on the other parameters.

`sigmatau` went with it, and is now `sigmandt`. It is the between-trial *range* of the non-decision time (`st0` in the usual notation), expressed directly in the same unit as the data rather than as a fraction of `minrt`, with `ndt` the lower bound of the resulting Uniform. The old name was a compound of a parameter that no longer exists and a scale factor the user no longer sees.

Between-trial variability

`sigmadrift`, `sigmabias` and `sigmandt` extend the classic 4-parameter process to the full 7-parameter one. Each is legitimately **zero**, and setting all three to zero in the formula recovers the classic model:

```
brms::bf(RT | dec(Error) ~ Condition, sigmadrift = 0, sigmabias = 0,
         sigmandt = 0, ndt ~ 1, poutlier ~ 1, family = cogmod_ddm())
```

Writing `sigmadrift = 0` **fixes** the parameter; leaving it out of `bf()` altogether *estimates* it, which is not the same thing. All three are hard to recover even from a lot of data, and each has a flat direction at its own floor - the link only reaches zero at minus infinity, and the likelihood stops changing well before then - so `cogmod_priors()` gives all three deliberately tight priors. Fixing the ones a design cannot identify is usually better than estimating them behind a prior.

The outlier component

A shifted distribution assigns exactly zero density to any response faster than `ndt`, which puts a hard boundary in the likelihood at the fastest observed RT. Mixing in a component with support over the whole positive line removes it: every response keeps positive density whatever `ndt` is, so the boundary becomes a finite cost rather than a wall and the log-density stays smooth and differentiable.

Because this model produces a **choice as well as a time**, the contaminant has to produce both. It is a guess: the choice is uniform over the two options, and the RT is a **half Normal** with scale `0.2` seconds.

$$f(t,k) = p\frac{1}{K}g(t) + (1-p)f_k(t - ndt)$$

The `1 / K` is what keeps the total summing to one over the response options; without it it would come to `1 + poutlier`.

`poutlier` is a *rate*, not a classification: the model never labels individual trials, it estimates what share of them came from elsewhere. Use `p_outlier()` for per-trial posterior probabilities.

Reaction times must be in seconds

The outlier component's scale is a constant in seconds, and so are the priors `cogmod_priors()` supplies. There is no argument for changing the unit: the `minrt` argument that used to rescale the component was removed in 0.2.1. Millisecond data fails silently rather than loudly - the outlier component contributes nothing and the min-RT boundary comes back. See the corresponding section of `cogmod_lognormal()` for the full account, which applies unchanged here.

Implementation

The 4-parameter density is `brms::dwiener()` (which requires the `RWiener` package). Draws are taken by inverting the first-passage CDF, which - unlike `brms::rwiener()`, and unlike `rtdists` - vectorises over parameter sets, so the per-call setup is paid once rather than once per posterior draw. That matters for `rstantools::posterior_predict()`, where every draw carries its own parameters; it is several times faster there and agrees with both packages' samplers to within sampling error.

The full 7-parameter model is built on top of these rather than delegated to another package: between-trial variability is simulated by drawing the per-trial parameters, and evaluated by combining a closed-form drift correction with Gauss-Legendre quadrature over the starting point and non-decision time.

In Stan the decision component is `wiener_lpdf()`, called with its own non-decision time set to zero because the shift is applied by the mixture around it, and short-circuited to the cheaper 4- and 5-parameter forms whenever the start-point and non-decision-time ranges both vanish. The cheapest of the three, Stan's classic 4-parameter density, is used only where it is sound: it returns `-inf` with **NaN** derivatives once the rescaled decision time `t / boundary^2` falls below about `6.6e-4`, or once the density underflows, and a NaN derivative on a trial the mixture gives no weight to still turns the gradient of the whole model to NaN. Those calls go to the `sv`-capable form instead, which the two agree with to `1e-13` where they meet.

Fitting

```
f <- brms::bf(RT | dec(Error) ~ Condition, boundary ~ Condition, bias ~ 1,
              sigmadrift = 0, sigmabias = 0, sigmandt = 0,
              ndt ~ 1, poutlier ~ 1, family = cogmod_ddm())
brms::brm(f, data = df,
          prior    = cogmod_priors(f, df),
          init     = cogmod_inits(f, df),
          stanvars = cogmod_stanvars(f))
```

Use `cogmod_inits()` rather than `init = 0`. `brms` initialises on the unconstrained scale, so `init = 0` puts `ndt` at `exp(0) = 1` second - above nearly every sub-second RT, which leaves every response attributed to the outlier component and the diffusion parameters with no gradient at all.

Predictions exclude the outlier component

`posterior_predict()` and `posterior_epred()` describe the **diffusion alone** by default, as if `poutlier` were zero, because the outlier component is a fixed regularizer rather than a claim about how guesses are distributed. Use `with_outliers()` for the fitted mixture - chiefly for `brms::pp_check()` - and `without_outliers()` to go back. `log_lik()` is always the full mixture.

Accuracy of pcogmod_ddm()

This CDF is more accurate than the one in rtdists, which is the usual reference implementation. Deviation from numerical integration of the density, at drift = -4, boundary = 0.6, bias = 0.25, by decision time:

decision time	this function	rtdists::pdiffusion()
0.02	+2.8e-16	+5.5e-04
0.05	-1.1e-16	+4.0e-04
0.30	-1.1e-16	+1.5e-04

Over a grid of 360 (drift, boundary, bias, boundary-reached, time) cells the worst deviation from integrating dcogmod_ddm() is 1e-15, i.e. rounding. The choice probabilities are exact to the same order, where rtdists::pdiffusion(Inf, ...) is out by up to 1.4e-04. The difference is not academic: this function exists because rcogmod_ddm() inverts it to draw from, so any error in it would land directly in the draws.

pcogmod_ddm() covers the classic **4-parameter** DDM plus the outlier component. The between-trial variability parameters would each need their own quadrature layer on top - and sigmadrift, which the density handles with a closed-form correction, has no such form here - so they are not arguments at all: passing one is an error rather than a silently wrong number. Integrate dcogmod_ddm() over q if you need them.

With a response, pcogmod_ddm() returns the **defective** CDF P(RT <= q, choice = response), which does not reach one: its limit is the probability of that boundary, given by pcogmod_ddm(Inf, response = k). The upper tail is then the matching defective survival P(RT > q, choice = response), so the two add to that response's own probability rather than to one. Marginally (response = NULL) they add to one as usual. pcogmod_rdm() follows the same convention.

References

- Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation*, *20*(4), 873-922. doi:10.1162/neco.2008.1206420

See Also

rcogmod_rdm(), rcogmod_lba2(), rcogmod_lnr()

Examples

```
# Simulate data, with 2% of trials from the outlier process
data <- rcogmod_ddm(1000,
  drift = 0.5, boundary = 1, bias = 0.5, ndt = 0.2, poutlier = 0.02
)
head(data)

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_ddm(0.1, ndt = 0.2, response = 1, poutlier = 0.02)
dcogmod_ddm(0.1, ndt = 0.2, response = 1, poutlier = 0)
```

```
# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpdf <- cogmod_ddm_lpdf_expose()
  lpdf(
    Y = 0.5, mu = 0.5, boundary = 1, bias = 0.5, sigmadrift = 0,
    sigmabias = 0, sigmandt = 0, ndt = 0.2, poutlier = 0.02, dec = 1
  )
}
```

`rcogmod_exgaussian` *Ex-Gaussian Model (Classical Parameterization)*

Description

Density, random generation, and `brms` custom family for the Ex-Gaussian distribution, using the "classical" parameterization familiar to experimental psychologists, in which `mu` and `sigma` are the mean and SD of the Gaussian component alone, and `tau` is the mean of the exponential component (the tail). This is unlike `brms`'s built-in `exgaussian()` family, in which `mu` indexes the mean of the *entire* distribution (Gaussian + exponential components combined).

Functions:

- `rcogmod_exgaussian()`: Simulates random draws from the Ex-Gaussian distribution.
- `dcogmod_exgaussian()`: Computes the density (likelihood) of the Ex-Gaussian distribution.
- `cogmod_exgaussian()`: Creates a `brms::custom_family()` for use in `brms` models.

Usage

```
rcogmod_exgaussian(n, mu = 0.5, sigma = 0.1, tau = 0.2)

dcogmod_exgaussian(x, mu = 0.5, sigma = 0.1, tau = 0.2, log = FALSE)

cogmod_exgaussian(
  link_mu = "identity",
  link_sigma = "softplus",
  link_tau = "softplus"
)

cogmod_exgaussian_lpdf_expose()

cogmod_exgaussian_stanvars()

log_lik_cogmod_exgaussian(i, prep)
```

```
posterior_predict_cogmod_exgaussian(i, prep, ...)

posterior_epred_cogmod_exgaussian(prep)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Mean of the Gaussian component. Unbounded - it is a location, not a scale - though for RT data it is normally positive. Range: (-Inf, Inf).
sigma	SD of the Gaussian component. Must be positive. Range: (0, Inf).
tau	Mean of the exponential component (the tail). Must be positive. Range: (0, Inf).
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_tau	Character of the type of link used to model the ex-Gaussian parameters. Defaults to "identity" for mu and "softplus" for sigma and tau (see Details).
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

The Ex-Gaussian distribution is the sum of an independent Normal (Gaussian) random variable with mean mu and SD sigma, and an Exponential random variable with mean tau (rate 1 / tau). Unlike brms's built-in exgaussian() family - in which mu indexes the mean of the *entire* distribution (Gaussian + exponential components combined) - here mu is the mean of the Gaussian component alone, so that the mean of the full distribution is mu + tau.

This distinction matters because changes in the Gaussian location (mu) and changes in the exponential tail (tau) can offset one another at the level of the overall mean, so effects estimated on brms's default mu can lead to different (and potentially incorrect) inferences than effects estimated on this classical mu.

In the brms custom family (cogmod_exgaussian()), sigma and tau use a "softplus" link (log(1 + exp(x))) rather than "log". Both are scales and must be strictly positive for the density to exist at all. A "log" link would enforce that too, but its curvature explodes as the linear predictor departs from zero, producing extreme gradients and making priors and sampling harder to calibrate - and tau is on the RT scale (seconds), where it can take comparatively large values. "softplus" is positive-constrained like "log" but behaves almost linearly (softplus(x) ~ x) away from zero, so weakly-informative priors can be stated directly on the RT scale.

mu is different, and uses "identity". It is the **location** of the Gaussian component, not a scale: the convolution is well defined for any real value, and the density integrates to one at mu = 0 or below just as it does above (the Stan lpdf has always accepted a non-positive mu - it checks only sigma and tau). Nothing is gained by constraining it, and two things are lost. First, interpretability, which is most of the reason to prefer the ex-Gaussian in the first place: behind a softplus link a coefficient is not in seconds, and the conversion factor moves with the intercept - the local slope is 0.33 at mu = 0.4 s, 0.39 at 0.5 s and 0.63 at 1 s, so the same effect reads as a different number

depending on where the intercept sits. On `"identity"` a coefficient is seconds, full stop. Second, fidelity: for fast, heavily-tailed data the Gaussian component genuinely belongs near or below zero with `tau` carrying the mass, and forcing `mu > 0` distorts the `mu`/`tau` split in exactly the cases where that decomposition is the thing being estimated. This also matches every other implementation - `brms`'s own `exgaussian()`, `retimes`, and the estimates reported in the literature - so fitted values are directly comparable.

The cost is that `brms`'s default intercept prior is no longer sensible for `mu` (`student_t(3, 0, 2.5)` centred at zero seconds), which is why `cogmod_priors()` now supplies one.

Value

`rcogmod_exgaussian()` returns a numeric vector of `n` simulated reaction times, in seconds. `dcogmod_exgaussian()` returns the density at each element of `x` - the log density if `log = TRUE` - recycled to the length of the longest argument. `cogmod_exgaussian()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_exgaussian_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_exgaussian_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_exgaussian()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, `posterior_predict_cogmod_exgaussian()` a draws x 1 matrix of reaction times simulated for observation `i`, and `posterior_epred_cogmod_exgaussian()` a draws x observations matrix of expected reaction times.

References

- Matzke, D., & Wagenmakers, E. J. (2009). Psychological interpretation of the ex-Gaussian and shifted Wald parameters: A diffusion model analysis. *Psychonomic Bulletin & Review*, *16*(5), 798-817. doi:10.3758/PBR.16.5.798

Examples

```
# Simulate 1000 RTs
rts <- rcogmod_exgaussian(1000, mu = 0.5, sigma = 0.1, tau = 0.2)
hist(rts, breaks = 50, main = "Simulated Ex-Gaussian RTs", xlab = "Reaction Time")
```

---

rcogmod_exwald *Shifted Ex-Wald Model*

Description

Density, random generation, and `brms` custom family for the shifted ex-Wald distribution of Schwarz (2001): a **Wald (diffusive) decision stage convolved with an exponential residual stage**. The decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_exwald()`: Simulates random draws.

- `dcogmod_exwald()`: Computes the density (likelihood).
- `cogmod_exwald()`: Creates a `brms::custom_family()`.
- `cogmod_exwald_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_exwald(n, mu = 3, boundary = 0.5, tau = 0.15, ndt = 0.2, poutlier = 0)

dcogmod_exwald(
  x,
  mu = 3,
  boundary = 0.5,
  tau = 0.15,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)

cogmod_exwald(
  link_mu = "softplus",
  link_boundary = "softplus",
  link_tau = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_exwald_lpdf_expose()

cogmod_exwald_stanvars()

log_lik_cogmod_exwald(i, prep)

posterior_predict_cogmod_exwald(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_exwald(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mu`	Drift rate: the average speed of evidence accumulation. Must be positive.
`boundary`	Decision threshold: the evidence needed to respond. Must be positive.
`tau`	Mean of the exponential residual stage, in seconds. Must be positive.
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).

poutlier — Proportion of responses generated by the outlier process rather than by the decision process. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LogNormal.

x — Vector of quantiles (observed reaction times).

log — Logical; if TRUE, probabilities p are given as log(p).

link_mu, link_boundary, link_tau, link_ndt, link_poutlier — Link functions for the parameters.

predict_outliers — Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().

i, prep — For brms' functions to run: index of the observation and a brms preparation object.

... — Additional arguments.

Details

The decision time is Wald(mu, boundary) + Exponential(1 / tau): evidence accumulates at rate mu until it reaches boundary, and an exponentially distributed residual stage of mean tau follows. It is the **mechanistic counterpart of** cogmod_exgaussian(), whose first stage is a descriptive Gaussian rather than a decision process, and tau means the same thing in both.

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds, what the outlier component is for, and why its scale is a constant rather than a dpar.

The mean exists and is ndt + boundary / mu + tau, so posterior_epred() returns a number - unlike cogmod_invgaussian() once its drift varies.

Value

rcogmod_exwald() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_exwald() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_exwald() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_exwald_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_exwald_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_exwald() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_exwald() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_exwald() returns a draws x observations matrix of expected reaction times.

ndt and tau

Both delay the response, and they are separated only by shape: ndt is a hard floor, tau a variable stage with an exponential spread. That makes them a ridge rather than a flat direction - the leading

edge of the distribution identifies `ndt` - but a ridge all the same. The `normal(-1.2, 0.2)` that `cogmod_priors()` puts on `ndt` is what holds it; widen it and expect the two to trade off. Schwarz's own model has no `ndt` at all, the exponential stage being the whole of the residual time, and fixing `ndt = 0` in `bf()` recovers exactly that.

There is deliberately no `sigmadrift`. It and `tau` both fatten the right tail and are very hard to tell apart; `cogmod_invgaussian()` is where the drift-variability route lives.

How the density is computed

Worth knowing, because the two branches are not equally common. Writing `g = 1 / tau`, the convolution has an elementary closed form, `g * exp(-g * t + boundary * (mu - k)) * F_Wald(t; k, boundary)` with `k = sqrt(mu^2 - 2 * g)`, whenever `mu^2 > 2 / tau`. At a drift of 3 and a threshold of 0.5 that asks for `tau > 0.22 s`, where a residual stage is more often nearer 0.1 s, so the other branch is reached routinely.

There `k` is imaginary, and the same expression continues analytically into `g * exp(-(boundary - mu * t)^2 / (2 * t)) * Re[w(z)]`, where `w` is the Faddeeva function and `z = (kappa * sqrt(t) + i * boundary / sqrt(t)) / sqrt(2)` with `kappa = sqrt(2 / tau - mu^2)`. The exponent is the Wald's own, so nothing overflows, and `w` is evaluated by Weideman's 24-term rational approximation. **Both branches are exact**, and they meet exactly at `mu^2 = 2 / tau` - at `kappa = 0` the second reduces to the first - so the relative step measured either side of the seam is 5e-8, which is rounding.

Quadrature on the original convolution integral was tried first and does not work: the log-integrand is bimodal, with one peak at the Wald bulk near zero and another at `u = t` where `exp(u / tau)` is climbing, and their widths vary independently over orders of magnitude. Fixed-panel rules reach only 1e-1 relative error somewhere in the region an RT fit actually visits, and leave a step of 0.74 at the seam.

References

- Schwarz, W. (2001). The ex-Wald distribution as a descriptive model of response times. *Behavior Research Methods, Instruments, & Computers*, *33*(4), 457-469. doi:10.3758/bf03195403

Examples

