Package: GEVStableGarch (via r-universe)

November 3, 2026

Type Package

Title ARMA-GARCH/APARCH Models with Stable, GEV and GAt Innovations

Version 2.0.0

Date 2026-09-28

Description Simulation, estimation and forecasting of autoregressive moving average (ARMA) models with generalized autoregressive conditional heteroskedasticity (GARCH) or asymmetric power ARCH (APARCH, with ARCH the autoregressive conditional heteroskedasticity model) errors, with stable (S0 parametrization), generalized extreme value (GEV) and generalized asymmetric t (GAt) innovations. The innovation distribution is a pluggable object, so user defined densities get the same fitting, simulation, stationarity and forecasting machinery.

License GPL (>= 2)

URL https://github.com/thiagodoregosousa/GEVStableGarch

BugReports https://github.com/thiagodoregosousa/GEVStableGarch/issues

Depends R (>= 4.1.0)

Imports libstable4u, Rsolnp, stats, utils

Suggests bgev, fGarch, ggplot2, knitr, rmarkdown, stabledist, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 8.0.0

NeedsCompilation no

Author Thiago do Rego Sousa [aut, cre], Cira Otiniano [ctb], Silvia Lopes [ctb], Diethelm Wuertz [ctb, cph] (filter and model structure adapted from fGarch)

Maintainer Thiago do Rego Sousa <thiagodoregosousa@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-03 21:07:48 UTC

RemoteUrl https://github.com/cran/GEVStableGarch

RemoteRef HEAD

RemoteSha d7c302ab224cd81652c922c61638c9103dae3b71

Contents

GAT 2
gs_aparch_moment 4
gs_bootstrap 5
gs_check_dist 6
gs_dist 6
gs_fit 8
gs_fit-methods 10
gs_gat 11
gs_gev 12
gs_model 13
gs_sim 14
gs_spec 14
gs_stable 15
gs_stationarity 16
predict.gs_fit 17

Index **19**

GAT *Generalized Asymmetric t (GAT) Distribution*

Description

Functions to compute the density, distribution function, quantile function, and to generate random variates for the Generalized Asymmetric t (GAT) distribution.

Usage

```
dgat(x, mean = 0, sd = 1, nu = 2, d = 3, xi = 1, log = FALSE)

pgat(q, mean = 0, sd = 1, nu = 2, d = 3, xi = 1)

qgat(p, mean = 0, sd = 1, nu = 2, d = 3, xi = 1)

rgat(n, mean = 0, sd = 1, nu = 2, d = 3, xi = 1)
```

Arguments

x	Numeric vector of values for calculating density.
mean	Location parameter.
sd	Scale parameter (must be > 0).
nu	Tail (shape) parameter (must be > 0).
d	Second shape parameter (must be > 0).
xi	Asymmetry parameter (must be > 0; $\xi = 1$ gives symmetry).
log	Logical; if TRUE, densities are returned on the log scale.
q	Numeric vector of quantiles.
p	Numeric vector of probabilities.
n	Number of observations for random generation.

Details

This distribution corresponds to the t3 distribution described in Paolella (1997) and Mittnik and Paolella (2000). The GAT family includes the Student's t, Laplace, Cauchy, and normal distributions as special cases. In particular, as $\nu \to \infty$ the distribution approaches normality.

Value

dgat	density values
pgat	distribution function values
qgat	quantile function values
rgat	random variates

Author(s)

Thiago do Rego Sousa

References

Mittnik, S., Paolella, M. S. (2000). Prediction of Financial Downside-Risk with Heavy-Tailed Conditional Distributions.

Paolella, M. (1997). Tail Estimation and Conditional Modeling of Heteroskedastic Time-Series. PhD Thesis, Institute of Statistics and Econometrics, Christian Albrechts University of Kiel.

Bertocchi, M., Giacometti, R., Ortobelli, S., & Rachev, S. T. (2005). The impact of different distributional hypothesis on returns in asset allocation. *Finance Letters*, 3(1), 17-27.

