

Package: gratis (via r-universe)

October 15, 2026

Type Package

Title Generating Time Series with Diverse and Controllable Characteristics

Version 1.9

Description Generates synthetic time series based on various univariate time series models including MAR and ARIMA processes. Kang, Y., Hyndman, R.J., Li, F.(2020) [doi:10.1002/sam.11461](https://doi.org/10.1002/sam.11461).

License GPL-3

URL <https://github.com/ykang/gratis>

BugReports <https://github.com/ykang/gratis/issues/>

Depends R (>= 3.5.0)

Imports doRNG, dplyr, foreach, forecast (>= 8.16), GA, generics, magrittr, methods, mvtnorm, polynom, purrr, stats, tibble, tsibble, utils

Suggests doParallel, feasts, fGarch, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0), tsfeatures

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyLoad yes

NeedsCompilation no

RoxygenNote 7.3.3

Repository <https://robjhyndman.r-universe.dev>

Date/Publication 2026-10-15 13:47:58 UTC

RemoteUrl <https://github.com/ykang/gratis>

RemoteRef HEAD

RemoteSha 0a8ba7e42a36848eb1f964e6763e52f156828e49

Contents

arima_model	2
ets_model	4
generate.mar	6
generate_msts	8
generate_ts	9
generate_ts_with_target	10
mar_model	11
pi_coefficients	13
rmixnorm	15
rmixnorm_ts	16
simulate.gratis_arima	17
simulate.mar	19
simulate_target	20
Index	24

arima_model	<i>Specify an ARIMA model</i>
-------------	-------------------------------

Description

Construct a Gaussian ARIMA(p, d, q)(P, D, Q)_m model that can be used with [simulate.gratis_arima\(\)](#) or [generate.Arima\(\)](#). Omitted orders and coefficients are inferred where possible or randomly selected from the supported parameter space. Construction does not simulate observations or store a seed for later simulations.

Usage

```
arima_model(
  frequency = 1,
  p = NULL,
  d = NULL,
  q = NULL,
  P = NULL,
  D = NULL,
  Q = NULL,
  constant = NULL,
  phi = NULL,
  theta = NULL,
  Phi = NULL,
  Theta = NULL,
  sigma = NULL
)
```

Arguments

frequency	Positive integer seasonal period, for example 12 for monthly data. A value of 1 denotes a non-seasonal model.
p, q	Non-negative integer non-seasonal AR and MA orders. If omitted, inferred from supplied phi and theta, respectively, or sampled.
d	Non-negative integer ordinary difference order, with $d + D \leq 2$.
P, Q	Non-negative integer seasonal AR and MA orders. If omitted, inferred from supplied Phi and Theta, respectively, or sampled.
D	Non-negative integer seasonal difference order. Seasonal arguments require frequency > 1.
constant	Mean when $d + D = 0$, or slope of a linear drift when $d + D = 1$. Must be zero when $d + D > 1$.
phi	Finite numeric vector of length p containing AR parameters.
theta	Finite numeric vector of length q containing MA parameters. MA lag polynomials use positive signs: $1 + \theta(B)$.
Phi	Finite numeric vector of length P containing seasonal AR parameters, at lags frequency, $2 * \text{frequency}$, and so on.
Theta	Finite numeric vector of length Q containing seasonal MA parameters, using the same positive-sign convention as theta.
sigma	The standard deviation of the noise. Its squared variance must lie in the positive normal floating-point range, in both R and Python.

Details

When orders are omitted, p and q are sampled from $\{0, 1, 2, 3\}$, P and Q from $\{0, 1, 2\}$, d from $\{0, 1, 2\}$, and D from $\{0, 1\}$, with the restriction $d + D \leq 2$. If constant is omitted, it is sampled from $(-3, 3)$ unless $d + D = 2$, in which case it is set to zero. AR and MA parameters are sampled so the stationary and invertible parts of the process are valid, and sigma is sampled from $(1, 5)$. Above order two, random AR polynomials allow every possible count of real roots. This has full support, but is not uniform over coefficient space.

Value

A `gratis_arima` object inheriting from `Arima`. Its simulation method uses scaled innovations and preserves all seasonal coefficients, including coefficients whose sum is zero. Models fitted separately with **forecast** retain their own simulation method.

Author(s)

Rob J Hyndman

See Also

[simulate.gratis_arima](#) for initialization and controlled innovations; [generate.Arima](#) for batches of paths.