```
rts <- rcogmod_exwald(1000, mu = 3, boundary = 0.5, tau = 0.15,
                      ndt = 0.2, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# The mean is boundary / mu + tau, on top of ndt.
mean(rcogmod_exwald(1e5, mu = 3, boundary = 0.5, tau = 0.15, ndt = 0.2))
0.2 + 0.5 / 3 + 0.15
```

---

rcogmod_gamma *Shifted Gamma Model*

Description

Density, random generation, and brms custom family for the shifted Gamma distribution. A Gamma-distributed decision time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the decision process.

Functions:

- rcogmod_gamma(): Simulates random draws.
- dcogmod_gamma(): Computes the density (likelihood).
- cogmod_gamma(): Creates a brms::custom_family().
- cogmod_gamma_stanvars(): Generates the stanvars to pass to brm().

Usage

```
rcogmod_gamma(n, mu = 3, sigma = 0.15, ndt = 0.2, poutlier = 0)

dcogmod_gamma(x, mu = 3, sigma = 0.15, ndt = 0.2, poutlier = 0, log = FALSE)

cogmod_gamma(
  link_mu = "softplus",
  link_sigma = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_gamma_lpdf_expose()

cogmod_gamma_stanvars()

log_lik_cogmod_gamma(i, prep)

posterior_predict_cogmod_gamma(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_gamma(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Shape of the Gamma decision time. Must be positive.
sigma	Scale of the Gamma decision time. Must be positive.
ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LogNormal.

x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_ndt, link_poutlier	Link functions for the parameters.
predict_outliers	Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

mu is the **shape** and sigma the **scale** of the Gamma decision time, so the mean decision time is mu * sigma and the median reaction time is ndt plus the Gamma median.

The Gamma is not merely a convenient skewed shape: Tejo et al. (2019) derive it as a first-passage time for an accumulator whose **starting point varies across trials**, which places it beside cogmod_invgaussian() (diffusion from a fixed start) and cogmod_bisa() (one-directional discrete cycles). The drift rate is not identified from the fit though - it enters only the back-calculation of the implied starting-point distribution - so mu and sigma stay a shape and a scale here rather than becoming a drift and a boundary.

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a dpar, and why reaction times have to be in seconds.

Note that the Gamma density is **unbounded at** ndt whenever the shape mu < 1, which makes the likelihood unbounded as ndt approaches the fastest response. The outlier component cannot repair that, since it adds density rather than capping it. Less obviously, a shape anywhere **below 2** leaves the derivative of the log-likelihood with respect to ndt unbounded at every observation, which costs sampling time rather than correctness - see the shape section of ?rcogmod_weibull, where the same three regimes are set out and measured. cogmod_loggamma() nests this family at shape = sigma and lets the data choose the shape instead of fixing it.

Do **not** fit this with init = 0, for two reasons at once: it puts ndt at exp(0) = 1 second, above most sub-second responses, and the shape at softplus(0) = 0.69, inside the unbounded region above. No single scalar avoids both - ndt = exp(c) wants c near -1.6 while shape = softplus(c) wants c above 1.9. Use cogmod_inits(), which sets them separately:

```
brms::brm(f, data = df, prior = cogmod_priors(f, df),
          stanvars = cogmod_stanvars(f), init = cogmod_inits(f, df))
```

Measured on 1500 simulated trials with a true shape of 3, init = 0 left the shape stuck at 0.69 and ndt at 0.999, with Rhat 2.3 and an effective sample size of 3 after 306 seconds; an informative prior on the shape did *not* rescue it, because a prior cannot move a chain whose gradient is zero. The brms default init = "random" also works, at 14 seconds. See cogmod_inits() for the full account.

Value

rcogmod_gamma() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_gamma() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_gamma() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_gamma_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_gamma_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_gamma() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_gamma() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_gamma() returns a draws x observations matrix of expected reaction times.

References

- Tejo, M., Araya, H., Niklitschek-Soto, S., & Marmolejo-Ramos, F. (2019). Theoretical models of reaction times arising from simple-choice tasks. *Cognitive Neurodynamics*, *13*(4), 409-416. doi:10.1007/s11571019095321

Examples

```
rts <- rcogmod_gamma(1000, mu = 3, sigma = 0.15, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density
dcogmod_gamma(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_gamma(0.1, ndt = 0.3, poutlier = 0)
```

---

rcogmod_geg *Generalised Ex-Gaussian (GEG) Distribution*

Description

The Generalised Ex-Gaussian of Marmolejo-Ramos et al. (2023): the ex-Gaussian with one extra shape parameter, obtained by raising its CDF to a power.

$$F_{GEG}(x) = [F_{EG}(x)]^{shape}$$

so that the density is

$$f_{GEG}(x) = shape \cdot [F_{EG}(x)]^{shape-1} \, f_{EG}(x)$$

At shape = 1 this is the ex-Gaussian exactly - not approximately - so cogmod_exgaussian() is nested inside it and loo_compare() between the two is like-for-like.

Usage

```
rcogmod_geg(n, mu = 0.4, sigma = 0.1, tau = 0.2, shape = 1)

dcogmod_geg(x, mu = 0.4, sigma = 0.1, tau = 0.2, shape = 1, log = FALSE)

pcogmod_geg(
  q,
  mu = 0.4,
  sigma = 0.1,
  tau = 0.2,
  shape = 1,
  lower.tail = TRUE,
  log.p = FALSE
)

cogmod_geg(
  link_mu = "identity",
  link_sigma = "softplus",
  link_tau = "softplus",
  link_shape = "log"
)

cogmod_geg_lpdf_expose()

cogmod_geg_stanvars()

log_lik_cogmod_geg(i, prep)

posterior_predict_cogmod_geg(i, prep, ...)

posterior_epred_cogmod_geg(prep)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Location of the Gaussian component. Unbounded. Range: (-Inf, Inf).
sigma	SD of the Gaussian component. Must be positive. Range: (0, Inf).
tau	Mean of the exponential component. Must be positive. Range: (0, Inf).
shape	Power applied to the ex-Gaussian CDF. Must be positive. shape = 1 gives the ex-Gaussian back exactly. Range: (0, Inf).
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
q	Vector of quantiles.
lower.tail	Logical; if TRUE (default), probabilities are P(X <= q).
log.p	Logical; if TRUE, probabilities are returned on the log scale.

link_mu, link_sigma, link_tau, link_shape
: Character of the type of link used to model the GEG parameters. Defaults to "identity" for mu, "softplus" for sigma and tau, and "log" for shape.

  shape is on a log link so that zero on the link scale is shape = 1, the ex-Gaussian. A prior centred at zero is then a prior centred on the nested model, which is what cogmod_priors() supplies.

i, prep
: For brms' functions to run: index of the observation and a brms preparation object.

...
: Additional arguments.

Value

rcogmod_geg() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_geg() returns the density at each element of x - the log density if log = TRUE - and pcogmod_geg() the cumulative probability at each element of q, honouring lower.tail and log.p; both are recycled to the length of the longest argument. cogmod_geg() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_geg_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_geg_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_geg() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, posterior_predict_cogmod_geg() a draws x 1 matrix of reaction times simulated for observation i, and posterior_epred_cogmod_geg() a draws x observations matrix of expected reaction times, obtained by numerical integration.

What the shape parameter buys

A wider range of shapes than the ex-Gaussian can reach. Sweeping sigma and tau across the values RT data occupy, the ex-Gaussian spans skewness 0 to 2 and excess kurtosis 0 to 6; freeing shape widens that to roughly -0.4 to 4.8 and 0 to 35. In particular the GEG can be **negatively skewed**, which the ex-Gaussian cannot be at any parameter value.

What it costs

Interpretability, and specifically the one property that makes the ex-Gaussian worth using as a *descriptive* model.

- **The mean is no longer** mu + tau**.** With mu = 0.4 and tau = 0.2, the mean runs 0.31 at shape = 0.2, 0.60 at shape = 1 and 1.15 at shape = 20. There is no closed form for it - see posterior_epred() below - and the bulk/tail decomposition that the ex-Gaussian is normally reported for does not survive.
- shape **is strongly confounded with** mu**.** Fitted by maximum likelihood to the lexical-decision data used in the package vignettes, the correlation between the two at the optimum is about -0.98, and the other estimates move with it: on one condition mu goes 0.429 to 0.508, sigma 0.051 to 0.037 and tau 0.119 to 0.162 once shape is freed. shape does not add an independent axis so much as re-slice the same bulk-and-tail split.

The practical consequence is that cogmod_priors() gives shape a deliberately informative prior centred on the ex-Gaussian, and that mu, sigma and tau should not be read as the Gaussian centre,

the Gaussian SD and the mean of the tail once shape is free. If those quantities are the point of the analysis, fit cogmod_exgaussian() instead. If a better-fitting descriptive family is the point, cogmod_logstudent() and cogmod_loggamma() decouple skew from tail weight with parameters that stay interpretable.

Construction

The power transform is Durrans' alpha-power (or "exponentiated") family, and for integer shape it is the distribution of the maximum of shape independent ex-Gaussian draws. That is a mathematical device rather than an account of a process, so unlike cogmod_invgaussian()'s sigmadrift there is no mechanism attached to it.

References

- Marmolejo-Ramos, F., Barrera-Causil, C., Kuang, S., Fazlali, Z., Wegener, D., Kneib, T., De Bastiani, F., & Martinez-Florez, G. (2023). Generalised exponential-Gaussian distribution: A method for neural reaction time analysis. *Cognitive Neurodynamics*, *17*(1), 221-237. doi:10.1007/s11571022098132
- Durrans, S. R. (1992). Distributions of fractional order statistics in hydrology. *Water Resources Research*, *28*(6), 1649-1655. doi:10.1029/92WR00554

Examples

```
# shape = 1 is the ex-Gaussian, to machine precision
x <- seq(0.2, 2, length.out = 5)
dcogmod_geg(x, 0.4, 0.1, 0.2, shape = 1)
dcogmod_exgaussian(x, 0.4, 0.1, 0.2)

rts <- rcogmod_geg(1000, mu = 0.4, sigma = 0.1, tau = 0.2, shape = 2)
hist(rts, breaks = 50, xlab = "RT (s)")
```

---

rcogmod_invgamma *Shifted Inverse Gamma Model*

Description

Density, random generation, and brms custom family for the shifted Inverse Gamma distribution. A Inverse Gamma-distributed decision time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the decision process.

Functions:

- rcogmod_invgamma(): Simulates random draws.
- dcogmod_invgamma(): Computes the density (likelihood).
- cogmod_invgamma(): Creates a brms::custom_family().
- cogmod_invgamma_stanvars(): Generates the stanvars to pass to brm().

Usage

```
rcogmod_invgamma(n, mu = 4, sigma = 1.5, ndt = 0.2, poutlier = 0)

dcogmod_invgamma(x, mu = 4, sigma = 1.5, ndt = 0.2, poutlier = 0, log = FALSE)

cogmod_invgamma(
  link_mu = "softplus",
  link_sigma = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_invgamma_lpdf_expose()

cogmod_invgamma_stanvars()

log_lik_cogmod_invgamma(i, prep)

posterior_predict_cogmod_invgamma(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_invgamma(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Shape of the inverse Gamma decision time. Must be positive.
sigma	Scale of the inverse Gamma decision time. Must be positive.
ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LogNormal.
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_ndt, link_poutlier	Link functions for the parameters.
predict_outliers	Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

mu is the **shape** and sigma the **scale** of the inverse Gamma decision time, whose mean is sigma / (mu - 1) and exists only for mu > 1.

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a dpar, and why reaction times have to be in seconds.

posterior_epred() returns Inf where mu <= 1, because the inverse Gamma has no mean there. The right tail is a power law, so this family is the one to reach for when the slow tail is heavy; cogmod_loggamma() covers the same territory continuously through negative shape.

Value

rcogmod_invgamma() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_invgamma() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_invgamma() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_invgamma_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_invgamma_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_invgamma() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_invgamma() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_invgamma() returns a draws x observations matrix of expected reaction times, with Inf wherever the mean does not exist.

Examples

```
rts <- rcogmod_invgamma(1000, mu = 4, sigma = 1.5, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_invgamma(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_invgamma(0.1, ndt = 0.3, poutlier = 0)
```

---

rcogmod_invgaussian *Shifted Wald Model (Inverse Gaussian)*

Description

Density, distribution function, random generation, and brms custom family for the Shifted Wald distribution, also known as the Shifted Inverse Gaussian. A Wald-distributed decision time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the decision process. The drift rate can be fixed across trials (the classic Wald) or drawn afresh on each one, with SD sigmadrift.

Functions:

- `rcogmod_invgaussian()`: Simulates random draws.
- `dcogmod_invgaussian()`: Computes the density (likelihood).
- `pcogmod_invgaussian()`: Computes the cumulative distribution function (CDF).
- `cogmod_invgaussian()`: Creates a `brms::custom_family()`.
- `cogmod_invgaussian_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_invgaussian(
  n,
  drift = 3,
  boundary = 0.5,
  ndt = 0.2,
  sigmadrift = 0,
  poutlier = 0
)

dcogmod_invgaussian(
  x,
  drift = 3,
  boundary = 0.5,
  ndt = 0.2,
  sigmadrift = 0,
  poutlier = 0,
  log = FALSE
)

pcogmod_invgaussian(
  q,
  drift = 3,
  boundary = 0.5,
  ndt = 0.2,
  sigmadrift = 0,
  poutlier = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

cogmod_invgaussian(
  link_mu = "softplus",
  link_boundary = "softplus",
  link_sigmadrift = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_invgaussian_lpdf_expose()
```

```
cogmod_invgaussian_stanvars()

log_lik_cogmod_invgaussian(i, prep)

posterior_predict_cogmod_invgaussian(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_invgaussian(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`drift`	Drift rate. Must be positive. Represents the average speed of evidence accumulation. Range: (0, Inf).
`boundary`	Decision threshold (boundary separation). Must be positive. Represents the amount of evidence needed to make a decision. Range: (0, Inf).
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
`sigmadrift`	Between-trial SD of the drift rate. Must be non-negative. The drift of each trial is drawn from a `Normal(drift, sigmadrift)` truncated at zero. Default `0`, which is the classic fixed-drift Wald. Range: [0, Inf).
`poutlier`	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
`x`	Vector of quantiles (observed reaction times).
`log`	Logical; if TRUE, probabilities p are given as log(p).
`q`	Vector of quantiles (observed reaction times).
`lower.tail`	Logical; if TRUE (default), probabilities are `P[X <= x]`, otherwise `P[X > x]`.
`log.p`	Logical; if TRUE, probabilities p are given as log(p).
`link_mu, link_boundary, link_sigmadrift, link_ndt, link_poutlier`	Link functions for the parameters. `mu` is the drift rate. `sigmadrift` is legitimately zero, which is the classic Wald, and is fixed there by writing `sigmadrift = 0` in the formula.
`predict_outliers`	Logical; whether `posterior_predict()` and `posterior_epred()` should include the outlier component. `FALSE` (the default) fixes `poutlier` to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See `with_outliers()`.
`i, prep`	For brms' functions to run: index of the observation and a `brms` preparation object.
`...`	Additional arguments.