Examples

```
oldpar <- par(mfrow = c(2, 2))
set.seed(1000)
r <- rgat(n = 1000)
plot(r, type = "l", main = "GAt Random Values")
```

```
hist(r, probability = TRUE, border = "white")
x <- seq(min(r), max(r), length = 201)
lines(x, dgat(x), lwd = 2)

plot(sort(r), (1:1000)/1000, main = "Probability", ylab = "Probability")
lines(x, pgat(x), lwd = 2)
par(oldpar)

round(qgat(pgat(q = seq(-10, 10, by = 0.5))), 6)
```

`gs_aparch_moment` *APARCH moment of the innovation distribution*

Description

Computes $\kappa = E(|z| - \gamma z)^\delta$ for standardized innovations z, the quantity that enters the APARCH stationarity condition and the multi step scale forecast. A closed form is used when the distribution provides one, otherwise numerical integration.

Usage

```
gs_aparch_moment(dist, par, gamma, delta)
```

Arguments

`dist`	A `gs_dist` object.
`par`	Distribution parameters, named by `dist$par_names`.
`gamma`	Asymmetry parameter in (-1, 1).
`delta`	Power, positive.

Value

κ, or `Inf` when δ is not below `dist$max_power(par)`.

References

Ding, Z., Granger, C. W. J. and Engle, R. F. (1993). A long memory property of stock market returns and a new model. Journal of Empirical Finance, 1, 83-106.

Mittnik, S., Paolella, M. S. and Rachev, S. T. (2002). Stationarity of stable power-GARCH processes. Journal of Econometrics, 106, 97-107.

Examples

```
gs_aparch_moment(gs_stable(), c(stable_alpha = 1.7, stable_beta = 0),
                 gamma = 0.3, delta = 1)
```

gs_bootstrap *Parametric bootstrap standard errors and confidence intervals*

Description

Simulates n_boot series of the same length from the fitted model, refits each one, and summarizes the resulting sampling distribution of the estimates. This is the recommended alternative to the Hessian standard errors of gs_fit() when the asymptotic normal approximation is unreliable: near a boundary (stable tail index near 2, GEV shape near ± 0.5, APARCH asymmetry near ± 1) or, for GEV innovations, because the support depends on the shape parameter and the usual regularity conditions do not hold (Efron, 1979).

Usage

```
gs_bootstrap(fit, n_boot = 199, level = 0.95, seed = NULL, ...)
```

Arguments

fit	A gs_fit object.
n_boot	Number of bootstrap replications.
level	Confidence level of the percentile intervals.
seed	Optional seed for reproducibility.
...	Passed to gs_fit() (for example algorithm).

Details

Each replication is refit with hessian = FALSE; replications whose fit fails are dropped and counted in n_fail.

Value

A list with se (named vector of bootstrap standard errors), ci (a matrix with lower and upper percentile limits), replicates (the n_boot by number-of-parameters matrix of estimates, failed fits as NA rows), n_fail and level.

References

Efron, B. (1979). Bootstrap methods: another look at the jackknife. The Annals of Statistics, 7(1), 1-26.

See Also

gs_fit()

Examples

```
spec <- gs_spec(garch = c(1, 1), dist = gs_gev())
model <- gs_model(spec, mu = 0, omega = 0.1, alpha = 0.1, beta = 0.8,
                  dist_par = c(xi = 0.2))
x <- gs_sim(model, n = 800, seed = 1)$y
fit <- gs_fit(x, spec)
gs_bootstrap(fit, n_boot = 99, seed = 1)
```

`gs_check_dist` *Self check of an innovation distribution*

Description

Verifies that the pieces of a `gs_dist()` object are consistent at a parameter value: names and bounds, finite and vectorized log density, density integrating to one, `quantile` inverting `cdf`, `cdf` matching the density, `random` matching `cdf` (Kolmogorov-Smirnov), moment conditions (`max_power` above `default_delta`, `mean` and `aparch_moment` against numerical integration). It stops with an error naming the failing field.

Usage

```
gs_check_dist(dist, par = dist$start, tol = 0.001)
```

Arguments

`dist`	A `gs_dist` object.
`par`	Parameter value where the checks are done (default `dist$start`).
`tol`	Tolerance for the numerical comparisons.

Value

`dist`, invisibly.

