Package 'bssm'

January 7, 2020

Type Package

Title Bayesian Inference of Non-Linear and Non-Gaussian State Space Models

Version 0.1.8-1

Date 2020-01-07

Description Efficient methods for Bayesian inference of state space models via particle Markov chain Monte Carlo and parallel importance sampling type weighted Markov chain Monte Carlo (Vihola, Helske, and Franks, 2017, <arXiv:1609.02541>). Gaussian, Poisson, binomial, or negative binomial 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 bayesplot, KFAS (>= 1.2.1), knitr (>= 1.11), MASS, ramcmc, rmarkdown (>= 0.8.1), sde, sitmo, testthat

Imports coda (>= 0.18-1), diagis, ggplot2 (>= 2.0.0), Rcpp (>= 0.12.3)

LinkingTo BH, Rcpp, RcppArmadillo, ramcmc, sitmo

SystemRequirements C++11

RoxygenNote 6.1.1

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 2020-01-07 08:50:07 UTC

R topics documented:

ar1 *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(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.

as_gssm *Convert SSModel Object to gssm or ngssm Object*

Description

Converts SSModel object of KFAS package to gssm or ngssm object.

Usage

```
as_gssm(model, kappa = 1e+05, ...)

as_ngssm(model, kappa = 1e+05, phi_prior, ...)
```

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 gssm and ngssm.
phi_prior	For non-Gaussian model, prior for parameter phi.

Value

Object of class gssm or ngssm.

`autoplot.predict_bssm` *Plot predictions based on bssm package*

Description

Plot predictions based on bssm package

Usage

```
## S3 method for class 'predict_bssm'
autoplot(object, plot_mean = TRUE,
  plot_median = TRUE, y, fit, obs_color = "black",
  mean_color = "red", median_color = "blue", fit_color = mean_color,
  interval_color = "#000000", alpha_fill = 0.25, ...)
```

Arguments

object	Object of class predict_bssm.
plot_mean	Draw mean predictions. Default is TRUE.
plot_median	Draw median predictions. Default is TRUE.
y	Optional values for observations. Defaults to object$y.
fit	Optional values for fitted values such as smoothed estimates of past observations.
obs_color, mean_color, median_color, fit_color, interval_color	Colors for corresponding components of the plot.
alpha_fill	Alpha value for controlling the transparency of the intervals.
...	Ignored.

`bootstrap_filter` *Bootstrap Filtering*

Description

Function bootstrap_filter performs a bootstrap filtering with stratification resampling.

Usage

```
bootstrap_filter(object, nsim, ...)

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

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

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

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

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

## S3 method for class 'ng_ar1'
bootstrap_filter(object, nsim,
  seed = sample(.Machine$integer.max, size = 1), ...)

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

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

Arguments

object	of class bsm, ng_bsm 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.

bsm *Basic Structural (Time Series) Model*

Description

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

Usage

```
bsm(y, sd_y, sd_level, sd_slope, sd_seasonal, beta, xreg = NULL,
  period = frequency(y), a1, P1, obs_intercept, state_intercept)
```

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.
obs_intercept, state_intercept	Intercept terms for observation and state equations, given as a length n vector and m times n matrix respectively.

Value

Object of class bsm.

Examples

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

mcmc_out <- run_mcmc(model, n_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)]
```

---

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 are also supported. For formal definition of the currently supported models and methods, as well as theory, 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 nlg_ssm, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

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

Arguments

object	Model object
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 nlg_ssm, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

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

Arguments

object	Model object
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 'nlg_ssm'
ekpf_filter(object, nsim,
  seed = sample(.Machine$integer.max, size = 1), ...)
```

Arguments

`object`	of class `nlg_ssm`.
`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 `"state"`.
`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.
`...`	Ignored.

`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(object, ...)

smoother(object, ...)
```

Arguments

object	Model object.
...	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 state space model*

Description

Returns the approximating Gaussian model.