Details

The Wald distribution describes the time it takes for a Wiener diffusion process starting at 0 to reach a threshold `boundary` > 0, given a positive drift rate `drift` > 0. That time is then shifted by a non-decision time `ndt`.

It is mathematically equivalent to shifting an Inverse Gaussian distribution with mean `boundary / drift` and shape `boundary^2`. In the `brms` family the drift rate is named `mu`, since `brms` requires a parameter of that name.

`ndt` and `poutlier` mean exactly what they do in `cogmod_lognormal()`, and `with_outliers()`, `without_outliers()`, `p_outlier()` and `cogmod_priors()` all work here too. See `?rcogmod_lognormal` for why `ndt` is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a `dpar`, and why reaction times have to be in seconds.

The Wald density vanishes at the shift with all derivatives, like the LogNormal, so `ndt` is well behaved here and there is no unbounded-likelihood boundary of the kind `cogmod_gamma()` and `cogmod_weibull()` have at shape < 1.

The random generation algorithm is that of Michael, Schucany, and Haas (1976), as used in the `statmod` package.

Value

`rcogmod_invgaussian()` returns a numeric vector of `n` simulated reaction times, in seconds. `dcogmod_invgaussian()` returns the density at each element of `x` - the log density if `log = TRUE` - recycled to the length of the longest argument. `pcogmod_invgaussian()` returns the cumulative probability at each element of `q`, honouring `lower.tail` and `log.p`. `cogmod_invgaussian()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_invgaussian_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_invgaussian_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_invgaussian()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, and `posterior_predict_cogmod_invgaussian()` a draws x 1 matrix of reaction times simulated for observation `i`. `posterior_epred_cogmod_invgaussian()` returns a draws x observations matrix of expected reaction times, with `Inf` wherever the mean does not exist.

Across-trial drift variability

`sigmadrift` is the between-trial SD of the drift rate. At `sigmadrift = 0`, its default, every trial accumulates at the same rate and the model is the classic Wald. Above zero, each trial draws its own drift from a `Normal(mu, sigmadrift)` **truncated at zero**, which is what lets the model produce the long right tails empirical RT distributions have, and is the single-accumulator counterpart of what `cogmod_ddm()` calls `sigmadrift` and `cogmod_lba1()` calls `sigma`. Marginalising over that draw is a Gaussian integral, so the density stays closed form and costs two normal CDFs.

The truncation matters. A single-boundary accumulator handed a negative drift never terminates, so an untruncated Normal would leave the density defective: it integrates to 0.99 at `mu = 3, boundary = 0.5, sigmadrift = 1.` and to 0.69 at `mu = 0.5, boundary = 1, sigmadrift = 2`. `cogmod_ddm()` needs no such truncation, a diffusion between two boundaries always absorbing at one of them.

In a formula, `sigmadrift = 0` **fixes** the parameter and gives the classic Wald; leaving it out of `bf()` altogether *estimates* it, which is not the same thing:

```
# Classic Wald
brms::bf(rt ~ 1, boundary ~ 1, sigmadrift = 0, ndt ~ 1, poutlier ~ 1,
         family = cogmod_invgaussian())

# With across-trial drift variability
brms::bf(rt ~ 1, boundary ~ 1, sigmadrift ~ 1, ndt ~ 1, poutlier ~ 1,
         family = cogmod_invgaussian())
```

Fixing it is the better default, for two reasons. `sigmadrift` and `poutlier` both fatten the right tail and are only weakly distinguishable: on 2000 simulated trials at `mu = 3, boundary = 0.5, sigmadrift = 0.8`, estimating `sigmadrift` buys about 2 log-likelihood units over fixing it at zero, and the outlier weight absorbs most of the difference. And scaling `mu`, `boundary` and `sigmadrift` up by a common factor sends the Wald to the reciprocal-normal (LATER) limit, where the within-trial noise stops mattering and `RT = boundary / drift` exactly - so large values of all three describe nearly the same distribution. `cogmod_priors()` gives `sigmadrift` a deliberately informative prior to fence off both.

One consequence is worth stating plainly: with `sigmadrift > 0` the density decays as `t^-2`, because drifts arbitrarily close to zero take arbitrarily long, so **the mean does not exist**. posterior_epred() returns `Inf`, as it does for `cogmod_invgamma()` at `shape <= 1`. Use posterior_predict() and summarise the draws with a median or a quantile instead.

References

- Michael, J. R., Schucany, W. R., & Haas, R. W. (1976). Generating Random Variates Using Transformations with Multiple Roots. *The American Statistician*, *30*(2), 88-90. doi:10.2307/2683801
- Anders, R., Alario, F., & Van Maanen, L. (2016). The shifted Wald distribution for response time data analysis. *Psychological Methods*, *21*(3), 309-327. doi:10.1037/met0000063
- Matzke, D., & Wagenmakers, E. J. (2009). Psychological interpretation of the ex-Gaussian and shifted Wald parameters: A diffusion model analysis. *Psychonomic Bulletin & Review*, *16*(5), 798-817. doi:10.3758/PBR.16.5.798
- Folks, J. L., & Chhikara, R. S. (1978). The inverse Gaussian distribution and its statistical application-a review. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *40*(3), 263-275.
- Tillman, G., Van Zandt, T., & Logan, G. D. (2020). Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making. *Psychonomic Bulletin & Review*, *27*(5), 911-936. doi:10.3758/s13423020017196

Examples

```
# Simulate 1000 RTs with 2% outliers
rts <- rcogmod_invgaussian(1000, drift = 3, boundary = 0.5, ndt = 0.2, poutlier = 0.02)
hist(rts, breaks = 50, xlab = "RT (s)")

# The same, with the drift varying across trials: a longer right tail
```

```
rts_sv <- rcogmod_invgaussian(1000, drift = 3, boundary = 0.5, ndt = 0.2,
                              sigmadrift = 1, poutlier = 0.02)
quantile(rts, c(0.5, 0.99))
quantile(rts_sv, c(0.5, 0.99))

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_invgaussian(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_invgaussian(0.1, ndt = 0.3, poutlier = 0)
```

`rcogmod_invweibull` *Shifted Inverse Weibull Model*

Description

Density, random generation, and `brms` custom family for the shifted Inverse Weibull distribution. A Inverse Weibull-distributed decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_invweibull()`: Simulates random draws.
- `dcogmod_invweibull()`: Computes the density (likelihood).
- `cogmod_invweibull()`: Creates a `brms::custom_family()`.
- `cogmod_invweibull_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_invweibull(n, mu = 3, sigma = 0.4, ndt = 0.2, poutlier = 0)

dcogmod_invweibull(
  x,
  mu = 3,
  sigma = 0.4,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)

cogmod_invweibull(
  link_mu = "softplus",
  link_sigma = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_invweibull_lpdf_expose()
```

```
cogmod_invweibull_stanvars()

log_lik_cogmod_invweibull(i, prep)

posterior_predict_cogmod_invweibull(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_invweibull(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Shape of the inverse Weibull (Frechet) decision time. Must be positive.
sigma	Scale of the inverse Weibull (Frechet) decision time. Must be positive.
ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LogNormal.
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_ndt, link_poutlier	Link functions for the parameters.
predict_outliers	Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

mu is the **shape** and sigma the **scale** of the inverse Weibull (Frechet) decision time, whose mean is sigma * gamma(1 - 1 / mu) and exists only for mu > 1.

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a dpar, and why reaction times have to be in seconds.

posterior_epred() returns Inf where mu <= 1, because the Frechet has no mean there. cogmod_loggamma() nests this family at shape = -1.

Value

rcogmod_invweibull() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_invweibull() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_invweibull() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_invweibull_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_invweibull_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_invweibull() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_invweibull() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_invweibull() returns a draws x observations matrix of expected reaction times, with Inf wherever the mean does not exist.

Examples

```
rts <- rcogmod_invweibull(1000, mu = 3, sigma = 0.4, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_invweibull(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_invweibull(0.1, ndt = 0.3, poutlier = 0)
```

---

rcogmod_lba1 *Shifted Single-Accumulator LBA Model*

Description

Density, random generation, and brms custom family for a single-accumulator Linear Ballistic Accumulator. Evidence rises linearly and ballistically - no within-trial noise - from a start point drawn uniformly on [0, sigmabias] at a rate drawn from a normal truncated at zero, until it reaches the threshold b = sigmabias + boundary. That finishing time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead.

Fixing sigmabias = 0 gives the **recinormal**, or LATER, model, in which 1 / (RT - ndt) is normally distributed; see the section below.

Functions:

- rcogmod_lba1(): Simulates random draws.
- dcogmod_lba1(): Computes the density (likelihood).
- cogmod_lba1(): Creates a brms::custom_family().
- cogmod_lba1_stanvars(): Generates the stanvars to pass to brm().

Usage

```
rcogmod_lba1(
  n,
  drift = 3,
  sigma = 1,
  sigmabias = 0.5,
  boundary = 0.5,
  ndt = 0.3,
  poutlier = 0
)

dcogmod_lba1(
  x,
  drift = 3,
  sigma = 1,
  sigmabias = 0.5,
  boundary = 0.5,
  ndt = 0.3,
  poutlier = 0,
  log = FALSE
)

cogmod_lba1(
  link_mu = "softplus",
  link_sigma = "softplus",
  link_sigmabias = "softplus",
  link_boundary = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_lba1_lpdf_expose()

cogmod_lba1_stanvars()

log_lik_cogmod_lba1(i, prep)

posterior_predict_cogmod_lba1(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_lba1(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
drift	Mean drift rate.
sigma	Standard deviation of the drift rate. Conventionally fixed to 1.

`sigmabias` The starting-point range (`A`); must be non-negative. Zero is the recinormal (LATER) model rather than an invalid value - see the section above.

`boundary` The threshold offset, such that `b = sigmabias + boundary`; must be positive.

`ndt` Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).

`poutlier` Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.

`x` Vector of quantiles (observed reaction times).

`log` Logical; if TRUE, probabilities p are given as log(p).

`link_mu`, `link_sigma`, `link_sigmabias`, `link_boundary`, `link_ndt`, `link_poutlier`
Link functions for the parameters.

`predict_outliers`
Logical; whether `posterior_predict()` should include the outlier component. `FALSE` (the default) fixes `poutlier` to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See `with_outliers()`.

`i`, `prep` For brms' functions to run: index of the observation and a `brms` preparation object.

`...` Additional arguments.

Details

The full LBA is a race between one accumulator per response option, with the winner determining both the choice and the RT. With no choice to model there is nothing to race, so this is the single-accumulator version and the RT is just that one accumulator's finishing time. All the RT variability comes from across-trial variability in the start point and the drift rate, rather than from moment-to-moment noise within the trial.

The threshold is written as an *offset*, `b = sigmabias + boundary`, which is the `B` parameterization of `DMC` and `EMC2` (`b = B + A`) rather than the absolute threshold `rtdists` estimates. The threshold has to sit above the highest possible starting point, and the offset makes `b > sigmabias` hold automatically for any positive value instead of needing an order constraint between two estimated parameters. See `cogmod_lba2()` for the full note. The cost is that `boundary` alone is not the quantity to read off a fitted model; `boundary + sigmabias` is.

`sigma` is conventionally **fixed to 1** rather than estimated, because the evidence scale is arbitrary: multiplying `mu`, `sigma`, `sigmabias` and `boundary` by a common constant leaves the decision time `(b - start) / drift` unchanged, so only ratios are identified and one parameter must be pinned at a non-zero value to fix the scale. Fix it in the formula with `sigma = 1`.

Value

`rcogmod_lba1()` returns a numeric vector of `n` simulated reaction times, in seconds. `dcogmod_lba1()` returns the density at each element of `x` - the log density if `log = TRUE` - recycled to the length of the longest argument. `cogmod_lba1()` returns a `brms::custom_family` object, to put on a

`brms::bf()` formula. `cogmod_lba1_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_lba1_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_lba1()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, and `posterior_predict_cogmod_lba1()` a draws x 1 matrix of reaction times simulated for observation `i`. `posterior_epred_cogmod_lba1()` returns nothing: the decision time has no finite mean, so it errors rather than report one - summarise `posterior_predict()` draws instead.

The recinormal (LATER) special case

Setting `sigmabias = 0` removes the start-point variability altogether: the accumulator starts at zero on every trial, the decision time is `b / drift`, and `1 / (RT - ndt)` is therefore normally distributed. That is the *recinormal*, better known in the oculomotor literature as the **LATER** model of Carpenter and Williams (1995), whose `mu` and `sigma` are the mean and SD of *promptness* - the quantity a reciprobit plot puts on its axis.

This is not an approximation reached in the limit. At `sigmabias = 0` the density evaluates to `dnorm(b / t, drift, sigma) * b / t^2 / pnorm(drift / sigma)` exactly, to machine precision, in both the R and the Stan implementation. Two pins are needed rather than one, because zero is the one value the arbitrary evidence scale leaves alone and so `sigmabias` drops off the scale ray rather than pinning it:

```
# free: mu and sigma, the mean and SD of promptness
bf(rt ~ 1, sigmabias = 0, boundary = 1)
```

Because `sigmabias` is then a constant rather than a parameter, none of the trouble described next applies to it, and `cogmod_priors()` emits no row for it.

Estimating the start-point range

Left free, `sigmabias` is estimable but treacherous, precisely because the recinormal limit above is reached *smoothly*: once the start-point range is small enough, making it smaller stops changing the density, so the likelihood goes **flat**. On a `softplus` link zero is at minus infinity, so a flat prior there leaves the posterior improper, and the symptom is a chain that wanders off rather than one that fails. Fitted without priors on the 4285-trial data in `vignette("rt_models")`, `sigmabias` for one condition ran to `softplus(-10.4) = 3e-05` with `Rhat` 1.69 and an effective sample size of 6.

There are two ways out, and the choice is a modelling decision rather than a technical one. Pin `sigmabias = 0` and fit the recinormal, which is the honest option when the design cannot identify a start-point range. Or keep it free and fence the flat direction off with a prior: `cogmod_priors()` does this for both `sigmabias` and `boundary` - the threshold is `b = sigmabias + boundary`, so the two share the ridge - in the same way and for the same reason it fences off `ndt` and `poutlier`. Pass `prior = cogmod_priors(f, df)`; the defaults are weak (`normal(0, 1)` on the softplus scale, so a start-point range of roughly 0.3 to 1.3) and are meant to be replaced rather than relied on if you know more. The two are nested, so `loo_compare()` on the two fits is a like-for-like comparison through the same likelihood.

`ndt` and `poutlier` mean exactly what they do in `cogmod_lognormal()`, and `with_outliers()`, `without_outliers()` and `cogmod_priors()` work here too. See `?rcogmod_lognormal` for the full account.

Note that `posterior_epred()` is not available: the decision time has no finite mean, because `E[1 / drift]` diverges for a normal truncated at zero.

References

Carpenter, R. H. S., & Williams, M. L. L. (1995). Neural computation of log likelihood in control of saccadic eye movements. *Nature, 377*(6544), 59-62.

Examples