`gs_dist` *Innovation distribution for ARMA-GARCH/APARCH models*

Description

Builds the object that describes the innovation distribution z_t of the model. The built in families `gs_stable()`, `gs_gev()` and `gs_gat()` are created with this constructor, and a user defined distribution gets exactly the same fitting, simulation, stationarity and forecasting machinery.

Usage

```
gs_dist(
  name,
  log_density,
  random,
  cdf,
  quantile,
  par_names,
  lower,
  upper,
  start,
  default_delta,
  max_power,
  mean,
  valid = NULL,
  aparch_moment = NULL,
  start_fun = NULL,
  check = TRUE
)
```

Arguments

name	Character, name of the distribution.
log_density	function(z, par) returning $\log f(z)$, vectorized in z, and -Inf outside the support.
random	function(n, par) returning n draws.
cdf	function(q, par), distribution function.
quantile	function(p, par), quantile function.
par_names	Character vector with the parameter names. They must not clash with model parameter names (mu, ar1, omega, alpha1, ...).
lower, upper, start	Numeric vectors named by par_names: box bounds used in estimation and the default start value (strictly inside).
default_delta	Power δ used when the model is a GARCH (not APARCH) model, e.g. 2 for a variance recursion, 1 for a scale recursion.
max_power	function(par) returning the supremum of the powers δ with $E\|z\|^\delta < \infty$ (Inf if all moments exist).
mean	function(par) returning $E[z]$; it must return NA explicitly where the mean does not exist.
valid	Optional function(par) returning TRUE/FALSE for constraints that are not box bounds.
aparch_moment	Optional function(gamma, delta, par) returning $E(\|z\| - \gamma z)^\delta$ in closed form. It may return NA where no closed form is available; numerical integration is then used.
start_fun	Optional function(z) returning data driven start values from standardized data.
check	Logical, run gs_check_dist() at start.

Details

All functions work on the standardized innovation (location 0, scale 1). `z` is a numeric vector and `par` a numeric vector named by `par_names`.

Every field is required except `valid`, `aparch_moment` and `start_fun`, which have well defined fallbacks. In particular `max_power` and `mean` have no defaults: the user must state when moments exist, so that fitting never fails silently. Unless `check = FALSE`, `gs_check_dist()` runs a self check at `start` and stops with an error naming the failing field.

Value

An object of class `gs_dist`.

See Also

`gs_check_dist()`, `gs_stable()`, `gs_gev()`, `gs_gat()`

Examples

```
# Normal innovations written as a user defined distribution
norm <- gs_dist(
  name = "norm",
  log_density = function(z, par) dnorm(z, log = TRUE),
  random = function(n, par) rnorm(n),
  cdf = function(q, par) pnorm(q),
  quantile = function(p, par) qnorm(p),
  par_names = character(0),
  lower = numeric(0), upper = numeric(0), start = numeric(0),
  default_delta = 2,
  max_power = function(par) Inf,
  mean = function(par) 0)
norm
```

---

gs_fit *Fit an ARMA-GARCH/APARCH model by maximum likelihood*

Description

Estimates the model of `gs_spec()` by conditional maximum likelihood. The data is rescaled internally to unit standard deviation (as in fGarch) and the estimates are transformed back. Parameters move in plain intervals; stationarity is not imposed but reported after fitting (see `gs_stationarity()`), with a warning when the persistence is 1 or more.

Usage

```
gs_fit(
  data,
  spec,
  algorithm = c("sqp", "nlminb"),
```

```
  start = NULL,
  hessian = TRUE,
  restarts = 0L,
  control = list()
)
```

Arguments

data	Numeric vector, the time series (for example log returns).
spec	A gs_spec object.
algorithm	"sqp" (sequential quadratic programming via Rsolnp::solnp(), Galanos and Ye 2025) or "nlminb".
start	Optional named vector of start values on the original scale, for example coef(previous_fit) in rolling windows. Missing names are filled with the default start values.
hessian	Logical, compute standard errors. Set FALSE to save time when only point estimates are needed.
restarts	Integer, number of additional optimizer runs from perturbed start values; the fit with the highest log likelihood is kept. A few restarts (for example 3 to 5) make the estimate robust to the local optima of multimodal surfaces, such as ARMA-GARCH models with heavy-tailed innovations, at a proportional increase in time. The default 0 keeps the single run from the data-driven start.
control	List passed to the optimizer.

