Package ‘stochvol’

November 24, 2020

Encoding UTF-8

Type Package

Title Efficient Bayesian Inference for Stochastic Volatility (SV) Models

Version 3.0.3

Description Efficient algorithms for fully Bayesian estimation of stochastic volatility (SV) models with and without asymmetry (leverage) via Markov chain Monte Carlo (MCMC) methods. Methodological details are given in Kastner and Frühwirth-Schnatter (2014) <doi:10.1016/j.csda.2013.01.002> and Hosszejni and Kastner (2019) <doi:10.1007/978-3-030-30611-3_8>; the most common use cases are described in Kastner (2016) <doi:10.18637/jss.v069.i05> and the package vignette.

License GPL (>= 2)

Depends R (>= 3.5)

Imports Rcpp (>= 1.0), coda (>= 0.19), graphics, stats, utils, grDevices

Suggests mvtnorm, testthat (>= 2.3.2), knitr

LinkingTo Rcpp, RcppArmadillo (>= 0.9.900)

RoxygenNote 7.1.1

BuildResaveData best

VignetteBuilder knitr

NeedsCompilation yes

Author Darjus Hosszejni [aut, cre] (<https://orcid.org/0000-0002-3803-691X>), Gregor Kastner [aut] (<https://orcid.org/0000-0002-8237-8271>)

Maintainer Darjus Hosszejni <`darjus.hosszejni@wu.ac.at`>

Repository CRAN

Date/Publication 2020-11-24 15:30:02 UTC

R topics documented:

`stochvol-package` *Efficient Bayesian Inference for Stochastic Volatility (SV) Models*

Description

This package provides an efficient algorithm for fully Bayesian estimation of stochastic volatility (SV) models via Markov chain Monte Carlo (MCMC) methods. Methodological details are given in Kastner and Frühwirth-Schnatter (2014); the most common use cases are described in Kastner (2016). Recently, the package has been extended to allow for the leverage effect.

Details

Bayesian inference for stochastic volatility models using MCMC methods highly depends on actual parameter values in terms of sampling efficiency. While draws from the posterior utilizing the standard centered parameterization break down when the volatility of volatility parameter in the latent state equation is small, non-centered versions of the model show deficiencies for highly persistent latent variable series. The novel approach of ancillarity-sufficiency interweaving (Yu and Meng, 2011) has recently been shown to aid in overcoming these issues for a broad class of multilevel models. This package provides software for “combining best of different worlds” which allows for inference for parameter constellations that have previously been infeasible to estimate without the need to select a particular parameterization beforehand.

Note

This package is currently in active development. Your comments, suggestions and requests are warmly welcome!

Author(s)

Gregor Kastner <gregor.kastner@wu.ac.at>, Darjus Hosszejni <darjus.hosszejni@wu.ac.at>

References

Kastner, G. and Frühwirth-Schnatter, S. (2014). Ancillarity-Sufficiency Interweaving Strategy (ASIS) for Boosting MCMC Estimation of Stochastic Volatility Models. *Computational Statistics & Data Analysis*, **76**, 408–423, http://dx.doi.org/10.1016/j.csda.2013.01.002.

Kastner, G. (2016). Dealing with Stochastic Volatility in Time Series Using the R Package stochvol. *Journal of Statistical Software*, **69**(5), 1–30, http://dx.doi.org/10.18637/jss.v069.i05.

Yu, Y. and Meng, X.-L. (2011). To Center or Not to Center: That is Not the Question—An Ancillarity-Suffiency Interweaving Strategy (ASIS) for Boosting MCMC Efficiency. *Journal of Computational and Graphical Statistics*, **20**(3), 531–570, http://dx.doi.org/10.1198/jcgs.2011.203main.