```
# Simulate 1000 trials with 2% outliers
rts <- rcogmod_lba1(1000, drift = 3, sigma = 1, sigmabias = 0.5, boundary = 0.5,
               ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# sigmabias = 0 is the recinormal (LATER): 1 / (RT - ndt) is normal
dcogmod_lba1(0.5, drift = 3, sigma = 1, sigmabias = 0, boundary = 0.5, ndt = 0.2)
dnorm(0.5 / 0.3, 3, 1) * 0.5 / 0.3^2 / pnorm(3)

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_lba1(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_lba1(0.1, ndt = 0.3, poutlier = 0)
```

`rcogmod_lba2` *Two-Accumulator Linear Ballistic Accumulator (LBA) Model*

Description

The Linear Ballistic Accumulator (LBA) treats a choice as a race between two accumulators that rise **linearly** - no within-trial noise at all - each at a rate drawn afresh on every trial. The first to reach the threshold determines both the reaction time and the choice. The observed RT is that decision time shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the race.

Functions:

- `rcogmod_lba2()`: Simulates random draws from the LBA.
- `dcogmod_lba2()`: Computes the density (likelihood).
- `cogmod_lba2()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_lba2_stanvars()`: Generates the `stanvars` to pass to `brm()`.
- `p_outlier()`: Per-trial posterior probability of being an outlier.

Usage

```
rcogmod_lba2(
  n,
  driftzero = 3,
  driftone = 3,
  sigmazero = 1,
  sigmaone = 1,
  sigmabias = 0.5,
  boundary = 0.5,
  ndt = 0.2,
  poutlier = 0
)

dcogmod_lba2(
  x,
  driftzero = 3,
  driftone = 3,
  sigmazero = 1,
  sigmaone = 1,
  sigmabias = 0.5,
  boundary = 0.5,
  ndt = 0.2,
  response,
  poutlier = 0,
  log = FALSE
)

cogmod_lba2(
  link_mu = "identity",
  link_driftone = "identity",
  link_sigmazero = "softplus",
  link_sigmaone = "softplus",
  link_sigmabias = "softplus",
  link_boundary = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_lba2_lpdf_expose()

cogmod_lba2_stanvars()

log_lik_cogmod_lba2(i, prep)

posterior_predict_cogmod_lba2(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_lba2(prep)
```

Arguments

n	Number of simulated trials. If length(n) > 1, the length is taken to be the number required.
driftzero, driftone	Mean drift rate of each accumulator (choice 0 and 1). Any real value; larger means faster. See Details on negative drifts.
sigmazero, sigmaone	Between-trial SD of each accumulator's drift rate. Must be positive.
sigmabias	Maximum starting point. The starting point of each accumulator on each trial is drawn from Uniform(0, sigmabias). Must be non-negative; 0 means both accumulators start at zero on every trial, so only the drift rates vary - the choice counterpart of the recinormal special case described in rcogmod_lba1().
boundary	Threshold offset, so the threshold is b = boundary + sigmabias. Must be positive.
ndt	Non-decision time (shift parameter), in seconds. Represents the time taken for processes unrelated to the decision (e.g., encoding, motor response). Must be non-negative. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the race. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LBA.
x	The observed reaction time (RT).
response	The winning accumulator (0 or 1). This gives the *defective* density f_response(x) * S_other(x), mixed with the outlier component, which is what a race likelihood needs.
log	Logical; if TRUE, returns the log-density. Default: FALSE.
link_mu, link_driftone	Link functions for the two mean drift rates. mu is driftzero: brms requires the first distributional parameter of a custom family to be called mu.
link_sigmazero, link_sigmaone	Link functions for the between-trial drift SDs.
link_sigmabias, link_boundary	Link functions for the start-point range and the threshold offset.
link_ndt, link_poutlier	Link functions for the non-decision time and the outlier rate.
predict_outliers	Logical; whether posterior_predict() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the race alone; the likelihood is always the full mixture either way. On the prediction method itself the default is NULL, which defers to the flag carried on the model - see with_outliers() to change it after fitting. See Details.
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Value

`rcogmod_lba2()` returns a data frame with `n` rows and two columns:

`rt`	The simulated reaction time.
`response`	The winning accumulator, coded `0` or `1`, matching the `dec()` coding used by the `brms` families.

`dcogmod_lba2()` returns the defective density at each element of `x` - the log density if `log = TRUE` - for the response given in `response`, recycled to the length of the longest argument. `cogmod_lba2()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_lba2_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_lba2_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_lba2()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, and `posterior_predict_cogmod_lba2()` a draws x 2 matrix of reaction times and choices simulated for observation `i`. `posterior_epred_cogmod_lba2()` returns nothing: the expected reaction time of a race has no closed form, so it errors rather than report one - summarise `posterior_predict()` draws instead.

Parameterization

Accumulator `k` starts at a point `z ~ Uniform(0, sigmabias)`, drawn afresh on every trial, and rises at a constant rate `v ~ Normal(drift_k, sigma_k)`, also drawn afresh on every trial, until it reaches the threshold `b = boundary + sigmabias`. Its finishing time is therefore `(b - z) / v`, and `boundary` is the threshold *offset*: the distance from the highest possible starting point to the threshold. All the randomness is between trials; within a trial the path is a straight line, which is what makes the density closed-form.

This is the `B` parameterization of `DMC` and `EMC2`, where `b = B + A` with `A` the start-point range. It is used for a reason rather than for taste: the threshold has to sit above the highest possible starting point, and writing the *offset* makes `b > A` hold automatically for any positive value. The alternative - estimating the absolute threshold, as `rtdists` does - needs an order constraint between two estimated parameters, which has to hold in every cell of the design once either of them carries a predictor. The cost is that `boundary` alone is not the quantity to read off a fitted model; `boundary + sigmabias` is.

`ndt` is expressed **directly, in seconds** (through a log link in the `brms` family). Nothing about it is taken from the data: it is not bounded by the fastest observed response, so a non-decision time that varies by condition or by participant can exceed the sample minimum wherever the data support it.

This replaces the earlier `tau` / `minrt` pair, in which `ndt = tau * minrt` with `minrt` set to the fastest observed RT. That capped the non-decision time at an order statistic of the sample, so any condition or participant whose true `ndt` exceeded the fastest observed response was inexpressible, and the misfit surfaced as spurious effects on the race parameters.

Negative drift rates

A normal drift rate can come out negative, and such an accumulator rises away from the threshold and never responds. A trial on which *both* drifts are negative produces no response at all, so it is not a trial: the process is conditioned on at least one of the two being positive, and `rcogmod_lba2()` draws from exactly that conditional distribution.

The density is conditioned to match, dividing by `1 - pnorm(-driftzero / sigmazero) * pnorm(-driftone / sigmaone)`. Without that factor the density integrates to the probability of the event rather than to one, which at low drift rates is a long way short - `0.83` at drifts of `0.5` and `0.2` with SDs of `1.5`, so the likelihood is wrong by 17% and wrong by *different* amounts at different parameter values, which is what makes it bias estimates rather than merely offset them.

The evidence scale is arbitrary

Multiply `driftzero`, `driftone`, `sigmazero`, `sigmaone`, `sigmabias` and `boundary` all by any `c > 0` and every finishing time `(b - z) / v` is unchanged. The likelihood is therefore **exactly** constant along that ray, which runs to infinity in both directions: the six parameters are identified only up to a common scale factor.

`cogmod_priors()` puts priors on all four positive parameters, which makes the posterior proper and the sampler well behaved, but a prior does not *identify* a direction the likelihood cannot see. If the individual parameters are to be interpreted - rather than the RT distribution they jointly generate, which is perfectly well identified - fix one SD in the formula, the usual convention being `sigmazero = 1`:

```
f <- brms::bf(RT | dec(Error) ~ Condition, driftone ~ Condition,
              sigmazero = 1, sigmaone ~ 1, sigmabias ~ 1, boundary ~ 1,
              ndt ~ 1, poutlier ~ 1, family = cogmod_lba2())
```

A second, milder flat direction remains either way: `sigmabias` and `boundary` enter only through the sum `b = boundary + sigmabias`, so they trade off along a ridge. The sum is the trustworthy quantity to interpret and to compare across conditions. `cogmod_lba1()` and `cogmod_rdm()` share it.

The outlier component

A shifted distribution assigns exactly zero density to any response faster than `ndt`, which puts a hard boundary in the likelihood at the fastest observed RT. Mixing in a component with support over the whole positive line removes it: every response keeps positive density whatever `ndt` is, so the boundary becomes a finite cost rather than a wall and the log-density stays smooth and differentiable.

Because this model produces a **choice as well as a time**, the contaminant has to produce both. It is a guess: the choice is uniform over the two options, and the RT is a **half Normal** with scale `0.2` seconds.

$$f(t,k) = p\frac{1}{K}g(t) + (1-p)f_k(t-ndt)$$

The `1 / K` is what keeps the total summing to one over the response options; without it it would come to `1 + poutlier`.

`poutlier` is a *rate*, not a classification: the model never labels individual trials, it estimates what share of them came from elsewhere. Use `p_outlier()` for per-trial posterior probabilities.

Reaction times must be in seconds

The outlier component's scale is a constant in seconds, and so are the priors `cogmod_priors()` supplies. There is no argument for changing the unit: the `minrt` argument that used to rescale the component was removed in 0.2.1. Millisecond data fails silently rather than loudly - the outlier component contributes nothing and the min-RT boundary comes back. See the corresponding section of `cogmod_lognormal()` for the full account, which applies unchanged here.

Fitting

```
f <- brms::bf(RT | dec(Error) ~ Condition, driftone ~ Condition,
              sigmazero = 1, sigmaone ~ 1, sigmabias ~ 1, boundary ~ 1,
              ndt ~ 1, poutlier ~ 1, family = cogmod_lba2())
brms::brm(f, data = df,
          prior    = cogmod_priors(f, df),
          init     = cogmod_inits(f, df),
          stanvars = cogmod_stanvars(f))
```

The `brms` family names the drift of the first accumulator `mu` (as `brms` requires) and that of the second `driftone`. Use `cogmod_inits()` rather than `init = 0`: `brms` initialises on the unconstrained scale, so `init = 0` puts `ndt` at `exp(0) = 1` second - above nearly every sub-second RT, which leaves every response attributed to the outlier component and the race parameters with no gradient at all.

Predictions exclude the outlier component

`posterior_predict()` describes the **race alone** by default, as if `poutlier` were zero, because the outlier component is a fixed regularizer rather than a claim about how guesses are distributed. Use `with_outliers()` for the fitted mixture - chiefly for `brms::pp_check()` - and `without_outliers()` to go back. `log_lik()` is always the full mixture.

`posterior_epred()` is not provided: for a race model the expectation needs numerical integration per draw and per observation, and users are better off summarising `posterior_predict()` draws.

References

- Brown, S. D., & Heathcote, A. (2008). The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive Psychology*, *57*(3), 153-178. doi:10.1016/j.cogpsych.2007.12.002

See Also

`rcogmod_lba1()`, `rcogmod_rdm()`, `rcogmod_lnr()`

Examples

```
# Simulate data, with 2% of trials from the outlier process
data <- rcogmod_lba2(1000,
  driftzero = 3, driftone = 2, sigmazero = 1, sigmaone = 1,
  sigmabias = 0.5, boundary = 0.5, ndt = 0.2, poutlier = 0.02
)
head(data)
```

```
# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_lba2(0.1, ndt = 0.2, response = 0, poutlier = 0.02)
dcogmod_lba2(0.1, ndt = 0.2, response = 0, poutlier = 0)


# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpdf <- cogmod_lba2_lpdf_expose()
  lpdf(
    Y = 0.5, mu = 3, driftone = 2, sigmazero = 1, sigmaone = 1,
    sigmabias = 0.5, boundary = 0.5, ndt = 0.2, poutlier = 0.02, dec = 0
  )
}
```

`rcogmod_lnr` *Log-Normal Race (LNR) Model*

Description

The Log-Normal Race (LNR) model is useful for modeling reaction times and choices in decision-making tasks. Each choice option (accumulator) draws a processing time from a LogNormal distribution; the winning accumulator (the minimum draw) determines both the observed reaction time and the choice. The observed RT is that decision time shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the race.

Functions:

- `rcogmod_lnr()`: Simulates random draws from the LNR model.
- `dcogmod_lnr()`: Computes the density (likelihood).
- `cogmod_lnr()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_lnr_stanvars()`: Generates the `stanvars` to pass to `brm()`.
- `p_outlier()`: Per-trial posterior probability of being an outlier.

Usage

```
rcogmod_lnr(
  n,
  nuzero = 0,
  nuone = 0,
  sigmazero = 1,
  sigmaone = 1,
  ndt = 0.2,
  poutlier = 0
)
```