Examples

```
# An AR(2) model with random parameters
model1 <- arima_model(p = 2, d = 0, q = 0)
# An AR(2) model with specific parameters
model2 <- arima_model(p = 2, d = 0, q = 0, phi = c(1.34, -0.64), sigma = 15)
# Seasonal ARIMA model with randomly selected parameters
model3 <- arima_model(frequency = 4)
# Simulate from each model and plot the results
plot(simulate(model1, 100))
plot(simulate(model2, 100))
plot(simulate(model3, 100))
```

ets_model

Specify an ETS model

Description

Construct an ETS state space model that can be used with [simulate.ets\(\)](#) or [generate.ets\(\)](#). Any omitted argument is randomly selected from the supported parameter space.

Usage

```
ets_model(
  frequency = 1,
  error = NULL,
  trend = NULL,
  seasonal = NULL,
  alpha = NULL,
  beta = NULL,
  gamma = NULL,
  phi = NULL,
  level = NULL,
  slope = NULL,
  season = NULL,
  damped = NULL,
  sigma = NULL
)
```

Arguments

frequency	Positive integer seasonal period, for example 12 for monthly data. Seasonal models require a value greater than 1.
error	Character string specifying the error component: "A" for additive or "M" for multiplicative.
trend	Character string specifying the trend component: "N" for none or "A" for additive.

seasonal	Character string specifying the seasonal component: "N" for none, "A" for additive, or "M" for multiplicative.
alpha	Level smoothing parameter, strictly between 0 and 1.
beta	Trend smoothing parameter in $[0, \alpha]$. Ignored without a trend.
gamma	Seasonal smoothing parameter in $[0, 1 - \alpha]$. Ignored without seasonality.
phi	Damping parameter in $(0, 1]$. Ignored without a damped trend.
level	Finite initial level ℓ_0 . Must be positive for multiplicative errors or seasonality.
slope	Finite initial additive trend state b_0 . Ignored without a trend.
season	Finite numeric vector of initial seasonal states $s_{1-m}, \dots, s_0$, ordered oldest to newest, of length frequency. Multiplicative factors must be positive. Supplied states are not normalized.
damped	Logical value indicating whether an additive trend is damped.
sigma	The standard deviation of the noise. Its squared variance must lie in the positive normal floating-point range, in both R and Python. It is measured on the observation scale for additive errors and is dimensionless for multiplicative errors.

Details

The error component is sampled from "A" and "M", the trend component from "N" and "A", and the seasonal component from "N", "A", and "M" when frequency > 1. Smoothing parameters are sampled until they satisfy the ETS admissibility conditions. The innovation variance is sampled from (1, 5) for additive errors and from (0.0001, 0.05) for multiplicative errors. Initial states are sampled from ranges appropriate to the selected components. Omitted smoothing parameters are resampled after rejection, with a maximum of 10000 attempts. Fixed parameters must satisfy the usual smoothing bounds: $0 < \alpha < 1$, $0 \leq \beta \leq \alpha$, $0 \leq \gamma \leq 1 - \alpha$, and $0 < \phi \leq 1$. Multiplicative seasonal factors are positive and sampled factors are normalized to have mean one.

Simulation starts from the supplied initial states without an automatic burn-in. Multiplicative innovations are clipped below at -1 by the **forecast** simulator, matching Python's convention. Admissibility and positive initial states do not guarantee positive or finite values for every simulated path.

Value

An ets object compatible with the **forecast** package's simulation methods.

Author(s)

Rob J Hyndman

See Also

[simulate.ets](#), [generate.ets](#)

Examples

```
# An ETS(A,A,N) model with random parameters
model1 <- ets_model(error = "A", trend = "A", seasonal = "N")
# An ETS(A,A,N) model with specific parameters
model2 <- ets_model(
  error = "A", trend = "A", seasonal = "N",
  alpha = 0.3, beta = 0.2, level = 0, slope = 1, sigma = 2
)
# A quarterly seasonal ETS model with random admissible parameters
model3 <- ets_model(error = "A", trend = "N", seasonal = "A", frequency = 4)
# Simulate from each model and plot the results
plot(simulate(model1, 100))
plot(simulate(model2, 100))
plot(simulate(model3, 100))
```

generate.mar

Generate a tsibble of synthetic time series

Description

Generate one or more synthetic time series from a fitted or specified model. Methods are provided for `mar_model()` objects, ETS objects from `ets_model()`, and ARIMA objects from `arima_model()`. Each series is produced by repeatedly calling the corresponding `simulate()` method. Automatic burn-in for models created by `arima_model()` is calculated once per batch. This does not change the simulation behavior of separately fitted **forecast** models.

