Package ‘bssm’

June 9, 2020

Type Package

Title Bayesian Inference of Non-Gaussian State Space Models

Version 1.0.0

Date 2020-06-09

Description Efficient methods for Bayesian inference of state space models via particle Markov chain Monte Carlo (MCMC) and MCMC based on parallel importance sampling type weighted estimators (Vihola, Helske, and Franks, 2020, <arXiv:1609.02541>). Gaussian, Poisson, binomial, negative binomial, and Gamma observation densities and basic stochastic volatility models with Gaussian state dynamics, as well as general non-linear Gaussian models and discretised diffusion models are supported.

License GPL (>= 2)

Depends R (>= 3.1.3)

Suggests dplyr, ggplot2 (>= 2.0.0), Hmisc, KFAS (>= 1.2.1), knitr (>= 1.11), MASS, ramcmc, rmarkdown (>= 0.8.1), sde, sitmo, testthat

Imports coda (>= 0.18-1), diagis, Rcpp (>= 0.12.3)

LinkingTo Rcpp, RcppArmadillo, ramcmc, sitmo

SystemRequirements C++11

RoxygenNote 7.1.0

VignetteBuilder knitr

BugReports `https://github.com/helske/bssm/issues`

ByteCompile true

Encoding UTF-8

NeedsCompilation yes

Author Jouni Helske [aut, cre] (<https://orcid.org/0000-0001-7130-793X>), Matti Vihola [aut] (<https://orcid.org/0000-0002-8041-7222>)

Maintainer Jouni Helske `<jouni.helske@iki.fi>`

Repository CRAN

Date/Publication 2020-06-09 15:20:07 UTC

R topics documented:

ar1_lg — *Univariate Gaussian model with AR(1) latent process*

Description

Constructs a simple Gaussian model where the state dynamics follow an AR(1) process.

Usage

```
ar1_lg(y, rho, sigma, mu, sd_y, beta, xreg = NULL)
```

Arguments

`y`	Vector or a `ts` object of observations.
`rho`	prior for autoregressive coefficient.
`sigma`	Prior for the standard deviation of noise of the AR-process.
`mu`	A fixed value or a prior for the stationary mean of the latent AR(1) process. Parameter is omitted if this is set to 0.
`sd_y`	Prior for the standard deviation of observation equation.
`beta`	Prior for the regression coefficients.
`xreg`	Matrix containing covariates.

Value

Object of class `ar1_lg`.

ar1_ng — *Non-Gaussian model with AR(1) latent process*

Description

Constructs a simple non-Gaussian model where the state dynamics follow an AR(1) process.

Usage

```
ar1_ng(y, rho, sigma, mu, distribution, phi, u = 1, beta, xreg = NULL)
```

Arguments

y	Vector or a ts object of observations.
rho	prior for autoregressive coefficient.
sigma	Prior for the standard deviation of noise of the AR-process.
mu	A fixed value or a prior for the stationary mean of the latent AR(1) process. Parameter is omitted if this is set to 0.
distribution	distribution of the observation. Possible choices are "poisson", "binomial" and "negative binomial".
phi	Additional parameter relating to the non-Gaussian distribution. For Negative binomial distribution this is the dispersion term, and for other distributions this is ignored.
u	Constant parameter for non-Gaussian models. For Poisson and negative binomial distribution, this corresponds to the offset term. For binomial, this is the number of trials.
beta	Prior for the regression coefficients.
xreg	Matrix containing covariates.

Value

Object of class ar1_ng.

as.data.frame.mcmc_output

Convert MCMC chain to data.frame

Description

Converts the MCMC chain output of run_mcmc to data.frame.

Usage

```
## S3 method for class 'mcmc_output'
as.data.frame(
  x,
  row.names,
  optional,
  variable = c("theta", "states"),
  times,
  states,
  expand = !(x$mcmc_type %in% paste0("is", 1:3)),
  ...
)
```

Arguments

`x`	Output from `run_mcmc`.
`row.names`	Ignored.
`optional`	Ignored.
`variable`	Return samples of `"theta"` (default) or `"states"`?
`times`	Vector of indices. In case of states, what time points to expand? Default is all.
`states`	Vector of indices. In case of states, what states to expand? Default is all.
`expand`	Should the jump-chain be expanded? Defaults to `TRUE` for non-IS-MCMC, and `FALSE` for IS-MCMC. For `expand = FALSE` and always for IS-MCMC, the resulting data.frame contains variable weight (= counts times IS-weights).
`...`	Ignored.

`as_bssm` *Convert KFAS Model to bssm Model*

Description

Converts `SSModel` object of `KFAS` package to `bssm` model.

Usage

```
as_bssm(model, kappa = 100, ...)
```

Arguments

`model`	Object of class `SSModel`.
`kappa`	For `SSModel` object, a prior variance for initial state used to replace exact diffuse elements of the original model.
`...`	Additional arguments to `ssm_mlg` and `ssm_mng` (such as prior and updating functions).

Value

Object of class `ssm_mlg` or `ssm_mng`.

bootstrap_filter *Bootstrap Filtering*

Description

Function bootstrap_filter performs a bootstrap filtering with stratification resampling.

Usage

```
bootstrap_filter(model, nsim, ...)

## S3 method for class 'gaussian'
bootstrap_filter(
  model,
  nsim,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)

## S3 method for class 'nongaussian'
bootstrap_filter(
  model,
  nsim,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)

## S3 method for class 'ssm_nlg'
bootstrap_filter(
  model,
  nsim,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)

## S3 method for class 'ssm_sde'
bootstrap_filter(
  model,
  nsim,
  L,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

model	of class bsm_lg, bsm_ng or svm.

nsim	Number of samples.
...	Ignored.
seed	Seed for RNG.
L	Integer defining the discretization level for SDE models.

Value

A list containing samples, weights from the last time point, and an estimate of log-likelihood.

Examples

```
set.seed(1)
x <- cumsum(rnorm(50))
y <- rnorm(50, x, 0.5)
model <- bsm_lg(y, sd_y = 0.5, sd_level = 1, P1 = 1)

out <- bootstrap_filter(model, nsim = 1000)
ts.plot(cbind(y, x, out$att), col = 1:3)
ts.plot(cbind(kfilter(model)$att, out$att), col = 1:3)

data("poisson_series")
model <- bsm_ng(poisson_series, sd_level = 0.1, sd_slope = 0.01,
  P1 = diag(1, 2), distribution = "poisson")

out <- bootstrap_filter(model, nsim = 100)
ts.plot(cbind(poisson_series, exp(out$att[, 1])), col = 1:2)
```

---

bsm_lg *Basic Structural (Time Series) Model*

Description

Constructs a basic structural model with local level or local trend component and seasonal component.

Usage

```
bsm_lg(
  y,
  sd_y,
  sd_level,
  sd_slope,
  sd_seasonal,
  beta,
  xreg = NULL,
  period = frequency(y),
  a1,
```