```
dcogmod_lnr(
  x,
  nuzero = 0,
  nuone = 0,
  sigmazero = 1,
  sigmaone = 1,
  ndt = 0.2,
  response,
  poutlier = 0,
  log = FALSE
)

cogmod_lnr(
  link_mu = "identity",
  link_nuone = "identity",
  link_sigmazero = "softplus",
  link_sigmaone = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_lnr_lpdf_expose()

cogmod_lnr_stanvars()

log_lik_cogmod_lnr(i, prep)

posterior_predict_cogmod_lnr(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_lnr(prep)
```

Arguments

n	Number of simulated trials. If length(n) > 1, the length is taken to be the number required.
nuzero, nuone	The (inverse of the) log-space mean parameter for both accumulators (choice 0 and 1). Controls the central tendency of the reaction time. Can take any real value (-Inf, Inf), with larger values leading to *faster* RTs. Named 'nu' (=-meanlog) for consistency with other race models.
sigmazero, sigmaone	The log-space standard deviation for both accumulators (choice 0 and 1). Controls the variability of reaction times. Must be positive (0, Inf). Larger values increase variability.
ndt	Non-decision time (shift parameter), in seconds. Represents the time taken for processes unrelated to the decision (e.g., encoding, motor response). Must be non-negative. Range: [0, Inf).

poutlier — Proportion of responses generated by the outlier process rather than by the race. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LNR.

x — The observed reaction time (RT).

response — The decision indicator (0 or 1). 0 for choice 0, 1 for choice 1.

log — Logical; if TRUE, returns the log-density. Default: FALSE.

link_mu, link_nuone — Link function for the nu parameters. mu is nuzero: brms requires the first distributional parameter of a custom family to be called mu, so that is the name the formula and this argument use, and nuzero is what it means.

link_sigmazero, link_sigmaone — Link function for the sigma parameters.

link_ndt, link_poutlier — Link functions for the non-decision time and the outlier rate.

predict_outliers — Logical; whether posterior_predict() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the race alone; the likelihood is always the full mixture either way. On the prediction method itself the default is NULL, which defers to the flag carried on the model - see with_outliers() to change it after fitting. See Details.

i, prep — For brms' functions to run: index of the observation and a brms preparation object.

... — Additional arguments.

Value

rcogmod_lnr() returns a data frame with n rows and two columns, rt (the simulated reaction time, in seconds) and response (the boundary reached, 0 or 1, matching the dec() coding used by the brms family). dcogmod_lnr() returns the defective density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_lnr() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_lnr_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_lnr_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_lnr() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_lnr() a draws x 2 matrix of reaction times and choices simulated for observation i. posterior_epred_cogmod_lnr() returns nothing: the expected reaction time of a race has no closed form, so it errors rather than report one - summarise posterior_predict() draws instead.

Parameterization

Each accumulator k finishes at a LogNormal time with meanlog = -nu_k and sdlog = sigma_k, so **larger** nu **means faster**. The observed reaction time is ndt + min(T_0, T_1) and the observed choice is whichever accumulator got there first.

ndt is expressed **directly, in seconds** (through a log link in the brms family). Nothing about it is taken from the data: it is not bounded by the fastest observed response, so a non-decision time that varies by condition or by participant can exceed the sample minimum wherever the data support it.

This replaces the earlier tau / minrt pair, in which ndt = tau * minrt with minrt set to the fastest observed RT. That capped the non-decision time at an order statistic of the sample, so any condition or participant whose true ndt exceeded the fastest observed response was inexpressible, and the misfit surfaced as spurious effects on the race parameters.

The outlier component

A shifted distribution assigns exactly zero density to any response faster than ndt, which puts a hard boundary in the likelihood at the fastest observed RT. Mixing in a component with support over the whole positive line removes it: every response keeps positive density whatever ndt is, so the boundary becomes a finite cost rather than a wall and the log-density stays smooth and differentiable. That is what makes the direct parameterization of ndt workable without taking a bound from the data.

Because this model produces a **choice as well as a time**, the contaminant has to produce both. It is a guess: the choice is uniform over the two options, and the RT is a **half Normal** with scale 0.2 seconds.

$$f(t,k) = p\frac{1}{K}g(t) + (1-p)f_k(t - ndt)$$

The 1 / K is what keeps the total summing to one over the response options; without it it would come to 1 + poutlier. A half Normal is used for the timing because it is **flat at the origin** (zero derivative), so the very fastest responses - the ones least plausibly decisions - are not starved of density, and because it dies away fast enough above ndt to leave the slow tail to the decision process. Plot it with curve(2 * dnorm(x, 0, 0.2), 0, 3).

poutlier is a *rate*, not a classification: the model never labels individual trials, it estimates what share of them came from elsewhere. Use p_outlier() for per-trial posterior probabilities.

Reaction times must be in seconds

The outlier component's scale is a constant in seconds, and so are the priors cogmod_priors() supplies. There is no argument for changing the unit: the minrt argument that used to rescale the component was removed in 0.2.1. Millisecond data fails silently rather than loudly - the outlier component contributes nothing and the min-RT boundary comes back. See the corresponding section of cogmod_lognormal() for the full account, which applies unchanged here.

Fitting

```
f <- brms::bf(RT | dec(Error) ~ Condition, nuone ~ Condition,
              sigmazero ~ 1, sigmaone ~ 1, ndt ~ 1, poutlier ~ 1,
              family = cogmod_lnr())
brms::brm(f, data = df,
          prior    = cogmod_priors(f, df),
          init     = cogmod_inits(f, df),
          stanvars = cogmod_stanvars(f))
```

Use cogmod_inits() rather than init = 0. brms initialises on the unconstrained scale, so init = 0 puts ndt at exp(0) = 1 second - above nearly every sub-second RT, which leaves every response attributed to the outlier component and the race parameters with no gradient at all.

cogmod_priors() is not a convenience here either. Beyond ndt and poutlier, a race has a flat direction of its own: push an accumulator's rate far enough down and it stops finishing first *ever*, so the density depends on it only through the loser's survival term, which has already saturated at 1. Past about nuone = -6 the log-likelihood is **exactly** constant, and that accumulator's sigma is unidentified along with it - nothing is left for it to act on. On the identity link nuone uses, that is an unbounded flat region under a flat prior: an improper posterior, the same failure as poutlier running to 1.

The outlier component makes this reachable rather than hypothetical. Without it, an accumulator that never wins would still have to explain the trials on which the *other* one lost, and the likelihood would object. With it, those trials are floored by the contaminant instead, so the plateau is there even when both responses are well represented. cogmod_priors() fences off nuone, sigmazero and sigmaone for this reason.

mu is nuzero and has the mirror-image plateau, but it is the response's own intercept, so brms already gives it a proper student_t default and cogmod_priors() leaves it alone. If one option is chosen only rarely, the accumulator that loses is the one at risk, and it is worth putting the same prior on both by hand:

```
priors <- c(cogmod_priors(f, df),
            brms::prior(normal(0.7, 1.5), class = "Intercept"),
            replace = TRUE)
```

Predictions exclude the outlier component

posterior_predict() describes the **race alone** by default, as if poutlier were zero, because the outlier component is a fixed regularizer rather than a claim about how guesses are distributed. Use with_outliers() for the fitted mixture - chiefly for brms::pp_check() - and without_outliers() to go back. log_lik() is always the full mixture.

posterior_epred() is not provided: for a race model the expectation needs numerical integration per draw and per observation, and users are better off summarising posterior_predict() draws.

References

- Rouder, J. N., Province, J. M., Morey, R. D., Gomez, P., & Heathcote, A. (2015). The log-normal race: A cognitive-process model of choice and latency with desirable psychometric properties. Psychometrika, 80(2), 491-513. doi:10.1007/s1133601393963

Examples

```
# Simulate data, with 2% of trials from the outlier process
data <- rcogmod_lnr(1000,
  nuzero = 1, nuone = 0.5, sigmazero = 1, sigmaone = 0.8,
  ndt = 0.2, poutlier = 0.02
)
head(data)
```

```
# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_lnr(0.1, ndt = 0.2, response = 0, poutlier = 0.02)
dcogmod_lnr(0.1, ndt = 0.2, response = 0, poutlier = 0)


# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpdf <- cogmod_lnr_lpdf_expose()
  lpdf(
    Y = 0.5, mu = 0.5, nuone = 0.2, sigmazero = 1.0, sigmaone = 0.8,
    ndt = 0.2, poutlier = 0.02, dec = 0
  )
}
```

`rcogmod_loggamma` *Shifted Log-Gamma (Generalized Gamma) Model*

Description

Density, random generation, and `brms` custom family for the shifted Log-Gamma distribution. A Log-Gamma-distributed decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process, exactly as in `cogmod_lognormal()`.

Functions:

- `rcogmod_loggamma()`: Simulates random draws from the shifted Log-Gamma model.
- `dcogmod_loggamma()`: Computes the density (likelihood).
- `cogmod_loggamma()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_loggamma_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_loggamma(n, mu = -0.7, sigma = 0.5, shape = 0, ndt = 0.2, poutlier = 0)

dcogmod_loggamma(
  x,
  mu = -0.7,
  sigma = 0.5,
  shape = 0,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)
```

```
cogmod_loggamma(
  link_mu = "identity",
  link_sigma = "softplus",
  link_shape = "identity",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_loggamma_lpdf_expose()

cogmod_loggamma_stanvars()

log_lik_cogmod_loggamma(i, prep)

posterior_predict_cogmod_loggamma(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_loggamma(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
mu	Location of the decision time on the log scale. Can take any real value. Range: (-Inf, Inf).
sigma	Scale of the decision time on the log scale. Must be positive. Range: (0, Inf).
shape	Shape (skewness) of the log-gamma on the log-RT scale. Unconstrained: `shape = 0` is the LogNormal, `shape = sigma` the Gamma, `shape = 1` the Weibull. See Details for the `sigma * shape >= 1` boundary. Range: (-Inf, Inf).
ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_shape, link_ndt, link_poutlier	Link functions for the parameters. `shape` is unconstrained and takes an `identity` link, so that the LogNormal (`shape = 0`) sits in the interior of its range rather than at a boundary.
predict_outliers	Logical; whether `posterior_predict()` and `posterior_epred()` should include the outlier component. `FALSE` (the default in `cogmod_loggamma()`) fixes `poutlier` to zero for prediction, so predictions describe the decision process

	alone; the likelihood is always the full mixture either way. On the prediction methods themselves the default is NULL, which defers to the flag carried on the model - see with_outliers() to change it after fitting.
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Value

rcogmod_loggamma() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_loggamma() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_loggamma() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_loggamma_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_loggamma_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_loggamma() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_loggamma() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_loggamma() returns a draws x observations matrix of expected reaction times.

What "Log-Gamma" means here

The **log-gamma** distribution is the distribution of log(G) for a Gamma variate G. Used as the distribution of the *log decision time* - in the same way the Normal is used in the shifted LogNormal - it gives a location-scale family on the log scale with one extra shape parameter:

```
log(RT - ndt) = mu + sigma * w,   w ~ standardized log-gamma with k = 1 / shape^2
```

Equivalently, RT - ndt follows a **generalized gamma** distribution (Stacy, 1962) in the parameterization of Prentice (1974), i.e. flexsurv::dgengamma(mu, sigma, Q = shape). The two names describe the same model: "log-gamma" names the distribution of log(RT - ndt), "generalized gamma" names the distribution of RT - ndt itself.

A **two-parameter** log-gamma is not a usable RT model, which is why there is a third parameter here. Exponentiating a plain two-parameter Gamma variate gives support on (1, Inf), so decision times would be forced above one second; adding a scale to fix that produces a second shift, perfectly confounded with ndt; and letting log(RT - ndt) be log-gamma with no location or scale just gives back the Gamma. Only the three-parameter location-scale-shape version is both non-degenerate and closed under a change of time unit (mu -> mu + log(c)), which everything else here relies on.

Relation to other "log-gamma" implementations

The name is used for two different distributions, and only one of them is this one.

scipy.stats.loggamma is the same distribution. It is log(G) for G ~ Gamma(c), with the usual loc and scale, so it is a three-parameter family exactly as this one is - c is the shape parameter, and it is not optional there either. Taking scipy's variate to be log(RT - ndt), the two line up exactly (verified to 3e-15):

`c = 1 / shape^2` `scale = sigma / shape` `loc = mu + (sigma / shape) * log(shape^2)`

Two deliberate differences. `shape` here is standardized so that the Normal limit sits at `shape = 0`, an interior point; in `scipy`'s parameterization that limit is `c -> Inf` with `loc` and `scale` drifting off to compensate, which is not something a sampler can explore. And because `scipy` requires `scale > 0`, it covers only `shape > 0`; the `shape < 0` half here - the inverse-Weibull side, with the power-law right tail - is the reflection, and would need `-loggamma` there.

`actuar::dlgamma` is a *different* distribution: `exp(G)` rather than `log(G)`, hence its support of `(1, Inf)`. That is the version with only two parameters, and the reason it needs only two is also the reason it is no use for reaction times - see the paragraph above on why exponentiating a Gamma does not give a usable RT model.

`shape`, and the families it nests

`shape` sets the skewness of the log-gamma on the log-RT scale; the Gamma it is the log of has its own shape `k = 1 / shape^2`. Throughout the docs below, "shape" unqualified means this parameter, never `k`.

It is unconstrained, with `shape = 0` in the *interior* rather than at a boundary, which is what makes it usable as a free parameter:

shape	Distribution	right tail
`< -1`	heavier still than the inverse Weibull	power law
`= -1`	**inverse Weibull** (Frechet)	power law
`-1 to 0`	between the LogNormal and the inverse Weibull	power law
`= 0`	**LogNormal** - exactly `cogmod_lognormal()`	lognormal
`0 to 1`	between the LogNormal and the Weibull; **Gamma** (shape `1 / sigma^2`) at `shape = sigma`	lighter than lognorma
`= 1`	**Weibull**, shape `1 / sigma`	lighter than lognorma
`> 1`	lighter still than the Weibull	lightest

The right tail decays like `exp(-c * t^(shape / sigma))` for `shape > 0`, so it thins monotonically as `shape` rises, and becomes a power law for `shape < 0`. `shape` therefore runs from heavy-tailed at the top of the table to light-tailed at the bottom, through the LogNormal in the middle. The Gamma sits inside `0 to 1` for any `sigma < 1`, which covers most RT data.

The model is therefore a strict generalisation of the shifted LogNormal, and fitting it is a way of *testing* whether the LogNormal shape is adequate: an interval for `shape` covering 0 says it is.

Where it misbehaves: `sigma * shape >= 1`

Just above the shift the decision density behaves like a Gamma whose own shape parameter is `1 / (sigma * shape)`. When `sigma * shape >= 1` that Gamma shape falls below 1 and the density becomes **unbounded at** `ndt`, so the likelihood can be driven up without limit by pushing `ndt` toward the fastest response - the exact pathology the outlier component exists to remove, reintroduced through the shape parameter. The outlier component cannot repair it, because it adds density rather than capping it.

This is the same degeneracy the shifted Gamma and shifted Weibull have when their own shape falls below 1; it is inherited here, not introduced. In practice the prior on `shape` is what keeps you out of

it: cogmod_priors() uses normal(0, 0.5) on the intercept, which for a typical sigma around 0.5 leaves the boundary at shape = 2, four prior SDs away. A posterior for shape pushing up against 1 / sigma is the model asking for a spike at the shift, not for a decision-time distribution.

Negative shape has the mirror-image caveat: the right tail is a power law, and the mean of the decision component is finite only when sigma * abs(shape) < 1. posterior_epred() returns Inf where it is not.

Fit with init = 0

The prior keeps the *posterior* clear of that boundary, but it does not control where a chain **starts**. brms initialises on the unconstrained scale from U(-2, 2), which for the default links puts shape in (-2, 2) and sigma in (0.13, 2.13) - and about **15% of chains start with** sigma * shape >= 1. A chain starting inside the unbounded region falls into the spike at ndt and does not come back out: it does not error, it simply runs for as long as you let it while the others finish.

init = 0 removes the problem by construction, starting every chain at shape = 0 - the LogNormal - with sigma * shape = 0:

```
f <- brms::bf(RT ~ 1, sigma ~ 1, shape ~ 1, ndt ~ 1, poutlier ~ 1,
              family = cogmod_loggamma())
brms::brm(f, data = df,
          prior = cogmod_priors(f, df),
          stanvars = cogmod_stanvars(f),
          init = 0)
```

This is not a tuning suggestion to try if sampling looks bad; it is how the model should be fitted. The one visible symptom of getting it wrong is a chain that never finishes.

ndt and poutlier

Identical in meaning, parameterization and defaults to cogmod_lognormal() - see its Details for the full account of why ndt is expressed directly in seconds, what the half Normal outlier component is for, why its scale is a constant rather than a dpar, and why predictions exclude the outlier component by default. with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work on this family too.

References

Stacy, E. W. (1962). A generalization of the gamma distribution. *The Annals of Mathematical Statistics*, 33(3), 1187-1192. doi:10.1214/aoms/1177704481

Prentice, R. L. (1974). A log gamma model and its maximum likelihood estimation. *Biometrika*, 61(3), 539-544. doi:10.1093/biomet/61.3.539

Examples

```
# shape = 0 is exactly the shifted LogNormal
dcogmod_loggamma(0.9, mu = -0.7, sigma = 0.5, shape = 0, ndt = 0.3)
dcogmod_lognormal(0.9, mu = -0.7, sigma = 0.5, ndt = 0.3)

# Simulate 1000 RTs with 2% outliers and a slightly Gamma-like shape
```

```
rts <- rcogmod_loggamma(1000,
  mu = -0.7, sigma = 0.5, shape = 0.5, ndt = 0.3,
  poutlier = 0.02
)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density, as in cogmod_lognormal()
dcogmod_loggamma(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_loggamma(0.1, ndt = 0.3, poutlier = 0)

# shape = sigma is the shifted Gamma, with shape 1 / sigma^2
dcogmod_loggamma(0.9, mu = -0.7, sigma = 0.5, shape = 0.5, ndt = 0.3)
stats::dgamma(0.6, shape = 4, scale = exp(-0.7) * 0.25)
```

`rcogmod_lognormal` *Shifted LogNormal Model*

Description

Density, random generation, and `brms` custom family for the shifted LogNormal distribution. A LogNormal-distributed decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_lognormal()`: Simulates random draws from the shifted LogNormal model.
- `dcogmod_lognormal()`: Computes the density (likelihood).
- `cogmod_lognormal()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_lognormal_stanvars()`: Generates the `stanvars` to pass to `brm()`.
- `p_outlier()`: Per-trial posterior probability of being an outlier.

Usage

```
rcogmod_lognormal(n, mu = -0.7, sigma = 0.5, ndt = 0.2, poutlier = 0)

dcogmod_lognormal(
  x,
  mu = -0.7,
  sigma = 0.5,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)

cogmod_lognormal(
  link_mu = "identity",
  link_sigma = "softplus",
```

```
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_lognormal_lpdf_expose()

cogmod_lognormal_stanvars()

log_lik_cogmod_lognormal(i, prep)

posterior_predict_cogmod_lognormal(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_lognormal(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mu`	Mean of the decision time on the log scale (`meanlog`). Can take any real value. Range: (-Inf, Inf).
`sigma`	SD of the decision time on the log scale (`sdlog`). Must be positive. Range: (0, Inf).
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
`poutlier`	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
`x`	Vector of quantiles (observed reaction times).
`log`	Logical; if TRUE, probabilities p are given as log(p).
`link_mu, link_sigma, link_ndt, link_poutlier`	Link functions for the parameters.
`predict_outliers`	Logical; whether `posterior_predict()` and `posterior_epred()` should include the outlier component. `FALSE` (the default in `cogmod_lognormal()`) fixes `poutlier` to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. On the prediction methods themselves the default is `NULL`, which defers to the flag carried on the model - see `with_outliers()` to change it after fitting. See Details.
`i, prep`	For brms' functions to run: index of the observation and a `brms` preparation object.
`...`	Additional arguments.