Usage

```
## S3 method for class 'mar'
generate(x, length = 100, nseries = 10, ..., seed = NULL)

## S3 method for class 'ets'
generate(x, length = 100, nseries = 10, ..., seed = NULL)

## S3 method for class 'Arima'
generate(x, length = 100, nseries = 10, ..., seed = NULL)
```

Arguments

<code>x</code>	A model object of class <code>mar</code> , <code>ets</code> , or <code>Arima</code> .
<code>length</code>	Positive integer output length of each series, excluding burn-in.
<code>nseries</code>	Positive integer number of series to generate.
<code>...</code>	Other arguments passed to the relevant <code>simulate()</code> method. See <code>simulate.mar()</code> and <code>simulate.gratis_arima()</code> for initialization controls. Supplied innovations must produce exactly length observations. Not every model accepts the same arguments.

seed Optional seed set once for the whole batch. Successive series use successive random draws, and the previous RNG state is restored after generation. The default, NULL, advances the current RNG normally.

Details

The generated series are returned as a tibble. The index is inferred from the model frequency or seasonal periods where possible, with special handling for common weekly, hourly, half-hourly, quarter-hourly, and minute data.

Value

A tibble containing the generated values. For multiple series the output has `length * nseries` rows and includes a key column identifying each series. Unlike the Python API, the result contains calendar or seasonal index information rather than being an unlabelled numeric matrix.

Author(s)

Rob J Hyndman

References

Feng Li, Mattias Villani, and Robert Kohn. (2010). Flexible Modeling of Conditional Distributions using Smooth Mixtures of Asymmetric Student T Densities, *Journal of Statistical Planning and Inference*, 140(12), pp. 3638-3654.

See Also

[mar_model](#), [arima_model](#), [ets_model](#), [simulate.mar](#)

Examples

```
# MAR model with constant variances
phi <- cbind(c(0.8, 0), c(0.6, 0.3))
weights <- c(0.8, 0.2)
model1 <- mar_model(phi = phi, sigmas = c(1, 2), weights = weights)
generate(model1, nseries = 5)

# MAR model for hourly data with daily and weekly periods
hourly_model <- mar_model(seasonal_periods = c(24, 24*7))
generate(hourly_model, length = 24 * 7, nseries = 2)

# ARIMA and ETS models also work
generate(arima_model(p = 1, d = 0, q = 0, phi = 0.5), length = 20, nseries = 2)
generate(ets_model(error = "A", trend = "N", seasonal = "N"), length = 20, nseries = 2)
```

generate_msts	<i>Generate a random multiple-seasonal MAR time series</i>
---------------	--

Description

This function is deprecated. Use `mar_model()` with `generate.mar()` or `simulate.mar()` instead.

Usage

```
generate_msts(
  seasonal.periods = c(7, 365),
  n = 800,
  nComp = NULL,
  output_format = "list"
)
```

Arguments

seasonal.periods	Numeric vector of seasonal periods.
n	Length of the generated series.
nComp	Number of mixture components. If NULL, this is sampled randomly for each generated component series.
output_format	Output format: "list" for the historical msts result or "tsibble" for a tsibble.

Value

A multiple-seasonal time series when `output_format = "list"`, or a tsibble when `output_format = "tsibble"`.

See Also

[mar_model](#), [generate.mar](#)

Examples

```
x <- generate_msts(seasonal.periods = c(7, 365), n = 800, nComp = 2, output_format = "list")
forecast::autoplot(x)
```

generate_ts	<i>Generate random MAR time series</i>
-------------	--

Description

This function is deprecated. Use `mar_model()` with `generate.mar()` or `simulate.mar()` instead.

Usage

```
generate_ts(n.ts = 1, freq = 1, nComp = NULL, n = 120, output_format = "list")
```

Arguments

n.ts	Number of time series to generate.
freq	Seasonal period of the generated series.
nComp	Number of mixture components. If NULL, this is sampled once and reused for the generated series in this call.
n	Length of each generated series.
output_format	Output format: "list" for the historical list format or "tsibble" for a list of tsibbles.

Value

When `output_format = "list"`, a list containing generated time series and the parameters used for each series. When `output_format = "tsibble"`, a list of tsibbles.

Author(s)

Yanfei Kang and Feng Li

References

Wong, CS & WK Li (2000).

See Also

[mar_model](#), [generate.mar](#)

Examples

```
x <- generate_ts(n.ts = 2, freq = 12, nComp = 2, n = 120)
x$N1$pars
forecast::autoplot(x$N1$x)
```

```
generate_ts_with_target
```

Generate time series with target features

Description

This function is deprecated. Use [generate_target\(\)](#) instead.