```
  P1,
  D,
  C
)
```

Arguments

y	Vector or a ts object of observations.
sd_y	A fixed value or prior for the standard error of observation equation. See priors for details.
sd_level	A fixed value or a prior for the standard error of the noise in level equation. See priors for details.
sd_slope	A fixed value or a prior for the standard error of the noise in slope equation. See priors for details. If missing, the slope term is omitted from the model.
sd_seasonal	A fixed value or a prior for the standard error of the noise in seasonal equation. See priors for details. If missing, the seasonal component is omitted from the model.
beta	Prior for the regression coefficients.
xreg	Matrix containing covariates.
period	Length of the seasonal component i.e. the number of
a1	Prior means for the initial states (level, slope, seasonals). Defaults to vector of zeros.
P1	Prior covariance for the initial states (level, slope, seasonals). Default is diagonal matrix with 1000 on the diagonal.
D, C	Intercept terms for observation and state equations, given as a length n vector and m times n matrix respectively.

Value

Object of class bsm_lg.

Examples

```
prior <- uniform(0.1 * sd(log10(UKgas)), 0, 1)
model <- bsm_lg(log10(UKgas), sd_y = prior, sd_level =  prior,
  sd_slope =  prior, sd_seasonal =  prior)

mcmc_out <- run_mcmc(model, iter = 5000)
summary(expand_sample(mcmc_out, "theta"))$stat
mcmc_out$theta[which.max(mcmc_out$posterior), ]
sqrt((fit <- StructTS(log10(UKgas), type = "BSM"))$coef)[c(4, 1:3)]
```

bsm_ng *Non-Gaussian Basic Structural (Time Series) Model*

Description

Constructs a non-Gaussian basic structural model with local level or local trend component, a seasonal component, and regression component (or subset of these components).

Usage

```
bsm_ng(
  y,
  sd_level,
  sd_slope,
  sd_seasonal,
  sd_noise,
  distribution,
  phi,
  u = 1,
  beta,
  xreg = NULL,
  period = frequency(y),
  a1,
  P1,
  C
)
```

Arguments

`y`	Vector or a `ts` object of observations.
`sd_level`	A fixed value or a prior for the standard error of the noise in level equation. See priors for details.
`sd_slope`	A fixed value or a prior for the standard error of the noise in slope equation. See priors for details. If missing, the slope term is omitted from the model.
`sd_seasonal`	A fixed value or a prior for the standard error of the noise in seasonal equation. See priors for details. If missing, the seasonal component is omitted from the model.
`sd_noise`	Prior for the standard error of the additional noise term. See priors for details. If missing, no additional noise term is used.
`distribution`	distribution of the observation. Possible choices are `"poisson"`, `"binomial"`, `"negative binomial"`.
`phi`	Additional parameter relating to the non-Gaussian distribution. For Negative binomial distribution this is the dispersion term, and for other distributions this is ignored.

u	Constant parameter for non-Gaussian models. For Poisson and negative binomial distribution, this corresponds to the offset term. For binomial, this is the number of trials.
beta	Prior for the regression coefficients.
xreg	Matrix containing covariates.
period	Length of the seasonal component i.e. the number of observations per season. Default is `frequency(y)`.
a1	Prior means for the initial states (level, slope, seasonals). Defaults to vector of zeros.
P1	Prior covariance for the initial states (level, slope, seasonals). Default is diagonal matrix with 1e5 on the diagonal.
C	Intercept terms for state equation, given as a m times n matrix.

Value

Object of class `bsm_ng`.

Examples

```
model <- bsm_ng(Seatbelts[, "VanKilled"], distribution = "poisson",
  sd_level = halfnormal(0.01, 1),
  sd_seasonal = halfnormal(0.01, 1),
  beta = normal(0, 0, 10),
  xreg = Seatbelts[, "law"])
## Not run:
set.seed(123)
mcmc_out <- run_mcmc(model, iter = 5000, nsim = 10)
mcmc_out$acceptance_rate
theta <- expand_sample(mcmc_out, "theta")
plot(theta)
summary(theta)

library("ggplot2")
ggplot(as.data.frame(theta[,1:2]), aes(x = sd_level, y = sd_seasonal)) +
  geom_point() + stat_density2d(aes(fill = ..level.., alpha = ..level..),
  geom = "polygon") + scale_fill_continuous(low = "green", high = "blue") +
  guides(alpha = "none")

## End(Not run)
```

---

bssm *Bayesian Inference of State Space Models*

Description

This package contains functions for Bayesian inference of basic stochastic volatility model and exponential family state space models, where the state equation is linear and Gaussian, and the conditional observation density is either Gaussian, Poisson, binomial, negative binomial or Gamma density. General non-linear Gaussian models and models with continuous SDE dynamics are also supported. For formal definition of the currently supported models and methods, as well as some theory behind the IS-MCMC and ψ-APF, see the package vignette and arXiv paper: http://arxiv.org/abs/1609.02541.

---

drownings — *Deaths by drowning in Finland in 1969-2014*

Description

Dataset containing number of deaths by drowning in Finland in 1969-2014, yearly average summer temperatures (June to August) and corresponding population sizes (in hundreds of thousands).

Format

A time series object containing 46 observations and.

Source

Statistics Finland http://pxnet2.stat.fi/PXWeb/pxweb/en/StatFin/.

---

ekf — *(Iterated) Extended Kalman Filtering*

Description

Function `ekf` runs the (iterated) extended Kalman filter for the given non-linear Gaussian model of class `ssm_nlg`, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

```
ekf(model, iekf_iter = 0)
```

Arguments

`model`	Model model
`iekf_iter`	If `iekf_iter` > `0`, iterated extended Kalman filter is used with `iekf_iter` iterations.

Value

List containing the log-likelihood, one-step-ahead predictions `at` and filtered estimates `att` of states, and the corresponding variances `Pt` and `Ptt`.

ekf_smoother *Extended Kalman Smoothing*

Description

Function `ekf_smoother` runs the (iterated) extended Kalman smoother for the given non-linear Gaussian model of class `ssm_nlg`, and returns the smoothed estimates of the states and the corresponding variances.

Usage

```
ekf_smoother(model, iekf_iter = 0)
```

Arguments

`model`	Model model
`iekf_iter`	If `iekf_iter` > `0`, iterated extended Kalman filter is used with `iekf_iter` iterations.

Value

List containing the log-likelihood, smoothed state estimates `alphahat`, and the corresponding variances `Vt` and `Ptt`.

ekpf_filter *Extended Kalman Particle Filtering*

Description

Function `ekpf_filter` performs a extended Kalman particle filtering with stratification resampling, based on Van Der Merwe et al (2001).

Usage

```
ekpf_filter(object, nsim, ...)