Value

`rcogmod_lognormal()` returns a numeric vector of `n` simulated reaction times, in seconds. `dcogmod_lognormal()` returns the density at each element of `x` - the log density if `log = TRUE` - recycled to the length of the longest argument. `cogmod_lognormal()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_lognormal_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_lognormal_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_lognormal()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, and `posterior_predict_cogmod_lognormal()` a draws x 1 matrix of reaction times simulated for observation `i`. `posterior_epred_cogmod_lognormal()` returns a draws x observations matrix of expected reaction times.

Parameterization

The observed reaction time is `ndt + LogNormal(mu, sigma)`, so `mu` and `sigma` are the mean and SD of the *decision time* on the log scale, and the median reaction time is `ndt + exp(mu)`.

`ndt` is expressed **directly, in seconds** (through a log link in the `brms` family). Nothing about it is taken from the data: it is not bounded by the fastest observed response, so a non-decision time that varies by condition or by participant can exceed the sample minimum wherever the data support it.

The outlier component

A shifted distribution assigns exactly zero density to any response faster than `ndt`, which puts a hard boundary in the likelihood at the fastest observed RT. Mixing in a component with support over the whole positive line removes it: every response keeps positive density whatever `ndt` is, so the boundary becomes a finite cost rather than a wall and the log-density stays smooth and differentiable. That is what makes the direct parameterization of `ndt` workable without taking a bound from the data.

The outlier component is a **half Normal** with scale `0.2` seconds, i.e. `2 * dnorm(x, 0, 0.2)` on `[0, Inf)`. Two properties motivate the shape. It is **flat at the origin** (zero derivative), so the very fastest responses - the ones least plausibly decisions - are not starved of density; a LogNormal or Gamma vanishes at zero and an Exponential peaks there with maximal slope, and all three get this backwards. And it stays close to flat across the whole range `ndt` plausibly occupies - 76% of its peak at 0.15 s and 46% at 0.25 s - while dying fast enough above that to leave the slow tail alone. Plot it with `curve(2 * dnorm(x, 0, 0.2), 0, 3)`.

Up to version 0.2.0 this was a half Student-t with 3 degrees of freedom and a user-supplied scale. That tail was heavier than every decision density in the package, so far-out slow responses were eventually better explained by the outlier component than by the model: at `poutlier = 0.02` a 5 s response was attributed to it with probability 0.86, and `ndt` was pulled up behind it. The slow tail now belongs to the decision family, which is what `cogmod_loggamma()`'s `shape` and `cogmod_invgaussian()`'s `sigmadrift` are for.

Reaction times must be in seconds

The outlier component's scale is a **constant in seconds**, and so are the priors `cogmod_priors()` supplies - `ndt` at roughly 0.17 to 0.30 s, `sigmandt` in `cogmod_ddm()` at 0.05 s, and so on. There is no argument for changing the unit, and no unit conversion anywhere in the package.

Feeding it milliseconds fails **silently**, which is worth knowing about. The outlier component's log-density at `RT = 400` is about `-2e6`, so it contributes nothing anywhere in the data and the mixture collapses to the unmixed shifted family: `poutlier` goes to zero and `ndt` is pinned by the fastest observed response again - exactly the min-RT boundary this parameterization exists to remove. Nothing errors, and the chains still initialise, because the decision density itself stays finite.

This failure was already reachable before 0.2.1 by leaving `minrt` at its default with millisecond data. Removing the argument makes it unconditional rather than optional, which is the trade: the equivariance `minrt` bought in the likelihood was already lost in the priors, and `cogmod_priors()` is not optional.

Divide by 1000 before fitting, and multiply `ndt` back afterwards if you want the answer in milliseconds.

`poutlier` is a *rate*, not a classification: the model never labels individual trials, it estimates what share of them came from elsewhere. Use `p_outlier()` for per-trial posterior probabilities.

Trimmed data: pin `poutlier` down, but not to zero

`poutlier` is only weakly identified when there is little to identify it *from* - which is why `cogmod_priors()` gives it an informative prior rather than leaving it flat. If the data have already been trimmed, or only a handful of implausibly fast responses remain, it is reasonable to stop asking the data to estimate a rate at all.

The right way to do that is a **very tight prior near zero**, not a hard zero:

```
f <- brms::bf(RT ~ 1, sigma ~ 1, ndt ~ 1, poutlier ~ 1,
              family = cogmod_lognormal())
priors <- c(
  cogmod_priors(f, df),
  brms::prior(normal(-7, 0.5), class = "Intercept", dpar = "poutlier"),
  replace = TRUE
)
```

`normal(-7, 0.5)` on the logit scale is centred at about 0.09%, with 95% of its mass between 0.03% and 0.24% - small enough to assert "there is essentially no contamination here", while leaving the rate free to rise if the data insist.

Fixing it outright is also possible, with `poutlier = 0` in the `bf()`, which makes `brms` treat it as a constant and reduces the model to the plain shifted family. **Prefer the tight prior.** At exactly zero the density is once again exactly zero below `ndt`, so the hard min-RT boundary returns and `ndt` is pinned by the fastest observed response - which is the very problem the outlier component was introduced to solve, reintroduced deliberately. A rate of 0.1% is numerically negligible for every other purpose but still keeps the density positive below `ndt`, so the likelihood stays smooth and `ndt` stays free.

Trim first either way. Neither option makes slow contaminants safe: those are confounded with the right tail and bias `ndt` upward, so filter them before fitting.

Slow outliers are deliberately not handled by this component. A slow contaminant is statistically confounded with the right tail of the RT distribution itself, so it cannot be identified, and leaving such trials in the data biases `ndt` upward. Filter implausibly slow responses before fitting.

Predictions exclude the outlier component

`posterior_predict()` and `posterior_epred()` describe the **decision process alone** by default, as if `poutlier` were zero. For visualising effects the outlier component is a nuisance that pulls expected values toward its own mean and adds a spike of implausibly fast draws; it is also a fixed regularizer rather than a claim about how guesses are distributed, so simulating from it means simulating from something the model does not assert.

```
brms::posterior_epred(m)
modelbased::estimate_means(m, by = "Condition")
marginaleffects::avg_predictions(m, by = "Condition")
```

Use `with_outliers()` for the fitted mixture, and `without_outliers()` to go back. The one case that genuinely wants the mixture is a posterior predictive check, since on untrimmed data the decision-only predictive has no fast spike to match the one in the data:

```
brms::pp_check(with_outliers(m))
```

The same flag can be set up front, with `cogmod_lognormal(predict_outliers = TRUE)`.

The flag is carried on the model rather than passed as an argument for a reason. `brms` sends the `...` of `posterior_predict()` and `posterior_epred()` to `prepare_predictions()`, not down to the family method; `posterior_epred` reaches the family method with `prep` and nothing else. So `posterior_epred(m, predict_outliers = TRUE)` is *silently ignored* rather than erroring, and `insight`, `modelbased` and `marginaleffects` inherit that behaviour. Carrying the flag on the object is what makes it work everywhere.

The `predict_outliers` argument on the methods themselves still works when they are called directly, and overrides the flag.

`log_lik` has no such argument: the likelihood *is* the mixture, and dropping a component from it would not be a different summary of the same model but a different model. One consequence is that `posterior_predict()` and `log_lik()` do not describe the same distribution by default. This also desyncs `loo_pit()`, `loo_predict()` and `bayes_R2()` from `loo()`, not just hand-rolled checks - anything that compares a simulated replicate against the likelihood should be run on `with_outliers()`.

Examples

```
# Simulate 1000 RTs with 2% outliers
rts <- rcogmod_lognormal(1000, mu = -0.7, sigma = 0.5, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, main = "Simulated shifted LogNormal RTs", xlab = "RT (s)")

# Responses faster than ndt have positive density, unlike the unmixed model
dcogmod_lognormal(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_lognormal(0.1, ndt = 0.3, poutlier = 0)

# Density of the outlier component alone
curve(2 * dnorm(x, 0, 0.2), from = 0, to = 3, n = 1000)
```

`rcogmod_logstudent` *Shifted Log-Student-t Model*

Description

Density, random generation, and `brms` custom family for the shifted Log-Student-t distribution - a **robust LogNormal**. A Log-Student-t distributed decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_logstudent()`: Simulates random draws.
- `dcogmod_logstudent()`: Computes the density (likelihood).
- `cogmod_logstudent()`: Creates a `brms::custom_family()`.
- `cogmod_logstudent_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_logstudent(n, mu = -0.7, sigma = 0.4, dof = 5, ndt = 0.2, poutlier = 0)

dcogmod_logstudent(
  x,
  mu = -0.7,
  sigma = 0.4,
  dof = 5,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)

cogmod_logstudent(
  link_mu = "identity",
  link_sigma = "softplus",
  link_dof = "log",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_logstudent_lpdf_expose()

cogmod_logstudent_stanvars()

log_lik_cogmod_logstudent(i, prep)

posterior_predict_cogmod_logstudent(i, prep, predict_outliers = NULL, ...)
```

```
posterior_epred_cogmod_logstudent(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
mu	Location of the Student-t on the log scale. Any real value.
sigma	Scale of the Student-t on the log scale. Must be positive.
dof	Degrees of freedom of the Student-t on the log scale. Must be positive. Smaller is heavier-tailed; `dof -> Inf` is `cogmod_lognormal()`.
ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_dof, link_ndt, link_poutlier	Link functions for the parameters.
predict_outliers	Logical; whether `posterior_predict()` and `posterior_epred()` should include the outlier component. `FALSE` (the default) fixes `poutlier` to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See `with_outliers()`.
i, prep	For brms' functions to run: index of the observation and a `brms` preparation object.
...	Additional arguments.

Details

`log(RT - ndt)` follows a **Student-t** distribution with location `mu`, scale `sigma` and `dof` degrees of freedom. As `dof` grows the Student-t becomes the Normal, so `cogmod_lognormal()` is the `dof -> Inf` limit: this family varies **kurtosis** where `cogmod_loggamma()` varies skew.

`dof` is what `brms::student()` calls `nu`. It is renamed here because `cogmod_lnr()` already spends `nuzero` and `nuone` on drift rates, and because `brms` recognises the name `nu` and supplies opinionated defaults for it; `dof` arrives flat like every other parameter this package defines, so `cogmod_priors()` simply fills it.

`ndt` and `poutlier` mean exactly what they do in `cogmod_lognormal()`, and `with_outliers()`, `without_outliers()`, `p_outlier()` and `cogmod_priors()` all work here too. See `?rcogmod_lognormal` for why `ndt` is expressed directly in seconds rather than as a fraction of the fastest observed response, what the outlier component is for, and why its scale is a constant rather than a `dpar`.

Value

`rcogmod_logstudent()` returns a numeric vector of `n` simulated reaction times, in seconds. `dcogmod_logstudent()` returns the density at each element of `x` - the log density if `log = TRUE` - recycled to the length of the longest argument. `cogmod_logstudent()` returns a `brms::custom_family` object, to put on a `brms::bf()` formula. `cogmod_logstudent_stanvars()` returns a `brms::stanvars` object holding the family's Stan `functions` block, to pass to `brms::brm()`, and `cogmod_logstudent_lpdf_expose()` compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are `brms` post-processing methods, called by `brms` rather than directly: `log_lik_cogmod_logstudent()` returns a numeric vector holding one log-likelihood value per posterior draw for observation `i`, and `posterior_predict_cogmod_logstudent()` a draws x 1 matrix of reaction times simulated for observation `i`. `posterior_epred_cogmod_logstudent()` returns nothing: the decision time has no finite mean, so it errors rather than report one - summarise `posterior_predict()` draws instead.

What the heavy tail is for

The outlier component behind `poutlier` is a half Normal, which by construction cannot explain a *slow* response: its density at 5 s is effectively zero, so a long right tail is the decision family's own business. `cogmod_loggamma()`'s `shape` and `cogmod_invgaussian()`'s `sigmadrift` are two ways of providing one. `dof` is a third, and the most direct: it absorbs slow contaminants into the likelihood rather than into a mixture component. At `dof = 5` the probability of a decision time beyond 5 s is about five orders of magnitude larger than the matching LogNormal's.

Two things to know before using it

The mean does not exist, for any finite `dof`. `E[exp(sigma * T)]` with `T` a Student-t diverges because the t has polynomial tails and `exp()` outruns them - there is no region of the parameter space where this family has an expectation, unlike `cogmod_logweibull()`, whose mean exists below `sigma = 1`. `posterior_epred()` therefore errors rather than returning a number. The **median is exact**: `ndt + exp(mu)`. For anything else, summarise `posterior_predict()` draws.

The density is unbounded at `ndt`. As `RT` approaches `ndt` from above the decision density grows like `1 / (t * |log t|^(dof + 1))`, where a LogNormal decays to zero. The spike is integrable for every `dof > 0`, so the posterior stays proper, but the likelihood has no maximum and the prior on `ndt` is what keeps the sampler off `min(RT)`. This is the same situation as `cogmod_loggamma()` above `sigma * shape = 1`, and the reason `cogmod_priors()` is not optional here.

A Student-t is symmetric **on the log scale**, so a small `dof` fattens both tails rather than only the slow one. At `dof = 2` some 1.5% of the decision distribution falls below 0.05 s, against a LogNormal's 5e-9 - territory `poutlier` also claims, so the two trade off. `cogmod_priors()` centres `dof` at 6 with 95% of its mass between 1.5 and 24, which keeps the fast-side spike under a tenth of a percent while leaving the slow tail worth having.