Usage

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

## S3 method for class 'ng_bsm'
gaussian_approx(object, max_iter = 100,
  conv_tol = 1e-08, ...)
```

Arguments

object	model object.
max_iter	Maximum number of iterations.
conv_tol	Tolerance parameter.
...	Ignored.

gssm *General univariate linear-Gaussian state space models*

Description

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

Usage

```
gssm(y, Z, H, T, R, a1, P1, xreg = NULL, beta, state_names, H_prior,
  Z_prior, T_prior, R_prior, obs_intercept, state_intercept)
```

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 or a m x n array, or an object which can be coerced to such.
H	Vector of standard deviations. Either a scalar or a vector of length n, or an 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.
xreg	Matrix containing covariates.
beta	Regression coefficients. Used as an initial value in MCMC. Defaults to vector of zeros.
state_names	Names for the states.
H_prior, Z_prior, T_prior, R_prior	Priors for the NA values in system matrices.
obs_intercept, state_intercept	Intercept terms for observation and state equations, given as a length n vector and m times n matrix respectively.

Details

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

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

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.

The priors are defined for each NA value of the system matrices, in the same order as these values are naturally read in R. For more flexibility, see lgg_ssm.

Value

Object of class gssm.

---

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(object, nsim, use_antithetic, max_iter, conv_tol, seed,
  ...)

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

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

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

## S3 method for class 'ung_ar1'
importance_sample(object, nsim, use_antithetic = TRUE,
  max_iter = 100, conv_tol = 1e-08,
  seed = sample(.Machine$integer.max, size = 1), ...)
```

Arguments

object	of class ng_bsm, svm or ngssm.
nsim	Number of samples.
use_antithetic	Logical. If TRUE (default), use antithetic variable for location in simulation smoothing.
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(object, ...)
```

Arguments

object	Model object
...	Ignored.

Details

For non-Gaussian models, the Kalman 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`

`lgg_ssm` *General multivariate linear Gaussian state space models*

Description

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

Usage

```
lgg_ssm(y, Z, H, T, R, a1, P1, theta, obs_intercept, state_intercept,
  known_params = NA, known_tv_params = matrix(NA), n_states, n_etas,
  log_prior_pdf, time_varying = rep(TRUE, 6),
  state_names = paste0("state", 1:n_states))
```

Arguments

y	Observations as multivariate time series (or matrix) of length n.
Z, H, T, R, a1, P1, obs_intercept, state_intercept	An external pointers for the C++ functions which define the corresponding model functions.
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 6, denoting whether the values of Z, H, T, R, D and C can vary with respect to time variable. If used, can speed up some computations.
state_names	Names for the states.

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.

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 `llg_ssm`.

---

logLik.gssm *Log-likelihood of the State Space Model*

Description

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

Usage

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

## S3 method for class 'ngssm'
logLik(object, nsim_states, method = "psi", seed = 1,
  max_iter = 100, conv_tol = 1e-08, ...)
```

Arguments

object	Model object.
...	Ignored.
nsim_states	Number of samples for importance sampling. If 0, approximate log-likelihood is returned. See vignette for details.
method	Method for computing the log-likelihood of non-Gaussian/non-linear model. Method "spdk" uses the importance sampling approach by Shephard and Pitt (1997), and Durbin and Koopman (1997). "psi" (the default for linear-Gaussian signals) uses psi-auxiliary filter and "bsf" bootstrap filter (default for general non-linear Gaussian models).
seed	Seed for the random number generator. Compared to other functions of the package, the default seed is fixed (as 1) in order to work properly in numerical optimization algorithms.
max_iter	Maximum number of iterations.
conv_tol	Tolerance parameter.

---

mv_gssm *General multivariate linear-Gaussian state space models*

Description

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

Usage

```
mv_gssm(y, Z, H, T, R, a1, P1, xreg = NULL, beta, state_names, H_prior,
  Z_prior, T_prior, R_prior, obs_intercept, state_intercept)
```

Arguments

y	Observations as multivariate time series (or matrix) of length n.
Z	System matrix Z of the observation equation. Either a p x m matrix or a p x m x n array, or an object which can be coerced to such.
H	Covarianc matrix for observational level noise.