Usage

```
generate_ts_with_target(
  n,
  ts.length,
  freq,
  seasonal,
  features,
  selected.features,
  target,
  parallel = TRUE,
  output_format = "list"
)
```

Arguments

<code>n</code>	Number of time series to generate.
<code>ts.length</code>	Length of each generated time series.
<code>freq</code>	Seasonal frequency, or seasonal periods for multiple-seasonal data.
<code>seasonal</code>	Integer seasonality flag: 0 for non-seasonal data, 1 for single-seasonal data, and 2 for multiple-seasonal data.
<code>features</code>	Character vector of feature function names passed to tsfeatures() .
<code>selected.features</code>	Character vector naming the features to match.
<code>target</code>	Numeric vector of target feature values in the same order as <code>selected.features</code> .
<code>parallel</code>	Logical value or parallel backend specification indicating whether the genetic algorithm should evaluate candidates in parallel.
<code>output_format</code>	Output format: "list" for a ts or msts object, or "tsibble" for a tsibble.

Value

A ts or msts object containing the generated series, or a tsibble when `output_format = "tsibble"`. The historical "list" option returns a time-series object, not an R list.

Author(s)

Yanfei Kang

See Also

[generate_target](#), [simulate_target](#)

Examples

```
## Not run:
library(tsfeatures)
x <- generate_ts_with_target(
  n = 1, ts.length = 60, freq = 1, seasonal = 0,
  features = c("entropy", "stl_features"), selected.features = c("entropy", "trend"),
  target = c(0.6, 0.9), parallel = FALSE
)
forecast::autoplot(x)

## End(Not run)
```

mar_model

Specify a mixture autoregressive model

Description

Construct a mixture autoregressive (MAR) model made up of k Gaussian ARIMA($p, d, 0$)($P, D, 0$) $_m$ components. The resulting mar object can be passed to [simulate.mar\(\)](#) to simulate one series or to [generate.mar\(\)](#) to generate a tibble of several series.

Usage

```
mar_model(
  k = NULL,
  p = NULL,
  d = NULL,
  phi = NULL,
  P = NULL,
  D = NULL,
  Phi = NULL,
  constants = NULL,
  sigmas = NULL,
  weights = NULL,
  seasonal_periods = 1L
)
```

Arguments

k	Positive integer number of mixture components.
p	Non-negative integer scalar or vector of length k giving the non-seasonal AR orders p_i . Ignored when phi is supplied.

d	Non-negative integer scalar or vector of length k giving the ordinary differencing orders d_i .
phi	Numeric matrix of non-seasonal AR parameters. Column i contains the parameters for component i . Use zero padding for components of different orders. The row count determines p.
P	Non-negative integer scalar or vector of length k giving the seasonal AR orders P_i . Ignored when all seasonal_periods are 1 or when Phi is supplied.
D	Non-negative integer scalar or vector of length k giving seasonal difference orders D_i . Forced to zero where $d = 2$ or the seasonal period is 1.
Phi	Numeric matrix of seasonal AR parameters. Column i contains the parameters for component i . Ignored when all seasonal_periods are 1. Use zero padding for components of different orders. The row count determines P.
constants	Numeric vector of length k containing the component constants $c_1, \dots, c_k$. These are recursion intercepts, not unconditional means. Forced to zero for components with $d + D > 1$.
sigmas	Positive innovation variances of length k, or a list of k GARCH specifications created by <code>garchSpec()</code> . GARCH standard deviations are squared internally to obtain innovation variances. These are externally simulated volatility paths: their recursion uses the latent GARCH shocks, not the subsequently generated MAR residuals. This is not a coupled MAR-GARCH model.
weights	Non-negative finite numeric vector of length k containing the component probabilities $\alpha_1, \dots, \alpha_k$. Values are normalized to sum to 1; at least one must be positive.
seasonal_periods	Positive integer seasonal period applied to every component, or a numeric vector of length k giving the seasonal period for each component. A value of 1 denotes a non-seasonal component.

Details

Component i has the form

$$(1 - B)^{d_i} (1 - B^{m_i})^{D_i} (1 - \phi_i(B)) (1 - \Phi_i(B)) y_t = c_i + \sqrt{v_{i,t}} \epsilon_t$$

and is selected independently at each time with probability α_i . Every component conditions on the same generated past; components are not independently generated paths that are subsequently averaged. Here B is the backshift operator, m_i is the seasonal period, ϵ_t is a standard normal variate, and $\phi_i(B)$ and $\Phi_i(B)$ are the non-seasonal and seasonal AR coefficient sums, with seasonal coefficients multiplying lags $B^{m_i}, B^{2m_i}, \dots$.