References

- Lange, K. L., Little, R. J. A., & Taylor, J. M. G. (1989). Robust statistical modeling using the t distribution. *Journal of the American Statistical Association*, *84*(408), 881-896. doi:10.2307/2290063

Examples

```
rts <- rcogmod_logstudent(1000, mu = -0.7, sigma = 0.4, dof = 5,
                          ndt = 0.2, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# A heavier tail than the LogNormal it nests, on the slow side...
dcogmod_logstudent(5, dof = 5, ndt = 0.2)
dcogmod_lognormal(5, ndt = 0.2)

# ...and on the fast side too, which is what `poutlier` also covers.
dcogmod_logstudent(0.21, dof = 5, ndt = 0.2)
dcogmod_lognormal(0.21, ndt = 0.2)
```

---

rcogmod_logweibull *Shifted Log-Weibull Model*

Description

Density, random generation, and `brms` custom family for the shifted Log-Weibull distribution. A Log-Weibull-distributed decision time is shifted by a non-decision time `ndt`, and a fixed proportion `poutlier` of responses is generated by an outlier process instead of by the decision process.

Functions:

- `rcogmod_logweibull()`: Simulates random draws.
- `dcogmod_logweibull()`: Computes the density (likelihood).
- `cogmod_logweibull()`: Creates a `brms::custom_family()`.
- `cogmod_logweibull_stanvars()`: Generates the `stanvars` to pass to `brm()`.

Usage

```
rcogmod_logweibull(n, mu = -0.8, sigma = 0.3, ndt = 0.2, poutlier = 0)

dcogmod_logweibull(
  x,
  mu = -0.8,
  sigma = 0.3,
  ndt = 0.2,
  poutlier = 0,
  log = FALSE
)

cogmod_logweibull(
  link_mu = "identity",
  link_sigma = "softplus",
  link_ndt = "log",
```

```
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_logweibull_lpdf_expose()

cogmod_logweibull_stanvars()

log_lik_cogmod_logweibull(i, prep)

posterior_predict_cogmod_logweibull(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_logweibull(prep, predict_outliers = NULL)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mu`	Location of the Gumbel distribution on the log scale. Any real value.
`sigma`	Scale of the Gumbel distribution on the log scale. Must be positive.
`ndt`	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
`poutlier`	Proportion of responses generated by the outlier process rather than by the decision process. Range: `[0, 1]`. At `poutlier = 0` the distribution reduces to the plain shifted LogNormal.
`x`	Vector of quantiles (observed reaction times).
`log`	Logical; if TRUE, probabilities p are given as log(p).
`link_mu`, `link_sigma`, `link_ndt`, `link_poutlier`	Link functions for the parameters.
`predict_outliers`	Logical; whether `posterior_predict()` and `posterior_epred()` should include the outlier component. `FALSE` (the default) fixes `poutlier` to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See `with_outliers()`.
`i`, `prep`	For brms' functions to run: index of the observation and a `brms` preparation object.
`...`	Additional arguments.

Details

`log(RT - ndt)` follows a **Gumbel** distribution with location `mu` and scale `sigma` - the log-Weibull. The mean decision time is `exp(mu) * gamma(1 - sigma)`, which exists only for `sigma < 1`.

`ndt` and `poutlier` mean exactly what they do in `cogmod_lognormal()`, and `with_outliers()`, `without_outliers()`, `p_outlier()` and `cogmod_priors()` all work here too. See `?rcogmod_lognormal`

for why ndt is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a dpar, and why reaction times have to be in seconds.

posterior_epred() returns Inf where sigma >= 1, because the mean does not exist there. Note this mean is *not* exp(mu + sigma * 0.5772), which is the geometric mean (the exponential of E[log(RT - ndt)]) rather than E[RT - ndt].

Value

rcogmod_logweibull() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_logweibull() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_logweibull() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_logweibull_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_logweibull_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_logweibull() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_logweibull() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_logweibull() returns a draws x observations matrix of expected reaction times, with Inf wherever the mean does not exist.

Examples

```
rts <- rcogmod_logweibull(1000, mu = -0.8, sigma = 0.3, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_logweibull(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_logweibull(0.1, ndt = 0.3, poutlier = 0)
```

---

rcogmod_rdm *Two-Accumulator Racing Diffusion Model (RDM)*

Description

The Racing Diffusion Model (RDM) treats a choice as a race between two diffusion processes, one per response option, each accumulating evidence at its own rate until it reaches a common threshold. The winner determines both the observed reaction time and the choice. The observed RT is that decision time shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the race.

Functions:

- rcogmod_rdm(): Simulates random draws from the RDM.
- dcogmod_rdm(): Computes the density (likelihood).

- `pcogmod_rdm()`: Computes the CDF of the reaction time, marginally over the choice or defectively for one response.
- `qcogmod_rdm()`: Computes the corresponding quantiles.
- `cogmod_rdm()`: Creates a `brms::custom_family()` for use in `brms` models.
- `cogmod_rdm_stanvars()`: Generates the `stanvars` to pass to `brm()`.
- `p_outlier()`: Per-trial posterior probability of being an outlier.

Usage

```
rcogmod_rdm(
  n,
  vzero = 3,
  vone = 2,
  boundary = 0.5,
  bias = 0.2,
  ndt = 0.2,
  poutlier = 0
)

dcogmod_rdm(
  x,
  vzero = 3,
  vone = 2,
  boundary = 0.5,
  bias = 0.2,
  ndt = 0.2,
  response = NULL,
  poutlier = 0,
  log = FALSE
)

pcogmod_rdm(
  q,
  vzero = 3,
  vone = 2,
  boundary = 0.5,
  bias = 0.2,
  ndt = 0.2,
  poutlier = 0,
  response = NULL,
  lower.tail = TRUE,
  log.p = FALSE
)

qcogmod_rdm(
  p,
  vzero = 3,
  vone = 2,
```

```
  boundary = 0.5,
  bias = 0.2,
  ndt = 0.2,
  poutlier = 0,
  response = NULL,
  scale_p = FALSE,
  lower.tail = TRUE,
  log.p = FALSE,
  interval = c(0, 10)
)

cogmod_rdm(
  link_mu = "softplus",
  link_driftone = "softplus",
  link_sigmabias = "softplus",
  link_boundary = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_rdm_lpdf_expose()

cogmod_rdm_stanvars()

log_lik_cogmod_rdm(i, prep)

posterior_predict_cogmod_rdm(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_rdm(prep)
```

Arguments

n	Number of simulated trials. If `length(n) > 1`, the length is taken to be the number required.
vzero, vone	Drift rates of the two accumulators (choice 0 and 1). Must be non-negative; larger means faster. Zero is allowed - such an accumulator is slow, but it still finishes, and can still win. Range: `[0, Inf)`.
boundary	Threshold offset, `boundary = b - bias`, where `b` is the decision threshold and `bias` the maximum starting point. Must be positive.
bias	Maximum starting point. The starting point of each accumulator on each trial is drawn from `Uniform(0, bias)`. Must be non-negative; zero is allowed and gives the plain Wald race, in which both accumulators start at `0` on every trial. Range: `[0, Inf)`. Called `sigmabias` in the `brms` family, to match `cogmod_lba2()`.
ndt	Non-decision time (shift parameter), in seconds. Represents the time taken for processes unrelated to the decision (e.g., encoding, motor response). Must be non-negative. Range: `[0, Inf)`.

poutlier — Proportion of responses generated by the outlier process rather than by the race. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted RDM.

x — The observed reaction time (RT).

response — Accumulator whose finishing time is being scored: 0 for the vzero accumulator, 1 for the vone accumulator. This gives the *defective* density f_response(x) * S_other(x), mixed with the outlier component, which is what a race likelihood needs. The default NULL instead returns the *marginal* density of the RT, ignoring which accumulator won - the sum of the two.

log — Logical; if TRUE, returns the log-density. Default: FALSE.

q — Vector of quantiles (reaction times).

lower.tail — If TRUE (default) return P(RT <= q), otherwise the survival P(RT > q). With a response, both are defective - see Details.

log.p — If TRUE, probabilities are returned on the log scale.

p — Vector of probabilities. With response = NULL these are ordinary probabilities of the marginal RT distribution. With a response they are read off the *defective* CDF unless scale_p = TRUE, so they must be below the probability of that response; anything above it has no quantile and comes back NA with a warning.

scale_p — Logical. If TRUE, p is taken as a fraction of the chosen response's own probability rather than of the whole distribution, so that p = 0.5 is that response's median. This is what a quantile-probability plot wants. Ignored when response is NULL. Default FALSE.

interval — Length-2 numeric giving the initial bracket, in seconds, for the root search. The upper end is doubled until it covers the requested probability, so this only affects speed.

link_mu, link_driftone — Link functions for the two drift rates. mu is vzero: brms requires the first distributional parameter of a custom family to be called mu, so that is the name the formula and this argument use, and the drift of accumulator 0 is what it means.

link_sigmabias, link_boundary — Link functions for the start-point range and the threshold offset.

link_ndt, link_poutlier — Link functions for the non-decision time and the outlier rate.

predict_outliers — Logical; whether posterior_predict() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the race alone; the likelihood is always the full mixture either way. On the prediction method itself the default is NULL, which defers to the flag carried on the model - see with_outliers() to change it after fitting. See Details.

i, prep — For brms' functions to run: index of the observation and a brms preparation object.

... — Additional arguments.

Details

pcogmod_rdm() with response = NULL (the default) describes the RT of the trial as a whole - whichever accumulator wins, and whether or not the trial came from the outlier component - since P(min(T0, T1) > q) = S0(q) * S1(q). That is a closed form and is exact.

With a response, it returns the **defective** CDF P(RT <= q, choice = response), which is what a defective-CDF or quantile-probability plot needs. It does not reach one: its limit is the probability of that response, which pcogmod_rdm(Inf, response = k) gives. There is no closed form for it, so it is obtained by quadrature over the defective density: accurate to about 1e-8 rather than to machine precision, and about ten times slower per element (roughly 3 ms against 0.3 ms), since the marginal is a vectorised closed form and this is a loop. lower.tail = FALSE integrates the upper side directly rather than subtracting, so the defective survival stays accurate into the tail.

qcogmod_rdm() inverts pcogmod_rdm() by root-finding, and so inherits its quadrature error where a response is given. It is the natural way to get the RT quantiles of each response for a quantile-probability plot: ask for p = c(0.1, 0.3, 0.5, 0.7, 0.9) with scale_p = TRUE, once per response.

Value

rcogmod_rdm() returns a data frame with n rows and two columns:

rt	The simulated reaction time.
response	The winning accumulator, coded 0 for vzero and 1 for vone, matching the dec() coding used by the brms families.

dcogmod_rdm() returns the density at each element of x - the log density if log = TRUE - pcogmod_rdm() the cumulative probability at each element of q, and qcogmod_rdm() the quantile at each element of p, in seconds. With a response the latter two are defective, i.e. scaled to that response's own probability rather than to one. All are numeric vectors, recycled to the length of the longest argument. cogmod_rdm() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_rdm_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_rdm_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_rdm() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_rdm() a draws x 2 matrix of reaction times and choices simulated for observation i. posterior_epred_cogmod_rdm() returns nothing: the expected reaction time of a race has no closed form, so it errors rather than report one - summarise posterior_predict() draws instead.

Parameterization

Each accumulator is a diffusion with drift rate v and unit diffusion coefficient, starting from a point z ~ Uniform(0, bias) drawn afresh on every trial and finishing when it reaches the threshold b = boundary + bias. The distance it has to cover is therefore b - z = boundary + bias - z, which is what makes boundary the threshold *offset*: the distance from the highest possible starting point to the threshold. Its first passage time is Wald (inverse Gaussian) with the start point integrated out. The observed reaction time is ndt + min(T_0, T_1) and the observed choice is whichever accumulator got there first.

This is the B parameterization of DMC and EMC2, where b = B + A with A the start-point range (bias here). It is used for a reason rather than for taste: the threshold has to sit above the highest possible starting point, and writing the *offset* makes b > A hold automatically for any positive value. The alternative - estimating the absolute threshold, as rtdists does - needs an order constraint between two estimated parameters, which has to hold in every cell of the design once either of them carries a predictor. The cost is that boundary alone is not the quantity to read off a fitted model; boundary + bias is.

A drift rate of **exactly zero is allowed**, and is not the same as an accumulator that never responds: driftless Brownian motion still reaches any positive level with probability one, so a zero-drift accumulator is slow but still finishes, and can still win the race.

A start-point range of **exactly zero is allowed** too, and is a model rather than a degenerate parameter: both accumulators then start at 0 on every trial and the race is between two plain Walds - equation 2 of Tillman et al. (2020), which is the limit the density already takes. cogmod_lba1() and cogmod_lba2() have always allowed it. Note that this does not make the sigmabias direction any better identified - see Fitting below.

ndt is expressed **directly, in seconds** (through a log link in the brms family). Nothing about it is taken from the data: it is not bounded by the fastest observed response, so a non-decision time that varies by condition or by participant can exceed the sample minimum wherever the data support it.

This replaces the earlier tau / minrt pair, in which ndt = tau * minrt with minrt set to the fastest observed RT. That capped the non-decision time at an order statistic of the sample, so any condition or participant whose true ndt exceeded the fastest observed response was inexpressible, and the misfit surfaced as spurious effects on the race parameters.

The outlier component

A shifted distribution assigns exactly zero density to any response faster than ndt, which puts a hard boundary in the likelihood at the fastest observed RT. Mixing in a component with support over the whole positive line removes it: every response keeps positive density whatever ndt is, so the boundary becomes a finite cost rather than a wall and the log-density stays smooth and differentiable. That is what makes the direct parameterization of ndt workable without taking a bound from the data.

Because this model produces a **choice as well as a time**, the contaminant has to produce both. It is a guess: the choice is uniform over the two options, and the RT is a **half Normal** with scale 0.2 seconds.

$$f(t,k) = p\frac{1}{K}g(t) + (1-p)f_k(t-ndt)$$

The 1 / K is what keeps the total summing to one over the response options; without it it would come to 1 + poutlier. The half-t is used for the timing because it is **flat at the origin** (zero derivative), so the very fastest responses - the ones least plausibly decisions - are not starved of density, and because its tails are heavy enough to cover the whole plausible RT range.

poutlier is a *rate*, not a classification: the model never labels individual trials, it estimates what share of them came from elsewhere. Use p_outlier() for per-trial posterior probabilities.

Reaction times must be in seconds

The outlier component's scale is a constant in seconds, and so are the priors `cogmod_priors()` supplies. There is no argument for changing the unit: the `minrt` argument that used to rescale the component was removed in 0.2.1. Millisecond data fails silently rather than loudly - the outlier component contributes nothing and the min-RT boundary comes back. See the corresponding section of `cogmod_lognormal()` for the full account, which applies unchanged here.

Fitting