Examples

```
## Simulate a highly persistent SV process
sim <- svsim(500, mu = -10, phi = 0.99, sigma = 0.2)

## Obtain 4000 draws from the sampler (that's too few!)
draws <- svsample(sim$y, draws = 4000, burnin = 100, priormu = c(-10, 1),
                 priorphi = c(20, 1.2), priorsigma = 0.2)

## Predict 20 days ahead
fore <- predict(draws, 20)

## plot the results
plot(draws, forecast = fore)

## Not run:
## Simulate an SV process with leverage
sim <- svsim(500, mu = -10, phi = 0.95, sigma = 0.2, rho=-0.5)

## Obtain 8000 draws from the sampler (that's too little!)
draws <- svsample(sim$y, draws = 4000, burnin = 3000, priormu = c(-10, 1),
                  priorphi = c(20, 1.2), priorsigma = 0.2,
                  priorrho = c(1, 1))

## Predict 20 days ahead
fore <- predict(draws, 20)
```

```
## plot the results
plot(draws, forecast = fore)

## End(Not run)
```

exrates *Euro exchange rate data*

Description

The data set contains the daily bilateral prices of one Euro in 23 currencies from January 3, 2000, until April 4, 2012. Conversions to New Turkish Lira and Fourth Romanian Leu have been incorporated.

Source

ECB Statistical Data Warehouse (http://sdw.ecb.europa.eu)

See Also

svsample

Examples

```
## Not run:
data(exrates)
dat <- logret(exrates$USD, demean = TRUE)  ## de-meaned log-returns
res <- svsample(dat)                       ## run MCMC sampler
plot(res, forecast = 100)                  ## display results

## End(Not run)
```

extractors *Common Extractors for 'svdraws' and 'svpredict' Objects*

Description

Some simple extractors returning the corresponding element of an `svdraws` and `svpredict` object.

Usage

```
para(x, chain = "concatenated")

latent0(x, chain = "concatenated")

latent(x, chain = "concatenated")

vola(x, chain = "concatenated")

svbeta(x, chain = "concatenated")

svtau(x, chain = "concatenated")

priors(x)

thinning(x)

runtime(x)

sampled_parameters(x)

predy(y, chain = "concatenated")

predlatent(y, chain = "concatenated")

predvola(y, chain = "concatenated")
```

Arguments

`x`	`svdraws` object.
`chain`	*optional* either a positive integer or the string `"concatenated"` (default) or the string `"all"`.
`y`	`svpredict` object.

Value

The return value depends on the actual funtion.

`para(x, chain = "concatenated")`	extracts the parameter draws.
`latent(x, chain = "concatenated")`	extracts the latent contemporaneous log-volatility draws.
`latent0(x, chain = "concatenated")`	extracts the latent initial log-volatility draws.
`svbeta(x, chain = "concatenated")`	extracts the linear regression coefficient draws.
`svtau(x, chain = "concatenated")`	extracts the tau draws.

`vola(x, chain = "concatenated")`
: extracts standard deviation draws.

`priors(x)`
: extracts the prior parameters used and returns them in a `prior_spec` object as generated by specify_priors.

`thinning(x)`
: extracts the thinning parameters used and returns them in a `list`.

`runtime(x)`
: extracts the runtime and returns it as a `proc_time` object.

`sampled_parameters(x)`
: returns the names of time independent model parameters that were actually sampled by `svsample`.

`predlatent(y, chain = "concatenated")`
: extracts the predicted latent contemporaneous log-volatility draws.

`predvola(y, chain = "concatenated")`
: extracts predicted standard deviation draws.

`predy(y, chain = "concatenated")`
: extracts the predicted observation draws.

Functions that have input parameter `chain` return an `mcmc.list` object if `chain=="all"` and return an `mcmc` object otherwise. If `chain` is an integer, then the specified chain is selected from all chains. If `chain` is `"concatenated"`, then all chains are merged into one `mcmc` object.