Details

Standard errors come from a numerical Hessian (central differences with steps adapted to the curvature). When a parameter sits on a bound, or a differentiation step leaves the parameter space, the affected standard errors are NA and a warning explains why.

These Hessian standard errors are asymptotic and rely on the usual regularity conditions. They become unreliable *near* a boundary of the parameter space, even when not exactly on it: the stable tail index approaching 2 (where the skewness is no longer identified), the GEV shape approaching ± 0.5, or an APARCH asymmetry approaching ± 1. The GEV support depends on its shape parameter, so for GEV innovations the usual maximum likelihood confidence intervals are not guaranteed valid; prefer the parametric bootstrap of gs_bootstrap() there and in the near-boundary cases above. The fit records convergence diagnostics in $diagnostics (scaled gradient, Hessian reciprocal condition number and definiteness, optimizer iterations), which flag these situations.

Value

An object of class gs_fit with methods coef(), vcov(), logLik(), residuals(), sigma(), fitted(), predict() and print().

References

Galanos, A. and Ye, Y. (2025). Rsolnp: General Non-Linear Optimization. R package version 2.0.1. https://CRAN.R-project.org/package=Rsolnp

Examples

```
if (requireNamespace("fGarch", quietly = TRUE)) {
  data("dem2gbp", package = "fGarch")
  spec <- gs_spec(arma = c(1, 0), garch = c(1, 1), dist = gs_stable())
  fit <- gs_fit(dem2gbp[, 1], spec)
  fit
}
```

`gs_fit-methods` *Methods for fitted models*

Description

Accessors for `gs_fit` objects. `residuals()` returns, by `type`: `"raw"` the residuals e_t, `"standardized"` the innovations e_t/σ_t, `"pit"` their probability integral transform $F(e_t/\sigma_t)$ under the fitted innovation distribution (uniform under a correct model; the input of copula models and of KS diagnostics). `sigma()` returns the conditional scale σ_t (not a standard deviation when the innovation variance is infinite), `fitted()` the conditional location $x_t - e_t$.

Usage

```
## S3 method for class 'gs_fit'
coef(object, ...)

## S3 method for class 'gs_fit'
vcov(object, ...)

## S3 method for class 'gs_fit'
logLik(object, ...)

## S3 method for class 'gs_fit'
nobs(object, ...)

## S3 method for class 'gs_fit'
residuals(object, type = c("raw", "standardized", "pit"), ...)

## S3 method for class 'gs_fit'
sigma(object, ...)

## S3 method for class 'gs_fit'
fitted(object, ...)

## S3 method for class 'gs_fit'
print(x, ...)

## S3 method for class 'gs_fit'
summary(object, ...)
```

Arguments

object, x	A gs_fit object.
...	Unused.
type	Type of residuals.

Value

- coef(): named numeric vector of the estimated parameters.
- vcov(): numeric matrix, the estimated covariance matrix of the parameters (rows and columns in the order of coef()).
- logLik(): an object of class "logLik", the maximized log-likelihood, with attributes df (number of parameters) and nobs.
- nobs(): integer, the number of observations used in the fit.
- residuals(): numeric vector with one value per observation, as described by type.
- sigma(), fitted(): numeric vectors with one value per observation, the conditional scale and the conditional location.
- print(), summary(): print the coefficient table (estimates, standard errors, t and p values), information criteria, persistence and convergence diagnostics, and return x (object) invisibly.

---

gs_gat *GAt innovations*

Description

Generalized asymmetric t distribution of Paolella (1997) with location 0, scale 1 and parameters nu (tail), d (peakedness) and xi (asymmetry, 1 is symmetric), see dgat(). Moments $E|z|^\delta$ exist for $\delta < \nu d$; the GARCH mode power is $\delta = 2$.

Usage

```
gs_gat(
  lower = c(nu = 0.05, d = 0.1, xi = 0.05),
  upper = c(nu = 100, d = 50, xi = 20),
  start = c(nu = 2, d = 4, xi = 1)
)
```

Arguments

lower, upper, start	Optional named vectors replacing the default bounds and start value.

