Package 'gratis'

May 8, 2026

Type Package

Title Generating Time Series with Diverse and Controllable Characteristics

Version 1.0.8

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>.

License GPL-3

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

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

Depends R (>= 3.5.0)

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

Suggests knitr, rlang, rmarkdown, shinydashboard

VignetteBuilder knitr

Encoding UTF-8

LazyLoad yes

NeedsCompilation no

Repository CRAN

RoxygenNote 7.3.1

Author Yanfei Kang [aut] (ORCID: <https://orcid.org/0000-0001-8769-6650>), Feng Li [aut, cre] (ORCID: <https://orcid.org/0000-0002-4248-9778>), Rob Hyndman [aut] (ORCID: <https://orcid.org/0000-0002-2140-5352>), Mitchell O'Hara-Wild [ctb] (ORCID: <https://orcid.org/0000-0001-6729-7695>), Bocong Zhao [ctb] (ORCID: <https://orcid.org/0000-0001-8434-9047>)

Maintainer Feng Li <feng.li@gsm.pku.edu.cn>

Date/Publication 2026-02-26 07:30:02 UTC

Contents

`app_gratis` *Web Application to generate time series with controllable features.*

Description

Web Application to generate time series with controllable features.

Usage

```
app_gratis()
```

`arima_model` *Specify parameters for an ARIMA model*

Description

This function allows the parameters of a Gaussian $ARIMA(p,d,q)(P,D,Q)[m]$ process to be specified. The output can be used in `simulate.Arima()` and `generate.Arima`. If any argument is `NULL`, the corresponding parameters are randomly selected. The AR and MA orders p and q are chosen from {0,1,2,3}, the seasonal AR and MA orders P and Q are from {0,1,2}, while the order of differencing, d is in {0,1,2}, and the order of seasonal differencing D is in {0,1}, with the restriction that $d+D \leq 2$. If `constant` is `NULL`, it is set to 0 if $d+D=2$, otherwise it is uniformly sampled on (-3,3). The model orders and the parameters are uniformly sampled. The AR and MA parameters are selected to give stationary and invertible processes when $d=D=0$. The noise variance sigma is uniformly sampled on (1,5). The parameterization is as specified in Hyndman & Athanasopoulos (2021).

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	The length of the seasonal period (e.g., 12 for monthly data).
p	An integer equal to the non-seasonal autoregressive order
d	An integer equal to the non-seasonal order of differencing
q	An integer equal to the non-seasonal moving average order
P	An integer equal to the seasonal autoregressive order
D	An integer equal to the seasonal order of differencing
Q	An integer equal to the seasonal moving average order
constant	The intercept term
phi	A numeric p-vector containing the AR parameters.
theta	A numeric p-vector containing the MA parameters.
Phi	A numeric p-vector containing the seasonal AR parameters.
Theta	A numeric p-vector containing the seasonal MA parameters.
sigma	The standard deviation of the noise.

Value

An ‘Arima‘ object as described in the arima function from the stats package.

Author(s)

Rob J Hyndman

See Also

simulate.Arima

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
library(forecast)
simulate(model1, 100) %>% plot()
simulate(model2, 100) %>% plot()
simulate(model3, 100) %>% plot()
```

---

ets_model *Specify parameters for an ETS model*

Description

This function allows the parameters of a ETS state space model to be specified. The output can be used in `simulate.ets()` and `generate.ets`. If any argument is `NULL`, the corresponding parameters are randomly selected. The error component is chosen from {A,M}, the trend component is chosen from {N,A,Ad}, and the seasonal component is chosen from {N,A,M}. In all cases, the component is selected uniformly on the options. The parameters are selected uniformly on the forecastable parameter space. The noise variance sigma is uniformly sampled on (1,5) for additive errors, and on (0.0001,0.05) for multiplicative errors. The initial states are chosen uniformly on (-1,1) in all cases except for multiplicative seasonal states which are uniform on (0.5, 1.5), and models with multiplicative errors for which the level is uniform on (2, 10). The parameterization is as specified in Hyndman & Athanasopoulos (2021).

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	The length of the seasonal period (e.g., 12 for monthly data).
error	A character string specifying the error part of the ETS model: either "A" or "M".
trend	A character string specifying the trend part of the ETS model: either "N", "A" or "Ad".
seasonal	A character string specifying the seasonal part of the ETS model: either "N", "A" or "M".
alpha	A numeric value for the smoothing parameter controlling the level.
beta	A numeric value for the smoothing parameter controlling the trend.
gamma	A numeric value for the smoothing parameter controlling the seasonality.
phi	A numeric value specifying the damping parameter.
level	A numeric value specifying the initial level ℓ_0.
slope	A numeric value specifying the initial slope b_0
season	A numeric vector specifying the initial states $s_{1-m}, ..., s_0$.
damped	A logical value indicating if the trend is damped or not.
sigma	The standard deviation of the noise.

Value

An 'ets' object as described in the `ets` function from the forecast package.

Author(s)

Rob J Hyndman

See Also

`simulate.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 multiplicative quarterly seasonal ETS model with random parameters
model3 <- ets_model(seasonal = "M", frequency = 4)
# Simulate from each model and plot the results
library(forecast)
simulate(model1, 100) %>% plot()
simulate(model2, 100) %>% plot()
simulate(model3, 100) %>% plot()
```

`generate.mar` *Generate a tsibble of synthetic data from a Mixture Autoregressive model*

Description

This function simulates multiple random sample paths from a mixture of k Gaussian AR(p) processes. The model is of the form

$$y_t = \phi_{0,i} + \phi_{1,i} y_{t-1} + \cdots + \phi_{p,i} y_{t-p} + \sigma_{i,t} \epsilon_t$$

with probability α_i, where ϵ_t is a N(0,1) variate. The index of the tsibble is guessed from the MAR model seasonal periods.