Any missing model argument is randomly selected. Randomly selected non-seasonal AR orders are sampled from $\{0, 1, 2, 3\}$, non-seasonal differences from $\{0, 1, 2\}$, seasonal AR orders from $\{0, 1, 2\}$, and seasonal differences from $\{0, 1\}$. AR parameters are sampled from the stationary region, constants from $(-3, 3)$, sigmas from $(1, 5)$, and mixture weights from $(0, 1)$ before normalization. If k is omitted, the number of components is sampled from $\{1, 2, 3, 4, 5\}$, unless it can be inferred from another argument.

Stationary component AR polynomials do not guarantee a stationary mixture. Integrated components are also allowed. Simulation uses zero initial states and a fixed burn-in by default; near-unit-root models may need a substantially longer `n.start` in `simulate.mar()`.

Value

An object of class `mar`. The object stores the component orders, coefficients, constants, sigmas, normalized weights, seasonal periods, and the expanded AR recursion coefficients used by `simulate.mar()`.

Author(s)

Rob J Hyndman

See Also

`simulate.mar`, `generate.mar`, `pi_coefficients`

Examples

```
# Quarterly MAR model with randomly selected parameters
model1 <- mar_model(seasonal_periods = 4)

# Daily MAR model with randomly selected parameters
model2 <- mar_model(seasonal_periods = c(7, 365))

# MAR model with constant variances and user-specified coefficients
phi <- cbind(c(0.8, 0), c(0.6, 0.3))
weights <- c(0.8, 0.2)
model3 <- mar_model(phi = phi, d = 0, sigmas = c(1, 2), weights = weights)

# MAR model with heteroskedastic errors
if (requireNamespace("fGarch", quietly = TRUE)) {
  sigmas_spec <- list(
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.06))),
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.05)))
  )
  model4 <- mar_model(phi = phi, sigmas = sigmas_spec, weights = weights)
}
```

pi_coefficients

Compute AR recursion coefficients from SARIMA coefficients

Description

Convert SARIMA coefficients into the equivalent infinite-order AR recursion coefficients, truncated after the last coefficient whose absolute value is greater than `tol`. These coefficients are used internally by `mar_model()` and are also exported for users who need the same conversion.

Usage

```
pi_coefficients(
  ar = 0,
  d = 0L,
  ma = 0,
  sar = 0,
  D = 0L,
  sma = 0,
  m = 1L,
  tol = 1e-07,
  n_terms = 500L
)
```

Arguments

ar, ma	Finite numeric vectors of non-seasonal AR and MA coefficients, excluding the leading 1 of each lag polynomial.
d, D	Non-negative integer ordinary and seasonal difference orders. Seasonal differencing is ignored when m = 1.
sar, sma	Finite numeric vectors of seasonal AR and MA coefficients. Ignored when m = 1.
m	Positive integer seasonal period. A value of 1 is non-seasonal.
tol	Non-negative finite truncation tolerance. Coefficients are returned through the last position whose absolute value exceeds tol.
n_terms	Lag limit for an infinite AR approximation with MA terms. The limit is increased to include the full AR and MA polynomial degrees. A warning is issued if the tail has not decayed below tol. Without MA terms the complete finite AR polynomial is returned.

Details

AR lag polynomials use negative coefficient signs, while MA lag polynomials use positive signs. The result expands the ratio of the AR/differencing polynomial to the MA polynomial; no stationarity or invertibility is imposed.

Value

A numeric vector of AR recursion coefficients in increasing lag order, without an intercept or leading 1. Complete cancellation returns `numeric(0)`. An unconverged MA approximation issues a warning; increase `n_terms`, noting that noninvertible MA terms may not converge.

Author(s)

Rob J Hyndman

See Also

[mar_model](#)

Examples

```
pi_coefficients(ar = 0.8)
pi_coefficients(ar = c(0.4, -0.2), d = 1)
pi_coefficients(ar = 0.3, sar = 0.2, D = 1, m = 12)
```

rmixnorm	<i>Generate random values from a mixture of multivariate normal distributions</i>
----------	---

Description

Draw random values from a finite mixture of k multivariate normal distributions, each with dimension q . For univariate mixtures, means and sigmas may be supplied as vectors. Covariance matrices must be finite, symmetric and positive definite, including those of zero-weight components.

Usage

```
rmixnorm(n, means, sigmas, weights)
```

Arguments

n	Non-negative integer number of observations to generate.
means	A $q \times k$ matrix of component means, or a vector of length k for univariate mixtures.
sigmas	A $q \times q \times k$ array of component covariance matrices, or a vector of length k for univariate mixtures. These are variances or covariances, not standard deviations.
weights	Finite, non-negative component weights, with at least one positive value. Weights are normalized to sum to one.

Value

An $n \times q$ matrix of generated data, or a numeric vector when $q = 1$.

Author(s)

Feng Li, Central University of Finance and Economics.

References

Villani et al 2009.