## S3 method for class 'ssm_nlg'
ekpf_filter(object, nsim, seed = sample(.Machine$integer.max, size = 1), ...)
```

Arguments

`object`	of class `ssm_nlg`.
`nsim`	Number of samples.
`...`	Ignored.
`seed`	Seed for RNG.

Value

A list containing samples, filtered estimates and the corresponding covariances, weights from the last time point, and an estimate of log-likelihood.

References

Van Der Merwe, R., Doucet, A., De Freitas, N., & Wan, E. A. (2001). The unscented particle filter. In Advances in neural information processing systems (pp. 584-590).

---

exchange — *Pound/Dollar daily exchange rates*

Description

Dataset containing daily log-returns from 1/10/81-28/6/85 as in [1]

Format

A vector of length 945.

Source

http://www.ssfpack.com/DKbook.html.

References

James Durbin, Siem Jan Koopman (2012). "Time Series Analysis by State Space Methods". Oxford University Press.

---

expand_sample — *Expand the Jump Chain representation*

Description

The MCMC algorithms of `bssm` use a jump chain representation where we store the accepted values and the number of times we stayed in the current value. Although this saves bit memory and is especially convenient for IS-corrected MCMC, sometimes we want to have the usual sample paths. Function `expand_sample` returns the expanded sample based on the counts. Note that for IS-corrected output the expanded sample corresponds to the approximate posterior.

Usage

```
expand_sample(x, variable = "theta", times, states, by_states = TRUE)
```

Arguments

`x`	Output from `run_mcmc`.
`variable`	Expand parameters `"theta"` or states `"states"`.
`times`	Vector of indices. In case of states, what time points to expand? Default is all.
`states`	Vector of indices. In case of states, what states to expand? Default is all.
`by_states`	If `TRUE` (default), return list by states. Otherwise by time.

`fast_smoother` *Kalman Smoothing*

Description

Methods for Kalman smoothing of the states. Function `fast_smoother` computes only smoothed estimates of the states, and function `smoother` computes also smoothed variances.

Usage

```
fast_smoother(model, ...)

smoother(model, ...)
```

Arguments

`model`	Model model.
`...`	Ignored.

Details

For non-Gaussian models, the smoothing is based on the approximate Gaussian model.

Value

Matrix containing the smoothed estimates of states, or a list with the smoothed states and the variances.

`gaussian_approx` *Gaussian Approximation of Non-Gaussian/Non-linear State Space Model*

Description

Returns the approximating Gaussian model. This function is rarely needed itself, and is mainly available for testing and debugging purposes.

Usage

```
gaussian_approx(model, max_iter, conv_tol, ...)

## S3 method for class 'nongaussian'
gaussian_approx(model, max_iter = 100, conv_tol = 1e-08, ...)

## S3 method for class 'ssm_nlg'
gaussian_approx(model, max_iter = 100, conv_tol = 1e-08, iekf_iter = 0, ...)
```

Arguments

`model`	Model to be approximated.
`max_iter`	Maximum number of iterations.
`conv_tol`	Tolerance parameter.
`...`	Ignored.
`iekf_iter`	For non-linear models, number of iterations in iterated EKF (defaults to 0).

Examples

```
data("poisson_series")
model <- bsm_ng(y = poisson_series, sd_slope = 0.01, sd_level = 0.1,
  distribution = "poisson")
out <- gaussian_approx(model)
```

`importance_sample` *Importance Sampling from non-Gaussian State Space Model*

Description

Returns `nsim` samples from the approximating Gaussian model with corresponding (scaled) importance weights.

Usage

```
importance_sample(model, nsim, use_antithetic, max_iter, conv_tol, seed, ...)

## S3 method for class 'nongaussian'
importance_sample(
  model,
  nsim,
  use_antithetic = TRUE,
  max_iter = 100,
  conv_tol = 1e-08,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

`model`	of class `bsm_ng`, `ar1_ng` `svm`, `ssm_ung`, or `ssm_mng`.
`nsim`	Number of samples.
`use_antithetic`	Logical. If `TRUE` (default), use antithetic variable for location in simulation smoothing. Ignored for `ssm_mng` models.
`max_iter`	Maximum number of iterations used for the approximation.
`conv_tol`	Convergence threshold for the approximation. Approximation is claimed to be converged when the mean squared difference of the modes is less than `conv_tol`.
`seed`	Seed for the random number generator.
`...`	Ignored.

`kfilter` *Kalman Filtering*

Description

Function `kfilter` runs the Kalman filter for the given model, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

```
kfilter(model, ...)
```

Arguments

`model`	Model Model object.
`...`	Ignored.

Details

For non-Gaussian models, the filtering is based on the approximate Gaussian model.

Value

List containing the log-likelihood (approximate in non-Gaussian case), one-step-ahead predictions `at` and filtered estimates `att` of states, and the corresponding variances `Pt` and `Ptt`.

See Also

bootstrap_filter

`logLik.gaussian` *Log-likelihood of a Gaussian State Space Model*

Description

Computes the log-likelihood of the state space model of `bssm` package.

Computes the log-likelihood of the state space model of `bssm` package.

Usage

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

## S3 method for class 'nongaussian'
logLik(
  object,
  nsim,
  method = "psi",
  max_iter = 100,
  conv_tol = 1e-08,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

`object`	Model model.
`...`	Ignored.
`nsim`	Number of samples for particle filter or importance sampling. If 0, approximate log-likelihood based on the gaussian approximation is returned.
`method`	Sampling method, default is psi-auxiliary filter (`"psi"`), other choices are `"bsf"` bootstrap particle filter, and `"spdk"`, which uses the importance sampling approach by Shephard and Pitt (1997) and Durbin and Koopman (1997).
`max_iter`	Maximum number of iterations for gaussian approximation algorithm.
`conv_tol`	Tolerance parameter for the approximation algorithm.
`seed`	Seed for the random number generator.

Examples

```
model <- ssm_ulg(y = c(1,4,3), Z = 1, H = 1, T = 1, R = 1)
logLik(model)
model <- ssm_ung(y = c(1,4,3), Z = 1, T = 1, R = 0.5, P1 = 2,
  distribution = "poisson")

model2 <- bsm_ng(y = c(1,4,3), sd_level = 0.5, P1 = 2,
  distribution = "poisson")
logLik(model, nsim = 0)
logLik(model2, nsim = 0)
logLik(model, nsim = 10)
logLik(model2, nsim = 10)
```

---

logLik.ssm_nlg *Log-likelihood of a Non-linear State Space Model*

Description

Computes the log-likelihood of the state space model of `bssm` package.