Value

A gs_dist object.

Examples

```
d <- gs_gat()
d$max_power(c(nu = 2, d = 4, xi = 1))
```

`gs_gev` *GEV innovations*

Description

Generalized extreme value distribution with location 0, scale 1 and shape `xi`, $F(z) = \exp\{-(1 + \xi z)^{-1/\xi}\}$ on $1 + \xi z > 0$ (Gumbel when $\xi = 0$). The shape is restricted to (-0.5, 0.5): above -0.5 for regular likelihood properties, below 0.5 for a finite variance, which the GARCH mode power $\delta = 2$ requires.

Usage

```
gs_gev(lower = c(xi = -0.49), upper = c(xi = 0.49), start = c(xi = 0.01))
```

Arguments

`lower`, `upper`, `start`
: Optional named vectors replacing the default bounds and start value.

Details

The innovations are not centered: $E[z] = (\Gamma(1-\xi)-1)/\xi$, so the model parameter `mu` is a location, not the conditional mean.

Value

A `gs_dist` object.

Examples

```
d <- gs_gev()
d$mean(c(xi = 0.1))
```

gs_model *Set the parameters of an ARMA-GARCH/APARCH model*

Description

Attaches parameter values to a gs_spec(), checking lengths, admissible ranges and moment conditions. Parameters can be given one by one or as a named vector par (for example coef(fit)).

Usage

```
gs_model(
  spec,
  mu = NULL,
  ar = NULL,
  ma = NULL,
  omega = NULL,
  alpha = NULL,
  gamma = NULL,
  beta = NULL,
  delta = NULL,
  dist_par = NULL,
  par = NULL
)
```

Arguments

spec	A gs_spec object.
mu	Intercept (only when spec$include_mean).
ar, ma	AR and MA coefficients, lengths m and n.
omega	Positive constant of the scale recursion.
alpha, beta	Nonnegative coefficients, lengths p and q.
gamma	Asymmetry coefficients in (-1, 1), length p (APARCH only).
delta	Positive power (APARCH only).
dist_par	Distribution parameters, named by spec$dist$par_names.
par	Alternatively, a named vector with all free parameters.

Value

An object of class gs_model.

Examples

```
spec <- gs_spec(arma = c(1, 0), garch = c(1, 1), dist = gs_stable())
gs_model(spec, mu = 0, ar = 0.1, omega = 0.05, alpha = 0.1, beta = 0.85,
         dist_par = c(stable_alpha = 1.8, stable_beta = 0))
```

gs_sim *Simulate an ARMA-GARCH/APARCH model*

Description

Simulates the model equations of `gs_spec()` with innovations drawn from the model's distribution. The recursion starts at the unconditional location and scale level and the first `burnin` values are discarded.

Usage

```
gs_sim(model, n = 1000, burnin = 1000, seed = NULL)
```

Arguments

`model`	A `gs_model` object.
`n`	Length of the returned series.
`burnin`	Number of initial values discarded.
`seed`	Optional seed passed to `set.seed()`.

Value

A data frame with columns `y` (series), `sigma` (conditional scale) and `z` (innovations).

Examples

```
spec <- gs_spec(arma = c(1, 0), garch = c(1, 1), dist = gs_stable())
model <- gs_model(spec, mu = 0, ar = 0.1, omega = 0.05, alpha = 0.1, beta = 0.8,
                  dist_par = c(stable_alpha = 1.8, stable_beta = 0))
x <- gs_sim(model, n = 500, seed = 1)
plot(x$y, type = "l")
```

gs_spec *Specify an ARMA-GARCH/APARCH model*

Description

The model, in the intercept form of Wuertz et al. (2006) and fGarch, is

$$x_t = \mu + \sum_{i=1}^{m} a_i x_{t-i} + \sum_{j=1}^{n} b_j e_{t-j} + e_t, \qquad e_t = \sigma_t z_t,$$

$$\sigma_t^{\delta} = \omega + \sum_{i=1}^{p} \alpha_i (|e_{t-i}| - \gamma_i e_{t-i})^{\delta} + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^{\delta},$$

with z_t i.i.d. from `dist`. In GARCH mode (`aparch = FALSE`) the asymmetry γ_i is 0 and δ is fixed at `dist$default_delta` (1 for stable, 2 otherwise). `mu` is an intercept: the unconditional location is $\mu/(1 - \sum a_i)$. Since the innovations are location scale (not standardized), σ_t is a conditional scale.