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.
xreg	An array containing p covariate matrices with dimensions n x h.
beta	matrix of regression coefficients with n columns. Used as an initial value in MCMC. Defaults to matrix of zeros.
state_names	Names for the states.
H_prior, Z_prior, T_prior, R_prior	Priors for the NA values in system matrices.
obs_intercept, state_intercept	Intercept terms for observation and state equations, given as a p times n and m times n matrices.

Details

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

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

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

Value

Object of class mv_gssm.

---

ngssm *General univariate non-Gaussian/non-linear state space models*

Description

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

Usage

```
ngssm(y, Z, T, R, a1, P1, distribution, phi, u = 1, xreg = NULL, beta,
  state_names, Z_prior, T_prior, R_prior, state_intercept)
```

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 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.
xreg	Matrix containing covariates.
beta	Regression coefficients. Used as an initial value in MCMC. Defaults to vector of zeros.
state_names	Names for the states.
Z_prior, T_prior, R_prior	Priors for the NA values in system matrices.
state_intercept	Intercept terms for state equation, given as a m times n matrix.

Details

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

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

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 or negative binomial distribution.

Value

Object of class ngssm.

ng_ar1 *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

```
ng_ar1(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 ng_ar1.

ng_bsm *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

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

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" and "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.
state_intercept	Intercept terms for state equation, given as a m times n matrix.

Value

Object of class ng_bsm.

Examples

```
model <- ng_bsm(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, n_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)
```

---

nlg_ssm *General multivariate nonlinear Gaussian state space models*

Description

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

Usage

```
nlg_ssm(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 nlg_ssm.

---

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(object, nsim, ...)

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

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

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

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

Arguments

object	Model.
nsim	Number of samples.
...	Ignored.
seed	Seed for RNG.
filter_type	Choice of particle filter algorithm. For Gaussian models, only option is "bsf" (bootstrap particle filter). In addition, for non-Gaussian or non-linear models, "psi" uses psi-particle filter, and for non-linear models options "ekf" (extended Kalman particle filter) is also available.
max_iter	Maximum number of iterations used in Gaussian approximation. Used psi-PF.
conv_tol	Tolerance parameter used in Gaussian approximation. Used psi-PF.
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

Posterior intervals of future observations or their means (success probabilities in binomial case). These are computed using the quantile method where the intervals are computed as empirical quantiles the posterior sample, or using a parametric method by Helske (2016) in a linear-Gaussian case.

Usage

```
## S3 method for class 'mcmc_output'
predict(object, future_model, type = "response",
  intervals = TRUE, probs = c(0.05, 0.95), nsim, return_MCSE = FALSE,
  seed = sample(.Machine$integer.max, size = 1), ...)
```

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`	Compute predictions on `"mean"` ("confidence interval"), `"response"` ("prediction interval"), or `"state"` level. Defaults to `"response"`.
`intervals`	If `TRUE`, intervals are returned. Otherwise samples from the posterior predictive distribution are returned.
`probs`	Desired quantiles. Defaults to `c(0.05,0.95)`. Always includes median 0.5.
`nsim`	Number of state samples to draw per MCMC iteration. Note that this has no effect for the time point $n+1$ (where n is the length of the original series) as this is directly obtained from the MCMC output. `nsim` defaults to 1 except for the EKF based MCMC output of non-linear Gaussian models (see below). For linear-Gaussian models the intervals are computed based on Kalman filter so this argument has no effect if `intervals` is `TRUE`. For non-linear Gaussian models of class `nlg_ssm`, if `nsim` is 0 and `intervals` is `TRUE`, EKF based approximation is used for computing the prediction intervals.
`return_MCSE`	For Gaussian models, if `TRUE`, the Monte Carlo standard errors of the intervals are also returned.
`seed`	Seed for RNG.
`...`	Ignored.

Value

List containing the mean predictions, quantiles and Monte Carlo standard errors .