Usage

```
## S3 method for class 'ssm_nlg'
logLik(
  object,
  nsim,
  method = "bsf",
  max_iter = 100,
  conv_tol = 1e-08,
  iekf_iter = 0,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

`object`	Model model.
`nsim`	Number of samples for particle filter. If 0, approximate log-likelihood is returned either based on the gaussian approximation or EKF, depending on the `method` argument.
`method`	Sampling method. Default is the bootstrap particle filter (`"bsf"`). Other choices are `"psi"` which uses psi-auxiliary filter (or approximating gaussian model in the case of `nsim = 0`), and `"ekf"` which uses EKF-based particle filter (or just EKF approximation in the case of `nsim = 0`).
`max_iter`	Maximum number of iterations for gaussian approximation algorithm.
`conv_tol`	Tolerance parameter for the approximation algorithm.

`iekf_iter`	If `iekf_iter` > `0`, iterated extended Kalman filter is used with `iekf_iter` iterations in place of standard EKF. Defaults to zero.
`seed`	Seed for the random number generator.
`...`	Ignored.

`logLik.ssm_sde` *Log-likelihood of a State Space Model with SDE dynamics*

Description

Computes the log-likelihood of the state space model of `bssm` package.

Usage

```
## S3 method for class 'ssm_sde'
logLik(object, nsim, L, seed = sample(.Machine$integer.max, size = 1), ...)
```

Arguments

`object`	Model model.
`nsim`	Number of samples for particle filter. If 0, approximate log-likelihood is returned either based on the gaussian approximation or EKF, depending on the `method` argument.
`L`	Integer defining the discretization level defined as (2^L).
`seed`	Seed for the random number generator.
`...`	Ignored.

`particle_smoother` *Particle Smoothing*

Description

Function `particle_smoother` performs filter-smoother or forward-backward smoother, using a either bootstrap filtering or psi-auxiliary filter with stratification resampling.

Usage

```
particle_smoother(model, nsim, ...)

## S3 method for class 'nongaussian'
particle_smoother(
  model,
  nsim,
  method = "psi",
  seed = sample(.Machine$integer.max, size = 1),
  max_iter = 100,
  conv_tol = 1e-08,
  ...
)

## S3 method for class 'ssm_nlg'
particle_smoother(
  model,
  nsim,
  method = "psi",
  seed = sample(.Machine$integer.max, size = 1),
  max_iter = 100,
  conv_tol = 1e-08,
  iekf_iter = 0,
  ...
)

## S3 method for class 'ssm_sde'
particle_smoother(
  model,
  nsim,
  L,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

`model`	Model.
`nsim`	Number of samples.
`...`	Ignored.
`method`	Choice of particle filter algorithm. For Gaussian and non-Gaussian models with linear dynamics, options are `"bsf"` (bootstrap particle filter) and `"psi"` (ψ-APF, the default), and for non-linear models options `"ekf"` (extended Kalman particle filter) is also available.
`seed`	Seed for RNG.
`max_iter`	Maximum number of iterations used in Gaussian approximation. Used ψ-APF.
`conv_tol`	Tolerance parameter used in Gaussian approximation. Used ψ-APF.

iekf_iter	If zero (default), first approximation for non-linear Gaussian models is obtained from extended Kalman filter. If iekf_iter > 0, iterated extended Kalman filter is used with iekf_iter iterations.
L	Integer defining the discretization level.

---

poisson_series *Simulated Poisson time series data*

Description

See example for code for reproducing the data.

Format

A vector of length 100

Examples

```
# The data is generated as follows:
set.seed(321)
slope <- cumsum(c(0, rnorm(99, sd = 0.01)))
y <- rpois(100, exp(cumsum(slope + c(0, rnorm(99, sd = 0.1)))))
```

---

predict.mcmc_output *Predictions for State Space Models*

Description

Draw samples from the posterior predictive distribution given the posterior draws of hyperparameters theta and alpha_n+1.

Usage

```
## S3 method for class 'mcmc_output'
predict(
  object,
  future_model,
  type = "response",
  seed = sample(.Machine$integer.max, size = 1),
  nsim,
  ...
)
```

Arguments

`object`	mcmc_output object obtained from `run_mcmc`
`future_model`	Model for future observations. Should have same structure as the original model which was used in MCMC, in order to plug the posterior samples of the model parameters to the right places.
`type`	Return predictions on `"mean"` `"response"`, or `"state"` level.
`seed`	Seed for RNG.
`nsim`	Number of samples to draw.
`...`	Ignored.

Value

Data frame of predicted samples.

Examples

```
require("graphics")
y <- log10(JohnsonJohnson)
prior <- uniform(0.01, 0, 1)
model <- bsm_lg(window(y, end = c(1974, 4)), sd_y = prior,
  sd_level = prior, sd_slope = prior, sd_seasonal = prior)

mcmc_results <- run_mcmc(model, iter = 5000)
future_model <- model
future_model$y <- ts(rep(NA, 25),
  start = tsp(model$y)[2] + 2 * deltat(model$y),
  frequency = frequency(model$y))
pred <- predict(mcmc_results, future_model, type = "state",
  nsim = 1000)

require("dplyr")
sumr_fit <- as.data.frame(mcmc_results, variable = "states") %>%
  group_by(time, iter) %>%
  mutate(signal =
      value[variable == "level"] +
      value[variable == "seasonal_1"]) %>%
  group_by(time) %>%
  summarise(mean = mean(signal),
    lwr = quantile(signal, 0.025),
    upr = quantile(signal, 0.975))

sumr_pred <- pred %>%
  group_by(time, sample) %>%
  mutate(signal =
      value[variable == "level"] +
      value[variable == "seasonal_1"]) %>%
  group_by(time) %>%
  summarise(mean = mean(signal),
    lwr = quantile(signal, 0.025),
    upr = quantile(signal, 0.975))
```

```
require("ggplot2")
rbind(sumr_fit, sumr_pred) %>%
  ggplot(aes(x = time, y = mean)) +
  geom_ribbon(aes(ymin = lwr, ymax = upr),
   fill = "#92f0a8", alpha = 0.25) +
  geom_line(colour = "#92f0a8") +
  theme_bw() +
  geom_point(data = data.frame(
    mean = log10(JohnsonJohnson),
    time = time(JohnsonJohnson)))
```

print.mcmc_output *Print Results from MCMC Run*

Description

Prints some basic summaries from the MCMC run by `run_mcmc`.

Usage

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

Arguments

x	Output from `run_mcmc`.
...	Ignored.

Details

In case of IS-corrected MCMC, the SE-IS is based only on importance sampling estimates, with weights corresponding to the block sizes of the jump chain multiplied by the importance correction weights (if IS-corrected method was used). These estimates ignore the possible autocorrelations but provide a lower-bound for the asymptotic standard error.

run_mcmc *Bayesian Inference of State Space Models*

Description

Adaptive Markov chain Monte Carlo simulation of state space models using Robust Adaptive Metropolis algorithm by Vihola (2012).

Usage

```
run_mcmc(model, iter, ...)
```

Arguments

model	State space model model of bssm package.
iter	Number of MCMC iterations.
...	Parameters to specific methods. See run_mcmc.gaussian and run_mcmc.nongaussian for details.

References

Matti Vihola (2012). "Robust adaptive Metropolis algorithm with coerced acceptance rate". Statistics and Computing, Volume 22, Issue 5, pages 997–1008. Matti Vihola, Jouni Helske, Jordan Franks (2020). "Importance sampling type estimators based on approximate marginal MCMC" ArXiv:1609.02541.