Usage

```
gs_spec(
  arma = c(0, 0),
  garch = c(1, 1),
  aparch = FALSE,
  include_mean = TRUE,
  dist = gs_stable()
)
```

Arguments

`arma`	Integer vector `c(m, n)`, ARMA orders.
`garch`	Integer vector `c(p, q)`, number of α (p >= 1) and β (q >= 0) terms.
`aparch`	Logical, estimate the asymmetry γ and the power δ.
`include_mean`	Logical, include the intercept μ.
`dist`	Innovation distribution, a `gs_dist` object.

Value

An object of class `gs_spec`.

References

Wuertz, D., Chalabi, Y. and Luksan, L. (2006). Parameter estimation of ARMA models with GARCH/APARCH errors: an R and SPlus software implementation. Journal of Statistical Software.

Examples

```
gs_spec(arma = c(1, 0), garch = c(1, 1), aparch = TRUE, dist = gs_stable())
```

`gs_stable` *Stable innovations (S0 parametrization)*

Description

Standardized stable distribution $S(\alpha, \beta, 1, 0; 0)$ in Nolan's S0 parametrization, computed with `libstable4u`. Parameters: `stable_alpha` (tail index, in (1, 2)) and `stable_beta` (skewness, in (-1, 1)).

Usage

```
gs_stable(
  lower = c(stable_alpha = 1.01, stable_beta = -0.99),
  upper = c(stable_alpha = 1.99, stable_beta = 0.99),
  start = c(stable_alpha = 1.8, stable_beta = 0)
)
```

Arguments

`lower`, `upper`, `start`
: Optional named vectors replacing the default bounds and start value.

Details

The variance is infinite, so σ_t is a conditional scale, not a standard deviation. In GARCH mode the power is $\delta = 1$ (absolute value GARCH), since $E|z|^\delta$ is finite only for $\delta < \alpha$. The mean is $-\beta \tan(\pi\alpha/2)$.

Value

A `gs_dist` object.

References

Nolan, J. P. (2020). Univariate Stable Distributions. Springer.

Royuela-del-Val, J., Simmross-Wattenberg, F. and Alberola-Lopez, C. (2017). libstable: Fast, Parallel, and High-Precision Computation of alpha-Stable Distributions in R, C/C++, and MATLAB. Journal of Statistical Software, 78(1), 1-25.

Examples

```
d <- gs_stable()
d$log_density(c(-1, 0, 1), c(stable_alpha = 1.7, stable_beta = 0.2))
```

`gs_stationarity` *Persistence of the APARCH scale recursion*

Description

Computes $\sum_j \beta_j + \sum_i \alpha_i \kappa_i$, with $\kappa_i = E(|z| - \gamma_i z)^\delta$ from `gs_aparch_moment()`. A value below 1 means $E(\sigma_t^\delta)$ is finite (Ding, Granger and Engle, 1993; Mittnik, Paolella and Rachev, 2002). It is `Inf` when δ is not below the moment bound of the innovations.

Usage

```
gs_stationarity(object)
```

Arguments

`object` A `gs_model` or `gs_fit` object.

Value

The persistence, a single number.

Examples

```
spec <- gs_spec(garch = c(1, 1), dist = gs_stable())
m <- gs_model(spec, mu = 0, omega = 0.05, alpha = 0.1, beta = 0.8,
              dist_par = c(stable_alpha = 1.8, stable_beta = 0))
gs_stationarity(m)
```

`predict.gs_fit` *Forecast an ARMA-GARCH/APARCH model*

Description

Forecasts from a fitted model (`predict.gs_fit`) or from a model with given parameters and a data history (`predict.gs_model`, useful in rolling windows with fixed parameters).