Usage

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

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

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

Arguments

`x`	A ‘mar‘ object, usually the output of `mar_model()`.
`length`	length of series to generate
`nseries`	number of series to generate
`...`	Other arguments, passed to `simulate.mar`.

Value

‘tsibble‘ object with ‘length‘ rows and 3 columns.

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, simulate.mar

Examples

```
# MAR model with constant variances
phi <- cbind(c(0, 0.8, 0), c(0, 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)
```

---

generate_msts *Generate multiple seasonal time series from random parameter spaces of the mixture autoregressive (MAR) models.*

Description

Deprecated function. Please use mar_model() and generate.mar() instead. Generates multiple seasonal time series from random parameter spaces of the mixture autoregressive (MAR) models.

Usage

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

Arguments

seasonal.periods	a vector of seasonal periods of the time series to be generated.
n	length of the generated time series.
nComp	number of mixing components when simulating time series using MAR models.
output_format	An optional argument which allows to choose output format between "list" and "tsibble"

Value

a time series with multiple seasonal periods.

Examples

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

`generate_ts` *Generate time series from random parameter spaces of the mixture autoregressive (MAR) models.*

Description

Deprecated function. Please use `mar_model()` and `generate.mar()` instead. Generate time series from random parameter spaces of the mixture autoregressive (MAR) models.

Usage

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

Arguments

`n.ts`	number of time series to be generated.
`freq`	seasonal period of the time series to be generated.
`nComp`	number of mixing components when simulating time series using MAR models.
`n`	length of the generated time series.
`output_format`	An optional argument which allows to choose output format between "list" and "tsibble"

Value

A list of time series together with the SARIMA coefficients used in each mixing component and the corresponding mixing weights.

Author(s)

Yanfei Kang and Feng Li

References

Wong, CS & WK Li (2000).

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` *Generating time series with controllable features.*

Description

Deprecated function. Please use `generate_target()` instead.

Usage

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

Arguments

`n`	number of time series to be generated.
`ts.length`	length of the time series to be generated.
`freq`	frequency of the time series to be generated.
`seasonal`	0 for non-seasonal data, 1 for single-seasonal data, and 2 for multiple seasonal data.
`features`	a vector of function names.
`selected.features`	selected features to be controlled.
`target`	target feature values.
`parallel`	An optional argument which allows to specify if the Genetic Algorithm should be run sequentially or in parallel.
`output_format`	An optional argument which allows to choose output format between 'list' and 'tsibble'

Value

A time-series object of class "ts" or "msts".

Author(s)

Yanfei Kang

Examples

```
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)
```

`mar_model` *Specify parameters for a Mixture Autoregressive model*

Description

This function allows the parameters of a mixture of k Gaussian ARIMA(p,d,0)(P,D,0)[m] processes to be specified. The output is used in `simulate.mar()` and `generate.mar`. The model is of the form

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

with probability α_i, where B is the backshift operator, m_i is the seasonal period, ϵ_t is a N(0,1) variate, and $\phi_i(B)$ and $\Phi_i(B)$ are polynomials in B of order d_i and D_i respectively. If any argument is `NULL`, the corresponding parameters are randomly selected. When randomly selected, the AR parameters are uniformly sampled from the stationary region, p is in {0,1,2,3}, d is in {0,1,2}, P is in {0,1,2} and D is in {0,1}. The model orders are uniformly sampled. The constants are uniformly sampled on (-3,3). The sigmas are uniformly sampled on (1,5) and the weights are uniformly sampled on (0,1). The number of components is uniformly sampled on {1,2,3,4,5}.

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	Number of components.
p	Non-negative integer vector giving the orders of non-seasonal AR polynomials $\phi_i(B)$. Ignored if `phi` provided.

`d` Non-negative integer vector giving the orders of non-seasonal differencing.

`phi` A max(p) x k numeric matrix containing the non-seasonal AR parameters $(\phi_{1,i}, \ldots, \phi_{p,i})$, $i = 1, \ldots, k$ for each component.

`P` Non-negative integer giving the orders of seasonal AR polynomiasl $\Phi_i(B)$. Ignored if `seasonal.periods==1` or `Phi` provided.

`D` Non-negative integer giving the orders of seasonal differencing. Ignored if `seasonal.periods==1`.

`Phi` A max(P) x k numeric matrix containing the seasonal AR parameters $(\Phi_{1,i}, \ldots, \phi_{P,i})$, $i = 1, \ldots, k$ for each component. Ignored if `seasonal.periods==1`.