---

run_mcmc.gaussian *Bayesian Inference of Linear-Gaussian State Space Models*

Description

Bayesian Inference of Linear-Gaussian State Space Models

Usage

```
## S3 method for class 'gaussian'
run_mcmc(
  model,
  iter,
  output_type = "full",
  burnin = floor(iter/2),
  thin = 1,
  gamma = 2/3,
  target_acceptance = 0.234,
  S,
  end_adaptive_phase = TRUE,
  n_threads = 1,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

`model`	Model model.
`iter`	Number of MCMC iterations.
`output_type`	Type of output. Default is `"full"`, which returns samples from the posterior $p(\alpha, \theta)$. Option `"summary"` does not simulate states directly but computes the posterior means and variances of states using fast Kalman smoothing. This is slightly faster, more memory efficient and more accurate than calculations based on simulation smoother. Using option `"theta"` will only return samples from the marginal posterior of the hyperparameters θ.
`burnin`	Length of the burn-in period which is disregarded from the results. Defaults to `iter / 2`. Note that all MCMC algorithms of `bssm` used adaptive MCMC during the burn-in period in order to find good proposal.
`thin`	Thinning rate. All MCMC algorithms in `bssm` use the jump chain representation, and the thinning is applied to these blocks. Defaults to 1.
`gamma`	Tuning parameter for the adaptation of RAM algorithm. Must be between 0 and 1 (not checked).
`target_acceptance`	Target acceptance ratio for RAM. Defaults to 0.234.
`S`	Initial value for the lower triangular matrix of RAM algorithm, so that the covariance matrix of the Gaussian proposal distribution is SS'. Note that for some parameters (currently the standard deviation and dispersion parameters of bsm_lg models) the sampling is done for transformed parameters with internal_theta = log(theta).
`end_adaptive_phase`	If `TRUE` (default), S is held fixed after the burnin period.
`n_threads`	Number of threads for state simulation.
`seed`	Seed for the random number generator.
`...`	Ignored.

`run_mcmc.nongaussian` *Bayesian Inference of Non-Gaussian State Space Models*

Description

Methods for posterior inference of states and parameters.

Usage

```
## S3 method for class 'nongaussian'
run_mcmc(
  model,
  iter,
  nsim,
```

```
  output_type = "full",
  mcmc_type = "da",
  sampling_method = "psi",
  burnin = floor(iter/2),
  thin = 1,
  gamma = 2/3,
  target_acceptance = 0.234,
  S,
  end_adaptive_phase = TRUE,
  local_approx = TRUE,
  n_threads = 1,
  seed = sample(.Machine$integer.max, size = 1),
  max_iter = 100,
  conv_tol = 1e-08,
  ...
)
```

Arguments

`model`	Model model.
`iter`	Number of MCMC iterations.
`nsim`	Number of state samples per MCMC iteration. Ignored if `mcmc_type` is `"approx"`.
`output_type`	Either `"full"` (default, returns posterior samples of states alpha and hyperparameters theta), `"theta"` (for marginal posterior of theta), or `"summary"` (return the mean and variance estimates of the states and posterior samples of theta).
`mcmc_type`	What MCMC algorithm to use? Possible choices are `"pm"` for pseudo-marginal MCMC, `"da"` for delayed acceptance version of PMCMC (default), `"approx"` for approximate inference based on the Gaussian approximation of the model, or one of the three importance sampling type weighting schemes: `"is3"` for simple importance sampling (weight is computed for each MCMC iteration independently), `"is2"` for jump chain importance sampling type weighting, or `"is1"` for importance sampling type weighting where the number of particles used for weight computations is proportional to the length of the jump chain block.
`sampling_method`	If `"psi"`, ψ-auxiliary particle filter is used for state sampling (default). If `"spdk"`, non-sequential importance sampling based on Gaussian approximation is used. If `"bsf"`, bootstrap filter is used.
`burnin`	Length of the burn-in period which is disregarded from the results. Defaults to `iter / 2`.
`thin`	Thinning rate. Defaults to 1. Increase for large models in order to save memory. For IS-corrected methods, larger value can also be statistically more effective. Note: With `output_type = "summary"`, the thinning does not affect the computations of the summary statistics in case of pseudo-marginal methods.
`gamma`	Tuning parameter for the adaptation of RAM algorithm. Must be between 0 and 1 (not checked).

`target_acceptance`
: Target acceptance ratio for RAM. Defaults to 0.234.

`S`
: Initial value for the lower triangular matrix of RAM algorithm, so that the covariance matrix of the Gaussian proposal distribution is SS'. Note that for some parameters (currently the standard deviation and dispersion parameters of bsm_ng models) the sampling is done for transformed parameters with internal_theta = log(theta).

`end_adaptive_phase`
: If `TRUE` (default), S is held fixed after the burnin period.

`local_approx`
: If `TRUE` (default), Gaussian approximation needed for importance sampling is performed at each iteration. If false, approximation is updated only once at the start of the MCMC.

`n_threads`
: Number of threads for state simulation.

`seed`
: Seed for the random number generator.

`max_iter`
: Maximum number of iterations used in Gaussian approximation.

`conv_tol`
: Tolerance parameter used in Gaussian approximation.

`...`
: Ignored. set.seed(1) n <- 50 slope <- cumsum(c(0, rnorm(n - 1, sd = 0.001))) level <- cumsum(slope + c(0, rnorm(n - 1, sd = 0.2))) y <- rpois(n, exp(level)) poisson_model <- bsm_ng(y, sd_level = halfnormal(0.01, 1), sd_slope = halfnormal(0.01, 0.1), P1 = diag(c(10, 0.1)), distribution = "poisson") mcmc_is <- run_mcmc(poisson_model, iter = 1000, nsim = 10, mcmc_type = "is2") summary(mcmc_is, what = "theta", return_se = TRUE)

`run_mcmc.ssm_nlg` *Bayesian Inference of non-linear state space models*

Description

Methods for posterior inference of states and parameters.

Usage

```
## S3 method for class 'ssm_nlg'
run_mcmc(
  model,
  iter,
  nsim,
  output_type = "full",
  mcmc_type = "da",
  sampling_method = "bsf",
  burnin = floor(iter/2),
  thin = 1,
  gamma = 2/3,
  target_acceptance = 0.234,
  S,
```