See Also

specify_priors, svsample, predict.svdraws

Examples

```
# Simulate data
sim <- svsim(150)

# Draw from vanilla SV
draws <- svsample(sim, draws = 2000)

## Summarize all parameter draws as a merged mcmc object
summary(para(draws))
## Extract the draws as an mcmc.list object
para(draws, chain = "all")


## Further short examples
summary(latent0(draws))
summary(latent(draws))
summary(vola(draws))
sampled_parameters(draws)
priors(draws)

# Draw 3 independent chains from heavy-tailed and asymmetric SV with AR(2) structure
draws <- svsample(sim, draws = 20000, burnin = 3000,
                  designmatrix = "ar2",
                  priornu = 0.1, priorrho = c(4, 4),
                  n_chains = 3)
```

```
## Extract beta draws from the second chain
svbeta(draws, chain = 2)
## ... tau draws from all chains merged/concatenated together
svtau(draws)
## Create a new svdraws object from the first and third chain
second_chain_excluded <- draws[c(1, 3)]

# Draw from the predictive distribution
pred <- predict(draws, steps = 2)

## Extract the predicted observations as an mcmc.list object
predicted_y <- predy(pred, chain = "all")
## ... the predicted standard deviations from the second chain
predicted_sd <- predvola(pred, chain = 2)
## Create a new svpredict object from the first and third chain
second_chain_excluded <- pred[c(1, 3)]
```

get_default_fast_sv *Default Values for the Expert Settings*

Description

These functions define meaningful expert settings for argument `expert` of `svsample` and its derivatives. The result of `get_default_fast_sv` should be provided as `expert$fast_sv` and `get_default_general_sv` as `expert$general_sv` when relevant.

Usage

```
get_default_fast_sv()

get_default_general_sv(priorspec)

default_fast_sv
```

Arguments

`priorspec`	a `priorspec` object created by `specify_priors`

Format

An object of class `list` of length 11.

See Also

`link{svsample}`, `specify_priors`, `svsample_roll`, `svsample_fast_cpp`, `svsample_general_cpp`

`logret` *Computes the Log Returns of a Time Series*

Description

`logret` computes the log returns of a time series, with optional de-meaning and/or standardization.

Usage

```
logret(dat, demean = FALSE, standardize = FALSE, ...)

## Default S3 method:
logret(dat, demean = FALSE, standardize = FALSE, ...)
```

Arguments

`dat`	The raw data.
`demean`	Logical value indicating whether the data should be de-meaned.
`standardize`	Logical value indicating whether the data should be standardized (in the sense that each component series has an empirical variance equal to one).
`...`	Ignored.

Value

Log returns of the (de-meaned / standardized) data.

Methods (by class)

- `default`: Log returns of vectors

`paradensplot` *Probability Density Function Plot for the Parameter Posteriors*

Description

Displays a plot of the density estimate for the posterior distribution of the parameters `mu`, `phi`, `sigma` (and potentially `nu` or `rho`), computed by the `density` function.

Usage

```
paradensplot(
  x,
  showobs = TRUE,
  showprior = TRUE,
  showxlab = TRUE,
  mar = c(1.9, 1.9, 1.9, 0.5),
  mgp = c(2, 0.6, 0),
  simobj = NULL,
  ...
)
```

Arguments

`x`	`svdraws` object.
`showobs`	logical value, indicating whether the observations should be displayed along the x-axis. If many draws have been obtained, the default (`TRUE`) can render the plotting to be quite slow, and you might want to try setting `showobs` to `FALSE`.
`showprior`	logical value, indicating whether the prior distribution should be displayed. The default value is `TRUE`.
`showxlab`	logical value, indicating whether the x-axis should be labelled with the number of iterations and the bandwith obtained from `density`. The default value is `TRUE`.
`mar`	numerical vector of length 4, indicating the plot margins. See `par` for details. The default value is `c(1.9,1.9,1.9,0.5)`, which is slightly smaller than the R-defaults.
`mgp`	numerical vector of length 3, indicating the axis and label positions. See `par` for details. The default value is `c(2,0.6,0)`, which is slightly smaller than the R-defaults.
`simobj`	object of class `svsim` as returned by the SV simulation function `svsim`. If provided, "true" data generating values will be added to the plots.
`...`	further arguments are passed on to the invoked `plot` function.

Details

`paradensplot` is modeled after `densplot` in the `coda` package, with some modifications for parameters that have (half-)bounded support.