Examples

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

mcmc_results <- run_mcmc(model, n_iter = 5000)
future_model <- model
future_model$y <- ts(rep(NA, 25), start = end(model$y),
  frequency = frequency(model$y))
pred_gaussian <- predict(mcmc_results, future_model,
  probs = c(0.05, 0.1, 0.5, 0.9, 0.95))
ts.plot(log10(JohnsonJohnson), pred_gaussian$intervals,
  col = c(1, rep(2, 5)), lty = c(1, 2, 2, 1, 2, 2))

head(pred_gaussian$intervals)
head(pred_gaussian$MCSE)

# Non-gaussian models
## Not run:
data("poisson_series")

model <- ng_bsm(poisson_series, sd_level =
  halfnormal(0.1, 1), sd_slope=halfnormal(0.01, 0.1),
  distribution = "poisson")
mcmc_poisson <- run_mcmc(model, n_iter = 5000, nsim = 10)

future_model <- model
future_model$y <- ts(rep(NA, 25), start = end(model$y),
  frequency = frequency(model$y))

pred <- predict(mcmc_poisson, future_model,
  probs = seq(0.05,0.95, by = 0.05))

library("ggplot2")
fit <- ts(colMeans(exp(expand_sample(mcmc_poisson,
    "alpha")$level)))
autoplot(pred, y = model$y, fit = fit)

## End(Not run)
```

`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, two-types of standard error and effective sample size estimates are returned. SE-IS (ESS-IS) are 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. The SE-AR (ESS-AR) estimates are based on the spectral density of $(x-hatx)*w$ where $hatx$ is the weighted mean of x and w contains the weights.

---

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(object, n_iter, ...)
```

Arguments

object	State space model object of bssm package.
n_iter	Number of MCMC iterations.
...	Parameters to specific methods. See run_mcmc.gssm and run_mcmc.ngssm 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 (2016). "Importance sampling type correction of Markov chain Monte Carlo and exact approximations." ArXiv:1609.02541.

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

Description

Bayesian Inference of Linear-Gaussian State Space Models

Usage

```
## S3 method for class 'gssm'
run_mcmc(object, n_iter, type = "full",
  n_burnin = floor(n_iter/2), n_thin = 1, gamma = 2/3,
  target_acceptance = 0.234, S, end_adaptive_phase = TRUE,
  n_threads = 1, seed = sample(.Machine$integer.max, size = 1), ...)

## S3 method for class 'bsm'
run_mcmc(object, n_iter, type = "full",
  n_burnin = floor(n_iter/2), n_thin = 1, gamma = 2/3,
  target_acceptance = 0.234, S, end_adaptive_phase = TRUE,
  n_threads = 1, seed = sample(.Machine$integer.max, size = 1), ...)

## S3 method for class 'ar1'
run_mcmc(object, n_iter, type = "full",
  n_burnin = floor(n_iter/2), n_thin = 1, gamma = 2/3,
  target_acceptance = 0.234, S, end_adaptive_phase = TRUE,
  n_threads = 1, seed = sample(.Machine$integer.max, size = 1), ...)

## S3 method for class 'lgg_ssm'
run_mcmc(object, n_iter, type = "full",
  n_burnin = floor(n_iter/2), n_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

object	Model object.
n_iter	Number of MCMC iterations.
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, 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.
n_burnin	Length of the burn-in period which is disregarded from the results. Defaults to n_iter / 2. Note that all MCMC algorithms of bssm used adaptive MCMC during the burn-in period in order to find good proposal.

n_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 models) the sampling is done for transformed parameters with internal_theta = log(1 + 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.ngssm` *Bayesian inference of non-Gaussian or non-linear state space models using MCMC*

Description

Methods for posterior inference of states and parameters.

Usage

```
## S3 method for class 'ngssm'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", simulation_method = "psi",
  n_burnin = floor(n_iter/2), n_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, ...)

## S3 method for class 'ng_bsm'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", simulation_method = "psi",
  n_burnin = floor(n_iter/2), n_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, ...)
```

```
## S3 method for class 'ng_ar1'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", simulation_method = "psi",
  n_burnin = floor(n_iter/2), n_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, ...)