```
  end_adaptive_phase = TRUE,
  n_threads = 1,
  seed = sample(.Machine$integer.max, size = 1),
  max_iter = 100,
  conv_tol = 1e-08,
  iekf_iter = 0,
  ...
)
```

Arguments

`model` Model model.

`iter` Number of MCMC iterations.

`nsim` Number of state samples per MCMC iteration. Ignored if `mcmc_type` is `"approx"` or `"ekf"`.

`output_type` Either `"full"` (default, returns posterior samples of states alpha and hyperparameters theta), `"theta"` (for marginal posterior of theta), or `"summary"` (return the mean and variance estimates of the states and posterior samples of theta).

`mcmc_type` What MCMC algorithm to use? Possible choices are `"pm"` for pseudo-marginal MCMC, `"da"` for delayed acceptance version of PMCMC (default), `"approx"` for approximate inference based on the Gaussian approximation of the model, `"ekf"` for approximate inference using extended Kalman filter, or one of the three importance sampling type weighting schemes: `"is3"` for simple importance sampling (weight is computed for each MCMC iteration independently), `"is2"` for jump chain importance sampling type weighting, or `"is1"` for importance sampling type weighting where the number of particles used for weight computations is proportional to the length of the jump chain block.

`sampling_method` If `"psi"`, ψ-auxiliary particle filter is used for state sampling. If `"ekf"`, particle filter based on EKF-proposals are used. If `"bsf"` (default), bootstrap filter is used.

`burnin` Length of the burn-in period which is disregarded from the results. Defaults to `iter / 2`.

`thin` Thinning rate. Defaults to 1. Increase for large models in order to save memory. For IS-corrected methods, larger value can also be statistically more effective. Note: With `output_type = "summary"`, the thinning does not affect the computations of the summary statistics in case of pseudo-marginal methods.

`gamma` Tuning parameter for the adaptation of RAM algorithm. Must be between 0 and 1 (not checked).

`target_acceptance` Target acceptance ratio for RAM. Defaults to 0.234.

`S` Initial value for the lower triangular matrix of RAM algorithm, so that the covariance matrix of the Gaussian proposal distribution is SS'. Note that for some parameters (currently the standard deviation and dispersion parameters of bsm_ng models) the sampling is done for transformed parameters with internal_theta = log(theta).

end_adaptive_phase	If TRUE (default), S is held fixed after the burnin period.
n_threads	Number of threads for state simulation.
seed	Seed for the random number generator.
max_iter	Maximum number of iterations used in Gaussian approximation.
conv_tol	Tolerance parameter used in Gaussian approximation.
iekf_iter	If iekf_iter > 0, iterated extended Kalman filter is used with iekf_iter iterations in place of standard EKF. Defaults to zero.
...	Ignored.

---

run_mcmc.ssm_sde *Bayesian Inference of SDE*

Description

Methods for posterior inference of states and parameters.

Usage

```
## S3 method for class 'ssm_sde'
run_mcmc(
  model,
  iter,
  nsim,
  output_type = "full",
  mcmc_type = "da",
  L_c,
  L_f,
  burnin = floor(iter/2),
  thin = 1,
  gamma = 2/3,
  target_acceptance = 0.234,
  S,
  end_adaptive_phase = TRUE,
  n_threads = 1,
  seed = sample(.Machine$integer.max, size = 1),
  ...
)
```

Arguments

model	Model model.
iter	Number of MCMC iterations.
nsim	Number of state samples per MCMC iteration.

output_type	Either "full" (default, returns posterior samples of states alpha and hyperparameters theta), "theta" (for marginal posterior of theta), or "summary" (return the mean and variance estimates of the states and posterior samples of theta). If nsim = 0, this is argument ignored and set to "theta".
mcmc_type	What MCMC algorithm to use? Possible choices are "pm" for pseudo-marginal MCMC, "da" for delayed acceptance version of PMCMC (default), or one of the three importance sampling type weighting schemes: "is3" for simple importance sampling (weight is computed for each MCMC iteration independently), "is2" for jump chain importance sampling type weighting, or "is1" for importance sampling type weighting where the number of particles used for weight computations is proportional to the length of the jump chain block.
L_c, L_f	Integer values defining the discretization levels for first and second stages (defined as 2^L). For PM methods, maximum of these is used.
burnin	Length of the burn-in period which is disregarded from the results. Defaults to iter / 2.
thin	Thinning rate. Defaults to 1. Increase for large models in order to save memory. For IS-corrected methods, larger value can also be statistically more effective. Note: With output_type = "summary", the thinning does not affect the computations of the summary statistics in case of pseudo-marginal methods.
gamma	Tuning parameter for the adaptation of RAM algorithm. Must be between 0 and 1 (not checked).
target_acceptance	Target acceptance ratio for RAM. Defaults to 0.234.
S	Initial value for the lower triangular matrix of RAM algorithm, so that the covariance matrix of the Gaussian proposal distribution is SS'. Note that for some parameters (currently the standard deviation and dispersion parameters of bsm_ng models) the sampling is done for transformed parameters with internal_theta = log(theta).
end_adaptive_phase	If TRUE (default), S is held fixed after the burnin period.
n_threads	Number of threads for state simulation.
seed	Seed for the random number generator.
...	Ignored.

---

sim_smoother *Simulation Smoothing*

Description

Function sim_smoother performs simulation smoothing i.e. simulates the states from the conditional distribution $p(\alpha|y, \theta)$.

Usage

```
sim_smoother(model, nsim, seed, use_antithetic = FALSE, ...)

## S3 method for class 'gaussian'
sim_smoother(
  model,
  nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE,
  ...
)

## S3 method for class 'nongaussian'
sim_smoother(
  model,
  nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE,
  ...
)
```

Arguments

`model`	Model object.
`nsim`	Number of independent samples.
`seed`	Seed for the random number generator.
`use_antithetic`	Use an antithetic variable for location. Default is `FALSE`. Ignored for multivariate models.
`...`	Ignored.

Details

For non-Gaussian/non-linear models, the simulation is based on the approximating Gaussian model.

Value

An array containing the generated samples.

Examples

```
model <- bsm_lg(rep(NA, 50), sd_level = uniform(1,0,5), sd_y = uniform(1,0,5))
sim <- sim_smoother(model, 12)
ts.plot(sim[, 1, ])
```

ssm_mlg *General multivariate linear Gaussian state space models*

Description

Constructs an object of class ssm_mlg by defining the corresponding terms of the observation and state equation:

Usage

```
ssm_mlg(
  y,
  Z,
  H,
  T,
  R,
  a1,
  P1,
  init_theta = numeric(0),
  D,
  C,
  state_names,
  update_fn = default_update_fn,
  prior_fn = default_prior_fn
)
```

Arguments

y	Observations as multivariate time series or matrix with dimensions n x p.
Z	System matrix Z of the observation equation as p x m matrix or p x m x n array.
H	Lower triangular matrix H of the observation. Either a scalar or a vector of length n.
T	System matrix T of the state equation. Either a m x m matrix or a m x m x n array. UPDATE!!
R	Lower triangular matrix R the state equation. Either a m x k matrix or a m x k x n array.
a1	Prior mean for the initial state as a vector of length m.
P1	Prior covariance matrix for the initial state as m x m matrix.
init_theta	Initial values for the unknown hyperparameters theta.
D	Intercept terms for observation equation, given as a p x n matrix.
C	Intercept terms for state equation, given as m x n matrix.
state_names	Names for the states.

update_fn Function which returns list of updated model components given input vector theta. This function should take only one vector argument which is used to create list with elements named as Z, H T, R, a1, P1, D, and C, where each element matches the dimensions of the original model. If any of these components is missing, it is assumed to be constant wrt. theta.

prior_fn Function which returns log of prior density given input vector theta.

Details

$$y_t = D(t,\theta) + Z(t,\theta)\alpha_t + H(t,\theta)\epsilon_t, \text{(observation equation)}$$

$$\alpha_{t+1} = C(t,\theta) + T(t,\theta)\alpha_t + R(t,\theta)\eta_t, \text{(transition equation)}$$

where $\epsilon_t \sim N(0, I_p)$, $\eta_t \sim N(0, I_m)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other.

Value

Object of class ssm_mlg.

---

ssm_mng *General Non-Gaussian State Space Model*

Description

Constructs an object of class ssm_mng by defining the corresponding terms of the observation and state equation:

Usage