Value

Called for its side effects. Returns argument `x` invisibly.

Note

You can call this function directly, but it is more commonly called by the `plot.svdraws` method.

See Also

Other plotting: `paratraceplot.svdraws()`, `paratraceplot()`, `plot.svdraws()`, `plot.svpredict()`, `volplot()`

`paratraceplot` *Trace Plot of MCMC Draws from the Parameter Posteriors*

Description

Generic function for plotting iterations vs. sampled parameter values. A detailed help for the method implemented in **stochvol** can be found in `paratraceplot.svdraws`.

Usage

```
paratraceplot(x, ...)
```

Arguments

`x`	An object used to select a method.
`...`	Further arguments passed to or from other methods.

Value

Called for its side effects. Returns argument `x` invisibly.

See Also

Other plotting: `paradensplot()`, `paratraceplot.svdraws()`, `plot.svdraws()`, `plot.svpredict()`, `volplot()`

`paratraceplot.svdraws` *Trace Plot of MCMC Draws from the Parameter Posteriors*

Description

Displays a plot of iterations vs. sampled values the parameters `mu`, `phi`, `sigma` (and potentially `nu` or `rho`), with a separate plot per variable.

Usage

```
## S3 method for class 'svdraws'
paratraceplot(
  x,
  mar = c(1.9, 1.9, 1.9, 0.5),
  mgp = c(2, 0.6, 0),
  simobj = NULL,
  ...
)
```

Arguments

`x`	`svdraws` object.
`mar`	numerical vector of length 4, indicating the plot margins. See `par` for details. The default value is `c(1.9,1.9,1.9,0.5)`, which is slightly smaller than the R-defaults.
`mgp`	numerical vector of length 3, indicating the axis and label positions. See `par` for details. The default value is `c(2,0.6,0)`, which is slightly smaller than the R-defaults.
`simobj`	object of class `svsim` as returned by the SV simulation function `svsim`. If provided, "true" data generating values will be added to the plots.
`...`	further arguments are passed on to the invoked `matplot` function.

Details

`paratraceplot` is modeled after `traceplot` in the coda package, with very minor modifications.

Value

Called for its side effects. Returns argument `x` invisibly.

Note

You can call this function directly, but it is more commonly called by the `plot.svdraws` method.

See Also

Other plotting: `paradensplot()`, `paratraceplot()`, `plot.svdraws()`, `plot.svpredict()`, `volplot()`

`plot.svdraws` *Graphical Summary of the Posterior Distribution*

Description

`plot.svdraws` and `plot.svldraws` generate some plots visualizing the posterior distribution and can also be used to display predictive distributions of future volatilities.

Usage

```
## S3 method for class 'svdraws'
plot(
  x,
  forecast = NULL,
  dates = NULL,
  show0 = FALSE,
  showobs = TRUE,
  showprior = TRUE,
  forecastlty = NULL,
  tcl = -0.4,
  mar = c(1.9, 1.9, 1.7, 0.5),
  mgp = c(2, 0.6, 0),
  simobj = NULL,
  newdata = NULL,
  ...
)
```