## S3 method for class 'svm'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", simulation_method = "psi",
  n_burnin = floor(n_iter/2), n_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, ...)

## S3 method for class 'nlg_ssm'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", simulation_method = "psi",
  n_burnin = floor(n_iter/2), n_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-04, iekf_iter = 0, ...)

## S3 method for class 'sde_ssm'
run_mcmc(object, n_iter, nsim_states, type = "full",
  method = "da", L_c, L_f, n_burnin = floor(n_iter/2), n_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

object	Model object.
n_iter	Number of MCMC iterations.
nsim_states	Number of state samples per MCMC iteration. If <2, approximate inference based on Gaussian approximation is performed.
type	Either "full" (default), or "summary". The former produces samples of states whereas the latter gives the mean and variance estimates of the states.
method	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.

simulation_method
: If "spdk", non-sequential importance sampling based on Gaussian approximation is used. If "bsf", bootstrap filter is used (default for "nlg_ssm" and only option for "sde_ssm"), and if "psi", psi-auxiliary particle filter is used (default for models with linear-Gaussian state equation).

n_burnin
: Length of the burn-in period which is disregarded from the results. Defaults to n_iter / 2.

n_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 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 models) the sampling is done for transformed parameters with internal_theta = log(1 + 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. Not used for non-linear models.

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. Used psi-PF.

conv_tol
: Tolerance parameter used in Gaussian approximation. Used psi-PF.

...
: Ignored.

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_c, L_f
: Integer values defining the discretization levels for first and second stages. For PM methods, maximum of these is used.

Examples

```
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 <- ng_bsm(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, n_iter = 1000, nsim_states = 10, method = "is2")
summary(mcmc_is, only_theta = TRUE, return_se = TRUE)
```

`sde_ssm` *Univariate state space model with continuous SDE dynamics*

Description

Constructs an object of class `sde_ssm` 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

```
sde_ssm(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 `nlg_ssm` 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 `sde_ssm`.

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(object, nsim, seed, use_antithetic = FALSE, ...)

## S3 method for class 'gssm'
sim_smoother(object, nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE, method = "dk", ...)

## S3 method for class 'bsm'
sim_smoother(object, nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE, method = "dk", ...)

## S3 method for class 'ar1'
sim_smoother(object, nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE, method = "dk", ...)

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

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

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

## S3 method for class 'ng_ar1'
sim_smoother(object, nsim = 1,
  seed = sample(.Machine$integer.max, size = 1),
  use_antithetic = FALSE, ...)
```

Arguments

object	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. Only used if method is "dk".
...	Ignored.
method	If "dk" (default), use simulation smoothing algorithm by Durbin and Koopman (2002). If "psi", use twisted SMC. Only used for Gaussian models of class "gssm", "bsm", and "ar1".

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

---

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,
  only_theta = FALSE, ...)
```

Arguments

object	Output from run_mcmc
return_se	if FALSE (default), computation of standard errors and effective sample sizes is omitted. This saves time, as computing the spectral densities (by coda) can be slow for large models.
only_theta	If TRUE, summaries are computed only for hyperparameters theta.
...	Ignored.

Details

Note that computing the state summaries can be slow for large models due to repeated calls to spectrum0.ar.

---

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_states = 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 nlg_ssm, and returns the filtered estimates and one-step-ahead predictions of the states α_t given the data up to time t.

Usage

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

Arguments

object	Model object
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, half-Normal and Normal distribution.

Usage

```
uniform(init, min, max)

halfnormal(init, sd)

normal(init, mean, sd)
```

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 prior .
max	Upper bound of the uniform prior.
sd	Standard deviation of the half-Normal and Normal priors.
mean	Mean of the Normal prior.

Value

object of class bssm_prior.

Note

Use Normal prior with care: currently there is no checks of non-negativity of standard deviation or stationarity of autoregressive coefficient in case Normal prior is used.

Index