```
ssm_mng(
  y,
  Z,
  T,
  R,
  a1,
  P1,
  distribution,
  phi = 1,
  u = 1,
  init_theta = numeric(0),
  D,
  C,
  state_names,
  update_fn = default_update_fn,
  prior_fn = default_prior_fn
)
```

Arguments

y	Observations as multivariate time series or matrix with dimensions n x p.
Z	System matrix Z of the observation equation as p x m matrix or p x m x n array.
T	System matrix T of the state equation. Either a m x m matrix or a m x m x n array.
R	Lower triangular matrix R the state equation. Either a m x k matrix or a m x k x n array.
a1	Prior mean for the initial state as a vector of length m.
P1	Prior covariance matrix for the initial state as m x m matrix.
distribution	vector of distributions of the observed series. Possible choices are "poisson", "binomial", "negative binomial", "gamma", and "gaussian".
phi	Additional parameters relating to the non-Gaussian distributions. For negative binomial distribution this is the dispersion term, for gamma distribution this is the shape parameter, for gaussian this is standard deviation, and for other distributions this is ignored.
u	Constant parameter for non-Gaussian models. For Poisson, gamma, and negative binomial distribution, this corresponds to the offset term. For binomial, this is the number of trials.
init_theta	Initial values for the unknown hyperparameters theta.
D	Intercept terms for observation equation, given as p x n matrix.
C	Intercept terms for state equation, given as m x n matrix.
state_names	Names for the states.
update_fn	Function which returns list of updated model components given input vector theta. This function should take only one vector argument which is used to create list with elements named as Z, T, R, a1, P1, D, C, and phi, where each element matches the dimensions of the original model. If any of these components is missing, it is assumed to be constant wrt. theta.
prior_fn	Function which returns log of prior density given input vector theta.

Details

$$p^i(y^i_t|D_t + Z_t\alpha_t), \text{(observation equation)}$$

$$\alpha_{t+1} = C_t + T_t\alpha_t + R_t\eta_t, \text{(transition equation)}$$

where $\eta_t \sim N(0, I_k)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other, and $p^i(y_t|.)$ is either Poisson, binomial, gamma, gaussian, or negative binomial distribution for each observation series $i = 1, ..., k$.

Value

Object of class ssm_mng. UDPATE!!

ssm_nlg *General multivariate nonlinear Gaussian state space models*

Description

Constructs an object of class ssm_nlg by defining the corresponding terms of the observation and state equation:

Usage

```
ssm_nlg(
  y,
  Z,
  H,
  T,
  R,
  Z_gn,
  T_gn,
  a1,
  P1,
  theta,
  known_params = NA,
  known_tv_params = matrix(NA),
  n_states,
  n_etas,
  log_prior_pdf,
  time_varying = rep(TRUE, 4),
  state_names = paste0("state", 1:n_states)
)
```

Arguments

y	Observations as multivariate time series (or matrix) of length n.
Z, H, T, R	An external pointers for the C++ functions which define the corresponding model functions.
Z_gn, T_gn	An external pointers for the C++ functions which define the gradients of the corresponding model functions.
a1	Prior mean for the initial state as a vector of length m.
P1	Prior covariance matrix for the initial state as m x m matrix.
theta	Parameter vector passed to all model functions.
known_params	Vector of known parameters passed to all model functions.
known_tv_params	Matrix of known parameters passed to all model functions.
n_states	Number of states in the model.

n_etas	Dimension of the noise term of the transition equation.
log_prior_pdf	An external pointer for the C++ function which computes the log-prior density given theta.
time_varying	Optional logical vector of length 4, denoting whether the values of Z, H, T, and R vary with respect to time variable (given identical states). If used, this can speed up some computations.
state_names	Names for the states.

Details

$$y_t = Z(t, \alpha_t, \theta) + H(t, \theta)\epsilon_t, \text{(observation equation)}$$

$$\alpha_{t+1} = T(t, \alpha_t, \theta) + R(t, \theta)\eta_t, \text{(transition equation)}$$

where $\epsilon_t \sim N(0, I_p)$, $\eta_t \sim N(0, I_m)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other, and functions Z, H, T, R can depend on α_t and parameter vector θ.

Compared to other models, these general models need a bit more effort from the user, as you must provide the several small C++ snippets which define the model structure. See examples in the vignette.

Value

Object of class `ssm_nlg`.

---

ssm_sde *Univariate state space model with continuous SDE dynamics*

Description

Constructs an object of class `ssm_sde` by defining the functions for the drift, diffusion and derivative of diffusion terms of univariate SDE, as well as the log-density of observation equation. We assume that the observations are measured at integer times (missing values are allowed).

Usage

```
ssm_sde(
  y,
  drift,
  diffusion,
  ddiffusion,
  obs_pdf,
  prior_pdf,
  theta,
  x0,
  positive
)
```

Arguments

y	Observations as univariate time series (or vector) of length n.
drift, diffusion, ddiffusion	An external pointers for the C++ functions which define the drift, diffusion and derivative of diffusion functions of SDE.
obs_pdf	An external pointer for the C++ function which computes the observational log-density given the the states and parameter vector theta.
prior_pdf	An external pointer for the C++ function which computes the prior log-density given the parameter vector theta.
theta	Parameter vector passed to all model functions.
x0	Fixed initial value for SDE at time 0.
positive	If TRUE, positivity constraint is forced by abs in Millstein scheme.

Details

As in case of ssm_nlg models, these general models need a bit more effort from the user, as you must provide the several small C++ snippets which define the model structure. See SDE vignette for an example.

Value

Object of class ssm_sde.

---

ssm_ulg *General univariate linear-Gaussian state space models*

Description

Construct an object of class ssm_ulg by defining the corresponding terms of the observation and state equation:

Usage

```
ssm_ulg(
  y,
  Z,
  H,
  T,
  R,
  a1,
  P1,
  init_theta = numeric(0),
  D,
  C,
  state_names,
```