Arguments

`x`	`svdraws` object.
`forecast`	nonnegative integer or object of class `svpredict`, as returned by `predict.svdraws`. If an integer greater than 0 is provided, `predict.svdraws` is invoked to obtain the `forecast`-step-ahead prediction. The default value is `0`.
`dates`	vector of length `ncol(x$latent)`, providing optional dates for labelling the x-axis. The default value is `NULL`; in this case, the axis will be labelled with numbers.
`show0`	logical value, indicating whether the initial volatility `exp(h_0/2)` should be displayed. The default value is `FALSE`. Only available for inputs `x` of class `svdraws`.
`showobs`	logical value, indicating whether the observations should be displayed along the x-axis. If many draws have been obtained, the default (`TRUE`) can render the plotting to be quite slow, and you might want to try setting `showobs` to `FALSE`.
`showprior`	logical value, indicating whether the prior distribution should be displayed. The default value is `TRUE`.
`forecastlty`	vector of line type values (see `par`) used for plotting quantiles of predictive distributions. The default value `NULL` results in dashed lines.
`tcl`	The length of tick marks as a fraction of the height of a line of text. See `par` for details. The default value is `-0.4`, which results in slightly shorter tick marks than usual.
`mar`	numerical vector of length 4, indicating the plot margins. See `par` for details. The default value is `c(1.9,1.9,1.9,0.5)`, which is slightly smaller than the R-defaults.
`mgp`	numerical vector of length 3, indicating the axis and label positions. See `par` for details. The default value is `c(2,0.6,0)`, which is slightly smaller than the R-defaults.
`simobj`	object of class `svsim` as returned by the SV simulation function `svsim`. If provided, the "true" data generating values will be added to the plots.
`newdata`	corresponds to parameter `newdata` in `predict.svdraws`. *Only if* `forecast` *is a positive integer and* `predict.svdraws` *needs a* `newdata` *object.* Corresponds to input parameter `designmatrix` in `svsample`. A matrix of regressors with number of rows equal to parameter `forecast`.
`...`	further arguments are passed on to the invoked plotting functions.

Details

These functions set up the page layout and call `volplot`, `paratraceplot` and `paradensplot`.

Value

Called for its side effects. Returns argument `x` invisibly.

Note

In case you want different quantiles to be plotted, use `updatesummary` on the `svdraws` object first. An example of doing so is given in the Examples section.

Author(s)

Gregor Kastner `<gregor.kastner@wu.ac.at>`

See Also

`updatesummary`, `predict.svdraws`

Other plotting: `paradensplot()`, `paratraceplot.svdraws()`, `paratraceplot()`, `plot.svpredict()`, `volplot()`

Examples

```
## Simulate a short and highly persistent SV process
sim <- svsim(100, mu = -10, phi = 0.99, sigma = 0.2)

## Obtain 5000 draws from the sampler (that's not a lot)
draws <- svsample(sim$y, draws = 5000, burnin = 1000,
  priormu = c(-10, 1), priorphi = c(20, 1.5), priorsigma = 0.2)

## Plot the latent volatilities and some forecasts
plot(draws, forecast = 10)

## Re-plot with different quantiles
newquants <- c(0.01, 0.05, 0.25, 0.5, 0.75, 0.95, 0.99)
draws <- updatesummary(draws, quantiles = newquants)

plot(draws, forecast = 20, showobs = FALSE,
     forecastlty = 3, showprior = FALSE)
```

plot.svpredict *Graphical Summary of the Posterior Predictive Distribution*

Description

plot.svpredict and plot.svlpredict generate some plots visualizing the posterior predictive distribution of future volatilites and future observations.

Usage

```
## S3 method for class 'svpredict'
plot(x, quantiles = c(0.05, 0.25, 0.5, 0.75, 0.95), ...)
```

Arguments

x	svpredict or svlpredict object.
quantiles	Which quantiles to plot? Defaults to c(.05,.25,.5,.75,.95).
...	further arguments are passed on to the invoked ts.plot or boxplot function.

Value

Called for its side effects. Returns argument x invisibly.

Note

Note that svpredict or svlpredict objects can also be used within plot.svdraws for a possibly more useful visualization. See the examples in predict.svdraws and those below for use cases.

See Also

Other plotting: paradensplot(), paratraceplot.svdraws(), paratraceplot(), plot.svdraws(), volplot()

Other plotting: paradensplot(), paratraceplot.svdraws(), paratraceplot(), plot.svdraws(), volplot()

Examples

```
## Simulate a short and highly persistent SV process
sim <- svsim(100, mu = -10, phi = 0.99, sigma = 0.1)

## Obtain 5000 draws from the sampler (that's not a lot)
draws <- svsample(sim$y, draws = 5000, burnin = 1000)

## Predict 10 steps ahead
pred <- predict(draws, 10)

## Visualize the predicted distributions
```

```
plot(pred)

## Plot the latent volatilities and some forecasts
plot(draws, forecast = pred)
```

predict.svdraws — *Prediction of Future Returns and Log-Volatilities*

Description

Simulates draws from the predictive density of the returns and the latent log-volatility process. The same mean model is used for prediction as was used for fitting, which is either a) no mean parameter, b) constant mean, c) AR(k) structure, or d) general Bayesian regression. In the last case, new regressors need to be provided for prediction.