See Also

[rmixnorm_ts](#) for autoregressive conditional means.

Examples

```
out <- rmixnorm(
  n = 1000, means = c(-5, 0, 5), sigmas = c(1, 1, 3),
  weights = c(0.3, 0.4, 0.3)
)
hist(out, breaks = 100, freq = FALSE)
```

rmixnorm_ts

Simulate a mixture-normal autoregressive process

Description

Simulate a univariate process whose conditional distribution is a finite mixture of Gaussian distributions. The conditional mean of each component is an autoregressive process with its own parameter vector. All components use the common observed past, rather than independent latent component paths. No burn-in is removed: the first $\min(n, \text{max_order})$ observations are fixed at `yinit`, where `max_order` is the largest component AR order. A shorter series consists entirely of these initial values.

Usage

```
rmixnorm_ts(n, means.ar.par.list, sigmas.list, weights, yinit = 0)
```

Arguments

<code>n</code>	Non-negative integer total path length, including initial observations.
<code>means.ar.par.list</code>	List of component AR parameter vectors. The first element of each vector is the intercept, followed by at least one lag coefficient. Orders may differ between components; values must be finite.
<code>sigmas.list</code>	List of positive, finite component variance vectors, each of length <code>n</code> . Values are variances, not standard deviations.
<code>weights</code>	Finite, non-negative component weights. At least one must be positive; weights are normalized to sum to one.
<code>yinit</code>	Finite scalar assigned to the initial observations.

Value

A `ts` object of length `n`, or `numeric(0)` for zero length. Stationarity is not enforced.

Author(s)

Feng Li, Central University of Finance and Economics.

References

Feng Li, Mattias Villani, and Robert Kohn. (2010). Flexible Modeling of Conditional Distributions using Smooth Mixtures of Asymmetric Student T Densities, *Journal of Statistical Planning and Inference*, 140(12), pp. 3638-3654.

See Also

[rmixnorm](#), [mar_model](#), [simulate.mar](#)

Examples

```
if (requireNamespace("fGarch", quietly = TRUE)) {
  n <- 1000
  means.ar.par.list <- list(c(0, 0.8), c(0, 0.6, 0.3))
  sigmas.spec <- list(
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.06)), cond.dist = "norm"),
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.05)), cond.dist = "norm")
  )
  sigmas.list <- lapply(
    lapply(sigmas.spec, fGarch::garchSim, extended = TRUE, n = n),
    function(x) x$sigma^2
  )
  weights <- c(0.8, 0.2)
  y <- rmixnorm_ts(
    n = n, means.ar.par.list = means.ar.par.list, sigmas.list = sigmas.list,
    weights = weights
  )
  plot(y)
}
```

simulate.gratis_arma *Simulate from a specified ARIMA model*

Description

Simulate the stationary ARMA part from zero states, discard burn-in, and then apply differencing inverses and the model's mean or linear drift. All random innovations, including burn-in, have standard deviation $\sqrt{\text{object}\$sigma^2}$. This method applies only to models constructed by [arma_model\(\)](#), not to fitted **forecast** models.

Usage

```
## S3 method for class 'gratis_arma'
simulate(
  object,
  nsim = 100,
  seed = NULL,
  n.start = NULL,
```

```

    innov = NULL,
    start.innov = NULL,
    ...
  )

```

Arguments

<code>object</code>	A model created by <code>arima_model()</code> .
<code>nsim</code>	Non-negative integer output length, excluding burn-in. Supplied <code>innov</code> overrides the length with <code>length(innov)</code> .
<code>seed</code>	Optional integer random seed. A supplied seed temporarily sets the RNG state, restoring the previous state or its absence afterwards. <code>NULL</code> uses and advances the current stream for unsupplied innovations.
<code>n.start</code>	Non-negative integer burn-in length. <code>NULL</code> selects an automatic length; zero starts from zero states without burn-in. An explicit value skips the energy search, but not the AR stationarity check.
<code>innov</code>	Optional finite numeric vector of output-period innovations, already on the model's noise scale. Values are not multiplied by <code>sqrt(object\$sigma2)</code> again.
<code>start.innov</code>	Optional finite numeric vector of burn-in innovations. Its length determines <code>n.start</code> when that argument is <code>NULL</code> .
<code>...</code>	Other arguments, not supported.

Value

A `ts` object with the model's frequency and the requested output length, or `numeric(0)` for zero output length.

Automatic initialization

For an AR model, impulse-response windows double from a power of two covering at least 64 steps, $2 * (\text{AR order} + \text{MA order} + 1)$, and twelve time constants of the slowest pole. Burn-in is half the first window whose latter half contains at most $1e-8$ of its energy. Repeated and seasonal poles and near AR-MA cancellation are included. A pure MA model uses its expanded MA order as burn-in. The same rule is used in Python.