`constants` A numeric vector of length k containing $c_1, \ldots, c_k$.

`sigmas` A numeric vector of length k or a list of k GARCH specifications. If it is a vector, it is assumed $\sigma_{i,t} = \sigma_i$ and `sigmas` = $\sigma_1, \ldots, \sigma_k$. If it is a list, each element should be the output from `fGarch::garchSpec()`.

`weights` A numeric vector of length k containing the probability of each of the component processes, $\alpha_1, \ldots, \alpha_k$.

`seasonal_periods`
Either a scalar or a numeric vector of length k containing the seasonal period of each component.

Value

A 'mar' object containing a list of `k`, `m`, `p`, `d`, `P`, `D`, `phi`, `Phi`, `sigmas` and `weights`.

Author(s)

Rob J Hyndman

See Also

`simulate.mar`

Examples

```
n <- 100
# 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
# containing an AR(1) component and an AR(2) component
phi <- cbind(c(0, 0.8, 0), c(0, 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
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 pi coefficients of an AR process from SARIMA coefficients.*

Description

Convert SARIMA coefficients to pi coefficients of an AR process.

Usage

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

Arguments

`ar`	AR coefficients in the SARIMA model.
`d`	number of differences in the SARIMA model.
`ma`	MA coefficients in the SARIMA model.
`sar`	seasonal AR coefficients in the SARIMA model.
`D`	number of seasonal differences in the SARIMA model.
`sma`	seasonal MA coefficients in the SARIMA model.
`m`	seasonal period in the SARIMA model.
`tol`	tolerance value used. Only return up to last element greater than tolerance.

Value

A vector of AR coefficients.

Author(s)

Rob J Hyndman

Examples

```
# Not Run
```

`rmixnorm` *Generate random variables from a mixture of multivariate normal distributions*

Description

Random variables from a mixture of k multivariate normal distributions, each of dimension q.

Usage

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

Arguments

`n`	an integer for the number of samples to be generated.
`means`	a q x k matrix (or a vector of length k if q=1) containing the means for each component.
`sigmas`	a q x q x k covariance array (or a vector of length k if q=1) for each component.
`weights`	a vector of length k containing the weights for each component.

Value

An n x q matrix (or a vector if q=1) containing the generated data.

Author(s)

Feng Li, Central University of Finance and Economics.

References

Villani et al 2009.

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 autoregressive random variables from mixture of normal*

Description

This function simulates random samples from a finite mixture of Gaussian distribution where the mean from each components are AR(p) process.

Usage

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

Arguments

`n`	number of samples.
`means.ar.par.list`	parameters in AR(p) within each mixing compoment.
`sigmas.list`	variance list.
`weights`	weight in each list.
`yinit`	initial values.

Value

vector of length n follows a mixture distribution.

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.

Examples

```
n <- 1000
means.ar.par.list <- list(c(0, 0.8), c(0, 0.6, 0.3))
require("fGarch")
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
)
```

```
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.mar` *Generate synthetic data from a Mixture Autoregressive model*

Description

This function simulates one random sample path from a mixture of k Gaussian AR(p) processes. The model is of the form

$$y_t = \phi_{0,i} + \phi_{1,i} y_{t-1} + \cdots + \phi_{p,i} y_{t-p} + \sigma_{i,t} \epsilon_t$$

with probability α_i, where ϵ_t is a N(0,1) variate.

Usage

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

Arguments

`object`	A 'mar' object, usually the output of `mar_model()`.
`nsim`	length of series to generate
`seed`	Either `NULL` or an integer that will be used in a call to `set.seed` before simulating the time series. The default, `NULL`, will not change the random generator state.
`n.start`	Length of 'burn-in' period.
`...`	Other arguments, not currently used.

Value

'ts' object of length `nsim`.

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, 0.8, 0), c(0, 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
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 *Generating time series with controllable features using MAR models*

Description

simulate_target simulate one time series of length 'length' from a MAR model with target features and returns a ts or msts object. generate_target simulate multiple time series of length 'length' from a MAR model with target features and returns a tsibble object. The index of the tsibble is guessed from the seasonal periods. The specified features should not depend on the scale of the time series as the series is scaled during simulation.

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	length of the time series to be generated.
seasonal_periods	Either a scalar or a numeric vector of length k containing the number of seasonal periods for each component.
feature_function	a function that returns a vector of features from a time series.
target	target feature values of the same length as that returned by feature_function().
k	integer specifying number of components to use for MAR models. Default is 3 unless there are multiple seasonal periods specified.
tolerance	average tolerance per feature. The genetic algorithm will attempt to find a solution where the average difference between the series features and target is less than tolerance. A larger value will give a faster but less precise solution.
trace	logical indicating if details of the search should be shown.
parallel	An optional argument which allows to specify if the Genetic Algorithm should be run sequentially or in parallel.
nseries	Number of series to generate

Value

A time series object of class "ts" or "msts".

Author(s)

Yanfei Kang and Rob J Hyndman

Examples

```
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)
## Not run:
# Generate a tsibble 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