Usage

```
## S3 method for class 'svdraws'
predict(object, steps = 1L, newdata = NULL, ...)
```

Arguments

`object`	`svdraws` or `svldraws` object.
`steps`	*optional* single number, coercible to integer. Denotes the number of steps to forecast.
`newdata`	*only in case d) of the description* corresponds to input parameter `designmatrix` in `svsample`. A matrix of regressors with number of rows equal to parameter `steps`.
`...`	currently ignored.

Value

Returns an object of class `svpredict`, a list containing three elements:

`vol`	`mcmc.list` object of simulations from the predictive density of the standard deviations `sd_(n+1),...,sd_(n+steps)`
`h`	`mcmc.list` object of simulations from the predictive density of `h_(n+1),...,h_(n+steps)`
`y`	`mcmc.list` object of simulations from the predictive density of `y_(n+1),...,y_(n+steps)`

Note

You can use the resulting object within `plot.svdraws` (see example below), or use the list items in the usual `coda` methods for `mcmc` objects to print, plot, or summarize the predictions.

See Also

`plot.svdraws`, `volplot`.

Examples

```
# Example 1
## Simulate a short and highly persistent SV process
sim <- svsim(100, mu = -10, phi = 0.99, sigma = 0.2)

## Obtain 5000 draws from the sampler (that's not a lot)
draws <- svsample(sim$y, draws = 5000, burnin = 100,
  priormu = c(-10, 1), priorphi = c(20, 1.5), priorsigma = 0.2)

## Predict 10 days ahead
fore <- predict(draws, 10)

## Check out the results
summary(fore$h)
summary(fore$y)
plot(draws, forecast = fore)


# Example 2
## Simulate now an SV process with an AR(1) mean structure
len <- 109L
simar <- svsim(len, phi = 0.93, sigma = 0.15, mu = -9)
for (i in 2:len) {
  simar$y[i] <- 0.1 - 0.7 * simar$y[i-1] + simar$vol[i] * rnorm(1)
}

## Obtain 7000 draws
drawsar <- svsample(simar$y, draws = 7000, burnin = 300,
  designmatrix = "ar1", priormu = c(-10, 1), priorphi = c(20, 1.5),
  priorsigma = 0.2)

## Predict 7 days ahead (using AR(1) mean for the returns)
forear <- predict(drawsar, 7)

## Check out the results
plot(forear)
plot(drawsar, forecast = forear)

## Not run:
# Example 3
## Simulate now an SV process with leverage and with non-zero mean
len <- 96L
regressors <- cbind(rep_len(1, len), rgamma(len, 0.5, 0.25))
betas <- rbind(-1.1, 2)
simreg <- svsim(len, rho = -0.42)
simreg$y <- simreg$y + as.numeric(regressors %*% betas)

## Obtain 12000 draws
drawsreg <- svsample(simreg$y, draws = 12000, burnin = 3000,
  designmatrix = regressors, priormu = c(-10, 1), priorphi = c(20, 1.5),
  priorsigma = 0.2, priorrho = c(4, 4))
```

```
## Predict 5 days ahead using new regressors
predlen <- 5L
predregressors <- cbind(rep_len(1, predlen), rgamma(predlen, 0.5, 0.25))
forereg <- predict(drawsreg, predlen, predregressors)

## Check out the results
summary(forereg$h)
summary(forereg$y)
plot(forereg)
plot(drawsreg, forecast = forereg)

## End(Not run)
```

`specify_priors` *Specify Prior Distributions for SV Models*

Description

This function gives access to a larger set of prior distributions in case the default choice is unsatisfactory.