This is a numerical tail estimate, not exact stationary initialization. Searches exceeding 2^{22} steps stop with an error requesting an explicit `n.start`. Differencing inverses and the mean or drift are applied only after the stationary ARMA burn-in has been discarded.

See Also

[arima_model](#), [generate.Arima](#)

Examples

```

model <- arima_model(p = 1, d = 0, q = 0, phi = 0.99, constant = 0, sigma = 0.01)
simulate(model, nsim = 20, seed = 1)
# Supplied innovations are already scaled; zero burn-in gives zero prehistory.
impulse_model <- arima_model(p = 1, d = 0, q = 0, phi = 0.5, constant = 0, sigma = 1)
as.numeric(simulate(impulse_model, innov = c(1, 0, 0), n.start = 0))

```

simulate.mar

*Simulate from a mixture autoregressive model***Description**

Simulate one random sample path from a mixture of Gaussian autoregressive components created by `mar_model()`. At each time point, a component is sampled according to the model weights, that component's conditional mean and innovation variance are used, and the resulting series is returned after discarding a burn-in period.

Usage

```
## S3 method for class 'mar'
simulate(object, nsim = 100, seed = NULL, n.start = 100, ...)
```

Arguments

<code>object</code>	A <code>mar</code> object, usually created by <code>mar_model()</code> . All components condition on the same generated past; the result is not a weighted average of independently simulated component paths. Numeric <code>object\$sigmas</code> are innovation variances, not standard deviations.
<code>nsim</code>	Non-negative integer output length, excluding burn-in.
<code>seed</code>	Optional integer passed to <code>set.seed()</code> before simulation. <code>NULL</code> uses and advances the current random stream without resetting it. Use the seed argument to <code>generate.mar()</code> to seed a batch once rather than restarting the stream for each path.
<code>n.start</code>	Non-negative integer burn-in length, discarded before returning the simulated series. The default of 100 does not guarantee convergence to a stationary distribution, particularly for near-unit-root models. Stationary components do not by themselves guarantee mixture stability.
<code>...</code>	Other arguments, not currently used.

Value

A `msts` object of length `nsim`. For single-seasonal or non-seasonal models this also behaves like a base `ts` object. Zero output length returns `numeric(0)`. Values are not centered or standardized.

Author(s)

Rob J Hyndman

References

Feng Li, Mattias Villani, and Robert Kohn. (2010). Flexible Modeling of Conditional Distributions using Smooth Mixtures of Asymmetric Student T Densities, *Journal of Statistical Planning and Inference*, 140(12), pp. 3638-3654.

See Also[mar_model](#)**Examples**

```
# MAR model with constant variances
phi <- cbind(c(0.8, 0), c(0.6, 0.3))
weights <- c(0.8, 0.2)
model1 <- mar_model(phi = phi, sigmas = c(1, 2), weights = weights)
y <- simulate(model1, 100)
plot(y)

# MAR model with heteroskedastic errors
if (requireNamespace("fGarch", quietly = TRUE)) {
  sigmas_spec <- list(
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.06))),
    fGarch::garchSpec(model = list(alpha = c(0.05, 0.05)))
  )
  model2 <- mar_model(phi = phi, sigmas = sigmas_spec, weights = weights)
  y <- simulate(model2, 100)
  plot(y)
}
```

simulate_target

*Generate time series with target features***Description**

Search for MAR model parameters that produce time series with feature values close to a user-specified target. `simulate_target()` returns one generated series, while `generate_target()` repeats the search to return several generated series in a tibble.

Usage

```
simulate_target(
  length = 100,
  seasonal_periods = 1,
  feature_function,
  target,
  k = ifelse(length(seasonal_periods) == 1, 3, length(seasonal_periods)),
  tolerance = 0.05,
  trace = FALSE,
  parallel = FALSE
)

generate_target(
  length = 100,
  nseries = 10,
```

```

    seasonal_periods = 1,
    feature_function,
    target,
    k = ifelse(length(seasonal_periods) == 1, 3, length(seasonal_periods)),
    tolerance = 0.05,
    trace = FALSE,
    parallel = FALSE
  )