Usage

```
## S3 method for class 'gs_fit'
predict(
  object,
  n_ahead = 1,
  level = c(0.01, 0.05),
  n_sim = 10000,
  seed = NULL,
  ...
)

## S3 method for class 'gs_model'
predict(
  object,
  data,
  n_ahead = 1,
  level = c(0.01, 0.05),
  n_sim = 10000,
  seed = NULL,
  ...
)
```

Arguments

`object`	A `gs_fit` or `gs_model` object.
`n_ahead`	Forecast horizon.
`level`	Probabilities of the reported quantiles.
`n_sim`	Number of simulated paths used when `n_ahead` > 1.
`seed`	Optional seed for the simulated paths.
`...`	Unused.
`data`	For `gs_model`, the observed history (numeric vector).

Details

One step ahead everything is exact: the conditional location $m_{T+1} = \mu + \sum a_i x_{T+1-i} + \sum b_j e_{T+1-j}$, the conditional scale σ_{T+1}, and the quantiles $m_{T+1} + \sigma_{T+1} F^{-1}(p)$ (the value at risk at level `p` is minus the `p` quantile). For horizons $k \geq 2$:

- `sigma` is $(E\,\sigma_{T+k}^{\delta})^{1/\delta}$ from the APARCH recursion with $\kappa = E(|z| - \gamma z)^{\delta}$ (as in fGarch);
- quantiles come from `n_sim` simulated paths, since the predictive distribution is a scale mixture and has no closed form;
- `mean` is exact when $E[z] = 0$ or $\delta = 1$, otherwise it is the average of the simulated paths (column `mean_method`).

With stable innovations there is no forecast variance; `sigma` is a scale. `mean` is `NA` when the innovations have no mean.

Value

A data frame with columns `horizon`, `location` (one step only), `mean`, `mean_method`, `sigma` and one column per `level` named `q_<level>`.

Examples

```
spec <- gs_spec(arma = c(1, 0), garch = c(1, 1), dist = gs_stable())
model <- gs_model(spec, mu = 0, ar = 0.1, omega = 0.05, alpha = 0.1, beta = 0.8,
                  dist_par = c(stable_alpha = 1.8, stable_beta = 0))
x <- gs_sim(model, n = 500, seed = 1)$y
predict(model, data = x, n_ahead = 5, n_sim = 2000, seed = 1)
```

Index

coef(), *9*
coef.gs_fit *(*gs_fit-methods*)*, 10

dgat *(*GAT*)*, 2
dgat(), *11*

fitted(), *9*
fitted.gs_fit *(*gs_fit-methods*)*, 10

GAT, 2
gat *(*GAT*)*, 2
gs_aparch_moment, 4
gs_aparch_moment(), *16*
gs_bootstrap, 5
gs_bootstrap(), *9*
gs_check_dist, 6
gs_check_dist(), *7, 8*
gs_dist, 6
gs_dist(), *6*
gs_fit, 8
gs_fit(), *5*
gs_fit-methods, 10
gs_gat, 11
gs_gat(), *6*, *8*
gs_gev, 12
gs_gev(), *6*, *8*
gs_model, 13
gs_sim, 14
gs_spec, 14
gs_spec(), *8*, *13, 14*
gs_stable, 15
gs_stable(), *6*, *8*
gs_stationarity, 16
gs_stationarity(), *8*

logLik(), *9*
logLik.gs_fit *(*gs_fit-methods*)*, 10

nobs.gs_fit *(*gs_fit-methods*)*, 10

pgat *(*GAT*)*, 2
predict(), *9*
predict.gs_fit, 17
predict.gs_model *(*predict.gs_fit*)*, 17
print.gs_fit *(*gs_fit-methods*)*, 10

qgat *(*GAT*)*, 2

residuals(), *9*
residuals.gs_fit *(*gs_fit-methods*)*, 10
rgat *(*GAT*)*, 2
Rsolnp::solnp(), *9*

set.seed(), *14*
sigma(), *9*
sigma.gs_fit *(*gs_fit-methods*)*, 10
summary.gs_fit *(*gs_fit-methods*)*, 10

vcov(), *9*
vcov.gs_fit *(*gs_fit-methods*)*, 10