```
f <- brms::bf(RT | dec(Error) ~ Condition, driftone ~ Condition,
              sigmabias ~ 1, boundary ~ 1, ndt ~ 1, poutlier ~ 1,
              family = cogmod_rdm())
brms::brm(f, data = df,
          prior    = cogmod_priors(f, df),
          init     = cogmod_inits(f, df),
          stanvars = cogmod_stanvars(f))
```

The `brms` family names the drift of the first accumulator `mu` (as `brms` requires) and that of the second `driftone`, and calls the start-point range `sigmabias` to match `cogmod_lba2()`, where it denotes the same quantity. Note that this is *not* the same thing as `bias` in `cogmod_ddm()`, which is a relative starting point in `[0, 1]`. Both drifts use a `softplus` link with a lower bound of zero, following `cogmod_invgaussian()`: a Wald drift must be non-negative for the accumulator to be a proper first passage time.

Use `cogmod_inits()` rather than `init = 0`. `brms` initialises on the unconstrained scale, so `init = 0` puts `ndt` at `exp(0) = 1` second - above nearly every sub-second RT, which leaves every response attributed to the outlier component and the race parameters with no gradient at all.

`cogmod_priors()` is not a convenience here either. Beyond `ndt` and `poutlier`, `sigmabias` and `boundary` are only weakly identified from each other, because they enter the threshold only through the sum `b = boundary + sigmabias` and trade off almost freely: on simulated data with 4000 trials the profile log-likelihood varies by only about 3 units as `sigmabias` ranges from 0 to half the threshold, while `boundary` slides to compensate. With flat priors the sampler tends to wander down the `sigmabias -> 0` ridge (the plain Wald race) and produce divergent transitions, and a `softplus` link reaches zero only at minus infinity - a flat prior over an unbounded flat region, which is an improper posterior. That the endpoint is now a legal parameter value does not help: the link never reaches it, so the flat direction is as long as it ever was. `cogmod_priors()` fences both off, exactly as it does for `cogmod_lba1()`, which shares this parameterisation.

The sum `boundary + sigmabias` is well identified either way, so it is the more trustworthy quantity to interpret and to compare across conditions. The same caveat applies to `cogmod_lba2()`.

Predictions exclude the outlier component

`posterior_predict()` describes the **race alone** by default, as if `poutlier` were zero, because the outlier component is a fixed regularizer rather than a claim about how guesses are distributed. Use `with_outliers()` for the fitted mixture - chiefly for `brms::pp_check()` - and `without_outliers()` to go back. `log_lik()` is always the full mixture.

`posterior_epred()` is not provided: for a race model the expectation needs numerical integration per draw and per observation, and users are better off summarising `posterior_predict()` draws.

References

- Michael, J. R., Schucany, W. R., & Haas, R. W. (1976). Generating Random Variates Using Transformations with Multiple Roots. *The American Statistician*, *30*(2), 88-90. doi:10.2307/2683801
- Tillman, G., Van Zandt, T., & Logan, G. D. (2020). Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making. *Psychonomic Bulletin & Review*, *27*, 911-936. doi:10.3758/s13423020017196
- Folks, J. L., & Chhikara, R. S. (1978). The inverse Gaussian distribution and its statistical application-a review. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *40*(3), 263-275.

See Also

`rcogmod_invgaussian()`, `rcogmod_lnr()`

Examples

```
# Simulate data, with 2% of trials from the outlier process
data <- rcogmod_rdm(1000,
  vzero = 2.5, vone = 1.6, boundary = 0.5, bias = 0.2,
  ndt = 0.2, poutlier = 0.02
)
head(data)

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_rdm(0.1, ndt = 0.2, response = 0, poutlier = 0.02)
dcogmod_rdm(0.1, ndt = 0.2, response = 0, poutlier = 0)

# Defective CDF of one response: at q = Inf it is that response's probability
pcogmod_rdm(c(0.4, 0.6, Inf), vzero = 2.5, vone = 1.6, response = 0)

# The RT quantiles of each response, for a quantile-probability plot
sapply(0:1, function(k) {
  qcogmod_rdm(c(0.1, 0.3, 0.5, 0.7, 0.9),
    vzero = 2.5, vone = 1.6, response = k, scale_p = TRUE
  )
})


# Exposing the Stan function needs cmdstanr and a CmdStan toolchain,
# which live outside CRAN - see the package website to install them.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  lpdf <- cogmod_rdm_lpdf_expose()
  lpdf(
    Y = 0.5, mu = 2, driftone = 1.5, sigmabias = 0.2, boundary = 0.5,
    ndt = 0.2, poutlier = 0.02, dec = 0
  )
}
```

rcogmod_weibull *Shifted Weibull Model*

Description

Density, random generation, and brms custom family for the shifted Weibull distribution. A Weibull-distributed decision time is shifted by a non-decision time ndt, and a fixed proportion poutlier of responses is generated by an outlier process instead of by the decision process.

Functions:

- rcogmod_weibull(): Simulates random draws.
- dcogmod_weibull(): Computes the density (likelihood).
- cogmod_weibull(): Creates a brms::custom_family().
- cogmod_weibull_stanvars(): Generates the stanvars to pass to brm().

Usage

```
rcogmod_weibull(n, mu = 2, sigma = 0.5, ndt = 0.2, poutlier = 0)

dcogmod_weibull(x, mu = 2, sigma = 0.5, ndt = 0.2, poutlier = 0, log = FALSE)

cogmod_weibull(
  link_mu = "softplus",
  link_sigma = "softplus",
  link_ndt = "log",
  link_poutlier = "logit",
  predict_outliers = FALSE
)

cogmod_weibull_lpdf_expose()

cogmod_weibull_stanvars()

log_lik_cogmod_weibull(i, prep)

posterior_predict_cogmod_weibull(i, prep, predict_outliers = NULL, ...)

posterior_epred_cogmod_weibull(prep, predict_outliers = NULL)
```

Arguments

n	Number of observations. If length(n) > 1, the length is taken to be the number required.
mu	Shape of the Weibull decision time. Must be positive.
sigma	Scale of the Weibull decision time. Must be positive.

ndt	Non-decision time (shift parameter), in seconds. Must be non-negative. Represents time for processes such as stimulus encoding and response execution. Range: [0, Inf).
poutlier	Proportion of responses generated by the outlier process rather than by the decision process. Range: [0, 1]. At poutlier = 0 the distribution reduces to the plain shifted LogNormal.
x	Vector of quantiles (observed reaction times).
log	Logical; if TRUE, probabilities p are given as log(p).
link_mu, link_sigma, link_ndt, link_poutlier	Link functions for the parameters.
predict_outliers	Logical; whether posterior_predict() and posterior_epred() should include the outlier component. FALSE (the default) fixes poutlier to zero for prediction, so predictions describe the decision process alone; the likelihood is always the full mixture either way. See with_outliers().
i, prep	For brms' functions to run: index of the observation and a brms preparation object.
...	Additional arguments.

Details

mu is the **shape** and sigma the **scale** of the Weibull decision time, whose mean is sigma * gamma(1 + 1 / mu).

ndt and poutlier mean exactly what they do in cogmod_lognormal(), and with_outliers(), without_outliers(), p_outlier() and cogmod_priors() all work here too. See ?rcogmod_lognormal for why ndt is expressed directly in seconds rather than as a fraction of the fastest observed response, what the half Student-t outlier component is for, and why the outlier component's scale is a constant rather than a dpar, and why reaction times have to be in seconds.

Value

rcogmod_weibull() returns a numeric vector of n simulated reaction times, in seconds. dcogmod_weibull() returns the density at each element of x - the log density if log = TRUE - recycled to the length of the longest argument. cogmod_weibull() returns a brms::custom_family object, to put on a brms::bf() formula. cogmod_weibull_stanvars() returns a brms::stanvars object holding the family's Stan functions block, to pass to brms::brm(), and cogmod_weibull_lpdf_expose() compiles that Stan code and returns it as an R function, for checking the density outside of a model. The remaining functions are brms post-processing methods, called by brms rather than directly: log_lik_cogmod_weibull() returns a numeric vector holding one log-likelihood value per posterior draw for observation i, and posterior_predict_cogmod_weibull() a draws x 1 matrix of reaction times simulated for observation i. posterior_epred_cogmod_weibull() returns a draws x observations matrix of expected reaction times.

The shape governs how well this samples

Near the shift the Weibull density behaves like (y - ndt)^(mu - 1), and that exponent decides how the mixture behaves as ndt passes an observation. Three regimes, in order of severity:

- `mu < 1`: the density is **unbounded at** `ndt`, so the likelihood is unbounded as `ndt` approaches the fastest response. The outlier component adds density rather than capping it, so it cannot repair that.
- `mu < 2`: the density is bounded, but the derivative of the log-likelihood with respect to `ndt` behaves like `(y - ndt)^(mu - 2)` and so is **unbounded at every observation**. The posterior is proper - `poutlier` keeps it so - but the gradient spikes wherever `ndt` sits close to a response, which is exactly where the data put it.
- `mu > 2`: bounded gradient. `mu > 3` additionally bounds the curvature.

The middle regime is the one to watch, because nothing warns about it. On the 4285-trial lexical-decision data in `vignette("rt_models")` the shape comes out at 1.4, `ndt` lands at 0.40 s inside the dense left edge of the data, and the sampler's step size collapses to 0.005 against 0.19 for `cogmod_lognormal()` on the same data: mean treedepth 8.1 against 3.9, which is 19x the gradient evaluations and 19x the wall time, with `Rhat` 1.18 on `ndt`. The density itself is cheap; **all** of the cost is geometry.

What does not help:

All of the obvious remedies were tried on that fit and measured. None of them works, and two make it worse, so they are recorded here rather than left for the next person to rediscover.

A prior on the shape. `normal(2.4, 0.4)` on the softplus scale puts 95% of its mass above `mu = 1.9`. It moved the posterior shape by 0.01, because the likelihood prefers the low-shape corner by around 100 log units and the prior contributes 5.

A narrow prior on `ndt`. This looks like the obvious fix - keep the shift below the data and the singular region is never visited - and it fails for an instructive reason. `normal(-1.25, 0.05)`, centred at 0.287 s with 95% of its mass below the fastest bulk response, left the posterior at 0.396 s: **6.5 prior SDs away**, essentially where it was without any prior at all. The `ndt` likelihood has a posterior SD of 0.003, so it is some fifteen times sharper than that prior; nothing weaker than fixing `ndt` outright competes with it. What the attempt did achieve was 4% divergent transitions against 0.5%, 16% of iterations at maximum treedepth against 7%, `Rhat` 1.43 against 1.18, and a slightly worse `loo`.

Fixing `ndt` at the fastest observed response. This does remove the problem, by removing the parameter - but it reinstates exactly the min-RT bound this parameterization exists to get rid of, and it is unsound wherever the outlier component is doing its job. On the data above the fastest response is 71 ms, which is not a decision; the mixture is there precisely so that an order statistic of the sample is not treated as a bound. See `cogmod_lognormal()`.

Note also what is *not* wrong: `ndt` and the shape are jointly identified, and sharply so - the posterior SD on `ndt` is 3 ms. This is not a case of two parameters trading off with nothing to separate them, so pinning one of them is not the missing ingredient. The sharpness simply sits on a ridge that is not smooth.

What to do instead:

Treat a fitted shape below 2 as the diagnostic it is, and use `cogmod_loggamma()`, which nests this family at `shape = 1` and lets the data choose the shape rather than having the family fix it. On the data above it samples in a third of the time with no divergences.

The slow sampling and the poor fit are the same fact, not two problems. Across the ten families fitted in `vignette("rt_models")` the Weibull comes **last** by `loo`, 196 elpd (SE 21) behind `cogmod_loggamma()` and 95 behind the next worst. What the sampler struggles with is the model contorting itself - pushing the shift up into the data, pulling the shape toward 1 - to represent a left

edge it cannot otherwise reach. That does not make the Weibull useless for reaction times in general; where the shape comes out above 2 the family is perfectly well behaved, as cogmod_gamma() is on these same data at a shape of 2.2. It does mean a shape below 2 should be read as the model telling you to use a different one.

Under the older ndt = tau * min(RT) parameterization the problem was hidden rather than absent: the logit Jacobian vanished as tau approached 1, which damped exactly this gradient.

Starting values

Do **not** fit this with init = 0: it puts ndt at exp(0) = 1 second and the shape at softplus(0) = 0.69, inside the mu < 1 regime above, and no single scalar avoids both. Use cogmod_inits(), which sets them separately:

```
brms::brm(f, data = df, prior = cogmod_priors(f, df),
          stanvars = cogmod_stanvars(f), init = cogmod_inits(f, df))
```

See cogmod_inits() for why, and ?rcogmod_gamma for what it costs when ignored.

Examples

```
rts <- rcogmod_weibull(1000, mu = 2, sigma = 0.5, ndt = 0.3, poutlier = 0.02)
hist(rts, breaks = 100, xlab = "RT (s)")

# Responses faster than ndt keep positive density, unlike the unmixed model
dcogmod_weibull(0.1, ndt = 0.3, poutlier = 0.02)
dcogmod_weibull(0.1, ndt = 0.3, poutlier = 0)
```

---

with_outliers *Include or exclude the outlier component in predictions*

Description

Switches the predict_outliers flag on a model fitted with cogmod_lognormal() or cogmod_loggamma(), controlling whether posterior_predict() and posterior_epred() describe the fitted mixture or the decision process alone.

Predictions **exclude** the outlier component by default, because for almost every downstream use it is a nuisance: it pulls expected values toward its own mean (0.16 s) and adds a spike of implausibly fast draws to posterior predictive samples. It is also a deliberately fixed regularizer rather than a claim about how guesses are distributed, so simulating from it means simulating from something the model does not assert.

with_outliers() restores the mixture. The main reason to want it is brms::pp_check(): on untrimmed data the decision-only predictive has no fast spike to match the one in the data, which reads as misfit. Use pp_check(with_outliers(m)) for a like-for-like check.

The flag is stored **on the model** rather than passed as an argument, because brms and the packages built on it (insight, modelbased, marginaleffects, emmeans) do not forward extra arguments

down to a custom family's prediction methods - `posterior_epred()` reaches the family method with `prep` and nothing else. Carrying it on the object is what makes it work through all of them. The same flag can be set up front with `cogmod_lognormal(predict_outliers = TRUE)`.

`log_lik()` is unaffected and has no equivalent switch: the likelihood *is* the mixture, and dropping a component from it would not be a different summary of the same model but a different model. One consequence worth knowing is that `posterior_predict()` and `log_lik()` do not describe the same distribution by default. This also desyncs `loo_pit()`, `loo_predict()` and `bayes_R2()` from `loo()`, not just hand-rolled checks - anything that compares a simulated replicate against the likelihood should be run on `with_outliers()`.

Usage

```
with_outliers(object)

without_outliers(object)
```

Arguments

`object`	A `brmsfit` fitted with `cogmod_lognormal()`, `cogmod_loggamma()` or any other family built on the outlier mixture - see the *Supported families* section of `cogmod_priors()` for the full list, which includes the choice-and-RT families such as `cogmod_lnr()`.

Value

The model, with the flag set. The fit itself is untouched - only how predictions are summarised changes.

Examples

```
# Fitting needs cmdstanr, which lives outside CRAN - see the package website.
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    !is.null(cmdstanr::cmdstan_version(error_on_NA = FALSE))) {
  df <- data.frame(
    RT = rcogmod_lognormal(200, ndt = 0.3, poutlier = 0.05),
    Condition = rep(c("A", "B"), each = 100)
  )
  f <- brms::bf(RT ~ Condition, ndt ~ 1, poutlier ~ 1,
    family = cogmod_lognormal()
  )
  m <- brms::brm(f,
    data = df, stanvars = cogmod_stanvars(f),
    prior = cogmod_priors(f, df), init = cogmod_inits(f, df),
    backend = "cmdstanr", chains = 1, iter = 500, refresh = 0
  )

  # the decision process alone - the default, everywhere downstream
  head(brms::posterior_epred(m)[, 1])

  # the fitted mixture, e.g. for a like-for-like predictive check
  m2 <- with_outliers(m)
  head(brms::posterior_epred(m2)[, 1])
```

```
  without_outliers(m2) # back to the default
}
```

Index