```

Arguments

length	Positive integer output length of each generated series.
seasonal_periods	Positive integer seasonal period, applied to every component, or a vector of length k specifying one period per component.
feature_function	Function that accepts a candidate time series and returns one numeric value per target feature. It should be deterministic for reproducible scores. Nonfinite values reject a candidate; errors raised by the function propagate. Callbacks used in parallel must be available to the workers along with their dependencies.
target	Numeric vector of target feature values. It must have the same length and order as the vector returned by <code>feature_function()</code> .
k	Number of MAR components to use. The default is 3 for single-seasonal or non-seasonal data, and the number of seasonal periods for multiple-seasonal data.
tolerance	Non-negative finite mean absolute feature-error tolerance. Reaching this value stops the search. Success is not guaranteed; a miss returns the best series with a warning reporting its error.
trace	Logical value indicating whether to show genetic algorithm progress.
parallel	Logical value or parallel backend specification passed to the genetic algorithm. Use TRUE to run fitness evaluations in parallel. A supplied cluster requires doParallel and remains caller-owned; stop it explicitly when it is no longer needed. Internally created clusters are stopped automatically.
nseries	Positive integer number of separate target searches. Searches are repeated sequentially, with successive draws from the current RNG stream; parallel controls candidate evaluations within each search.

Details

A genetic algorithm evaluates candidate MAR parameter vectors by simulating a series, scaling it, computing `feature_function()`, and comparing those features to `target`. Because each candidate series is scaled before feature evaluation, the target features should not depend on the absolute scale of the series. Finite nonconstant series are centered and sample-standardized using overflow-safe scaling; exactly constant series are left unchanged.

Mixture weights are the spacings between sorted breakpoints in $[0, 1]$, allowing the full simplex, including zero-weight components. If tolerance is not reached, the best series is returned with a warning giving its mean absolute feature error. Callback errors and incompatible feature lengths

are reported as errors; nonfinite candidate paths or features are rejected. The best final population member is selected using its cached fitness and series; generation histories are not retained or callbacks reevaluated.

Value

`simulate_target()` returns a `msts` object of length `length`. `generate_target()` returns a tibble containing `nseries` generated series. Returned values are on the standardized scale used for feature evaluation, except that exactly constant candidates are unchanged. Neither function returns fitted MAR parameters. A search with no finite valid candidate stops with an error.

Author(s)

Yanfei Kang and Rob J Hyndman

See Also

[mar_model](#), [generate.mar](#)

Examples

```
## Not run:
set.seed(1)
library(tsfeatures)
my_features <- function(y) {
  c(entropy(y), acf = acf(y, plot = FALSE)$acf[2:3, 1, 1])
}
# Simulate a ts with specified target features
y <- simulate_target(
  length = 60, feature_function = my_features, target = c(0.5, 0.9, 0.8)
)
my_features(y)
plot(y)

# Generate a tibble with specified target features
df <- generate_target(
  length = 60, feature_function = my_features, target = c(0.5, 0.9, 0.8)
)
df %>%
  as_tibble() %>%
  group_by(key) %>%
  dplyr::reframe(
    value = my_features(value),
    feature=c("entropy", "acf1", "acf2")
  )
autoplot(df)
# Simulate time series similar to an existing series
my_features <- function(y) {
  c(stl_features(y)[c("trend", "seasonal_strength", "peak", "trough")])
}
y <- simulate_target(
  length = length(USAccDeaths),
```

```
    seasonal_periods = frequency(USAccDeaths),  
    feature_function = my_features, target = my_features(USAccDeaths)  
  )  
  tsp(y) <- tsp(USAccDeaths)  
  plot(cbind(USAccDeaths, y))  
  cbind(my_features(USAccDeaths), my_features(y))  
  
## End(Not run)
```

Index

arima_model, [2](#), [6](#), [7](#), [17](#), [18](#)

ets_model, [4](#), [6](#), [7](#)

garchSpec, [12](#)

generate.Arima, [2](#), [3](#), [18](#)

generate.Arima(generate.mar), [6](#)

generate.ets, [4](#), [5](#)

generate.ets(generate.mar), [6](#)

generate.mar, [6](#), [8](#), [9](#), [11](#), [13](#), [19](#), [22](#)

generate_msts, [8](#)

generate_target, [10](#), [11](#)

generate_target(simulate_target), [20](#)

generate_ts, [9](#)

generate_ts_with_target, [10](#)

mar_model, [6–9](#), [11](#), [13](#), [14](#), [17](#), [19](#), [20](#), [22](#)

msts, [19](#), [22](#)

pi_coefficients, [13](#), [13](#)

rmixnorm, [15](#), [17](#)

rmixnorm_ts, [15](#), [16](#)

set.seed, [19](#)

simulate, [6](#)

simulate.ets, [4](#), [5](#)

simulate.gratis_arima, [2](#), [3](#), [6](#), [17](#)

simulate.mar, [6–9](#), [11–13](#), [17](#), [19](#)

simulate_target, [11](#), [20](#)

ts, [16](#), [18](#), [19](#)

tsfeatures, [10](#)