```
  update_fn = default_update_fn,
  prior_fn = default_prior_fn
)
```

Arguments

`y`	Observations as time series (or vector) of length n.
`Z`	System matrix Z of the observation equation as m x 1 or m x n matrix.
`H`	Vector of standard deviations. Either a scalar or a vector of length n.
`T`	System matrix T of the state equation. Either a m x m matrix or a m x m x n array.
`R`	Lower triangular matrix R the state equation. Either a m x k matrix or a m x k x n array.
`a1`	Prior mean for the initial state as a vector of length m.
`P1`	Prior covariance matrix for the initial state as m x m matrix.
`init_theta`	Initial values for the unknown hyperparameters theta.
`D`	Intercept terms for observation equation, given as a length n vector.
`C`	Intercept terms for state equation, given as m x n matrix.
`state_names`	Names for the states.
`update_fn`	Function which returns list of updated model components given input vector theta. This function should take only one vector argument which is used to create list with elements named as `Z`, `H` `T`, `R`, `a1`, `P1`, `D`, and `C`, where each element matches the dimensions of the original model. If any of these components is missing, it is assumed to be constant wrt. theta.
`prior_fn`	Function which returns log of prior density given input vector theta.

Details

$$y_t = X_t beta + D_t + Z_t \alpha_t + H_t \epsilon_t, \text{(observation equation)}$$

$$\alpha_{t+1} = C_t + T_t \alpha_t + R_t \eta_t, \text{(transition equation)}$$

where $\epsilon_t \sim N(0,1)$, $\eta_t \sim N(0, I_k)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other, X_t are fixed covariates and $beta$ contains the corresponding (known) coefficients.

Value

Object of class `ssm_ulg`.

ssm_ung *General univariate non-Gaussian state space model*

Description

Construct an object of class ssm_ung by defining the corresponding terms of the observation and state equation:

Usage

```
ssm_ung(
  y,
  Z,
  T,
  R,
  a1,
  P1,
  distribution,
  phi = 1,
  u = 1,
  init_theta = numeric(0),
  D,
  C,
  state_names,
  update_fn = default_update_fn,
  prior_fn = default_prior_fn
)
```

Arguments

y	Observations as time series (or vector) of length n.
Z	System matrix Z of the observation equation. Either a vector of length m, a m x n matrix, or object which can be coerced to such.
T	System matrix T of the state equation. Either a m x m matrix or a m x m x n array, or object which can be coerced to such.
R	Lower triangular matrix R the state equation. Either a m x k matrix or a m x k x n array, or object which can be coerced to such.
a1	Prior mean for the initial state as a vector of length m.
P1	Prior covariance matrix for the initial state as m x m matrix.
distribution	Distribution of the observed time series. Possible choices are "poisson", "binomial", "gamma", and "negative binomial".
phi	Additional parameter relating to the non-Gaussian distribution. For negative binomial distribution this is the dispersion term, for gamma distribution this is the shape parameter, and for other distributions this is ignored.

`u`	Constant parameter for non-Gaussian models. For Poisson, gamma, and negative binomial distribution, this corresponds to the offset term. For binomial, this is the number of trials.
`init_theta`	Initial values for the unknown hyperparameters theta.
`D`	Intercept terms D_t for the observations equation, given as a 1 x 1 or 1 x n matrix.
`C`	Intercept terms C_t for the state equation, given as a m times 1 or m times n matrix.
`state_names`	Names for the states.
`update_fn`	Function which returns list of updated model components given input vector theta. This function should take only one vector argument which is used to create list with elements named as `Z`, `T`, `R`, `a1`, `P1`, `D`, `C`, and `phi`, where each element matches the dimensions of the original model. If any of these components is missing, it is assumed to be constant wrt. theta.
`prior_fn`	Function which returns log of prior density given input vector theta.

Details

$$p(y_t|D_t + Z_t\alpha_t), \text{(observation equation)}$$

$$\alpha_{t+1} = C_t + T_t\alpha_t + R_t\eta_t, \text{(transition equation)}$$

where $\eta_t \sim N(0, I_k)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other, and $p(y_t|.)$ is either Poisson, binomial, gamma, or negative binomial distribution.

Value

Object of class `ssm_ung`.

`summary.mcmc_output` *Summary of MCMC object*

Description

This functions returns a list containing mean, standard deviations, standard errors, and effective sample size estimates for parameters and states.

Usage

```
## S3 method for class 'mcmc_output'
summary(object, return_se = FALSE, variable = "theta", only_theta = FALSE, ...)
```

Arguments

object	Output from run_mcmc
return_se	if FALSE (default), computation of standard errors and effective sample sizes is omitted.
variable	Are the summary statistics computed for either "theta" (default), "states", or "both"?
only_theta	Deprecated. If TRUE, summaries are computed only for hyperparameters theta.
...	Ignored.

---

svm — *Stochastic Volatility Model*

Description

Constructs a simple stochastic volatility model with Gaussian errors and first order autoregressive signal.

Usage

```
svm(y, rho, sd_ar, sigma, mu)
```

Arguments

y	Vector or a ts object of observations.
rho	prior for autoregressive coefficient.
sd_ar	Prior for the standard deviation of noise of the AR-process.
sigma	Prior for sigma parameter of observation equation.
mu	Prior for mu parameter of transition equation. Ignored if sigma is provided.

Value

Object of class svm or svm2.

Examples

```
data("exchange")
exchange <- exchange[1:100] # faster CRAN check
model <- svm(exchange, rho = uniform(0.98,-0.999,0.999),
 sd_ar = halfnormal(0.15, 5), sigma = halfnormal(0.6, 2))

obj <- function(pars) {
   -logLik(svm(exchange, rho = uniform(pars[1],-0.999,0.999),
   sd_ar = halfnormal(pars[2],sd=5),
   sigma = halfnormal(pars[3],sd=2)), nsim = 0)
```

```
}
opt <- nlminb(c(0.98, 0.15, 0.6), obj, lower = c(-0.999, 1e-4, 1e-4), upper = c(0.999,10,10))
pars <- opt$par
model <- svm(exchange, rho = uniform(pars[1],-0.999,0.999),
  sd_ar = halfnormal(pars[2],sd=5),
  sigma = halfnormal(pars[3],sd=2))
```

ukf *Unscented Kalman Filtering*

Description

Function ukf runs the unscented Kalman filter for the given non-linear Gaussian model of class ssm_nlg, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

```
ukf(model, alpha = 1, beta = 0, kappa = 2)
```

Arguments

model	Model model
alpha, beta, kappa	Tuning parameters for the UKF.

Value

List containing the log-likelihood, one-step-ahead predictions at and filtered estimates att of states, and the corresponding variances Pt and Ptt.

uniform *Prior objects for bssm models*

Description

These simple objects of class bssm_prior are used to construct a prior distributions for the MCMC runs of bssm package. Currently supported priors are uniform (uniform()), half-normal (halfnormal()), normal (normal()), and truncated normal distribution (tnormal()).

Usage

```
uniform(init, min, max)

halfnormal(init, sd)

normal(init, mean, sd)

tnormal(init, mean, sd, min = -Inf, max = Inf)
```

Arguments

`init`	Initial value for the parameter, used in initializing the model components and as a starting value in MCMC.
`min`	Lower bound of the uniform and truncated normal prior.
`max`	Upper bound of the uniform and truncated normal prior.
`sd`	Standard deviation of the (underlying i.e. non-truncated) Normal distribution.
`mean`	Mean of the Normal prior.

Value

object of class `bssm_prior`.

Index