Usage

```
specify_priors(
  mu = sv_normal(mean = 0, sd = 100),
  phi = sv_beta(shape1 = 5, shape2 = 1.5),
  sigma2 = sv_gamma(shape = 0.5, rate = 0.5),
  nu = sv_infinity(),
  rho = sv_constant(0),
  latent0_variance = "stationary",
  beta = sv_multinormal(mean = 0, sd = 10000, dim = 1)
)
```

Arguments

`mu`	one of sv_normal and sv_constant
`phi`	one of sv_beta, sv_normal, and sv_constant. If sv_beta, then the specified beta distribution is the prior for (phi+1)/2
`sigma2`	one of sv_gamma, sv_inverse_gamma, and sv_constant
`nu`	one of sv_infinity, sv_exponential, and sv_constant. If sv_exponential, then the specified exponential distribution is the prior for nu-2
`rho`	one of sv_beta and sv_constant. If sv_beta, then the specified beta distribution is the prior for (rho+1)/2
`latent0_variance`	either the character string `"stationary"` or an sv_constant object. If `"stationary"`, then h0 ~ N(`mu, sigma^2/(1-phi^2)`). If an sv_constant object with value `v`, then h0 ~ N(`mu, v`). Here, N(b, B) stands for mean b and variance B
`beta`	an sv_multinormal object

See Also

Other priors: sv_constant()

`svsample` *Markov Chain Monte Carlo (MCMC) Sampling for the Stochastic Volatility (SV) Model*

Description

svsample simulates from the joint posterior distribution of the SV parameters mu, phi, sigma (and potentially nu and rho), along with the latent log-volatilities h_0,...,h_n and returns the MCMC draws. If a design matrix is provided, simple Bayesian regression can also be conducted.

Usage

```
svsample(
  y,
  draws = 10000,
  burnin = 1000,
  designmatrix = NA,
  priormu = c(0, 100),
  priorphi = c(5, 1.5),
  priorsigma = 1,
  priornu = 0,
  priorrho = NA,
  priorbeta = c(0, 10000),
  priorlatent0 = "stationary",
  priorspec = NULL,
  thin = 1,
  thinpara = thin,
  thinlatent = thin,
  keeptime = "all",
  quiet = FALSE,
  startpara = NULL,
  startlatent = NULL,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  n_chains = 1L,
  print_progress = "automatic",
  expert = NULL,
  ...
)

svtsample(
  y,
  draws = 10000,
  burnin = 1000,
  designmatrix = NA,
  priormu = c(0, 100),
  priorphi = c(5, 1.5),
  priorsigma = 1,
  priornu = 0.1,
  priorrho = NA,
  priorbeta = c(0, 10000),
  priorlatent0 = "stationary",
  priorspec = NULL,
  thin = 1,
  thinpara = thin,
  thinlatent = thin,
  keeptime = "all",
  quiet = FALSE,
  startpara = NULL,
  startlatent = NULL,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  n_chains = 1L,
  print_progress = "automatic",
  expert = NULL,
  ...
)

svlsample(
  y,
  draws = 20000,
  burnin = 2000,
  designmatrix = NA,
  priormu = c(0, 100),
  priorphi = c(5, 1.5),
  priorsigma = 1,
  priornu = 0,
  priorrho = c(4, 4),
  priorbeta = c(0, 10000),
  priorlatent0 = "stationary",
  priorspec = NULL,
  thin = 1,
  thinpara = thin,
  thinlatent = thin,
  keeptime = "all",
  quiet = FALSE,
  startpara = NULL,
  startlatent = NULL,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  n_chains = 1L,
  print_progress = "automatic",
  expert = NULL,
  ...
)

svtlsample(
  y,
  draws = 20000,
  burnin = 2000,
  designmatrix = NA,
  priormu = c(0, 100),
  priorphi = c(5, 1.5),
  priorsigma = 1,
  priornu = 0.1,
  priorrho = c(4, 4),
  priorbeta = c(0, 10000),
  priorlatent0 = "stationary",
  priorspec = NULL,
  thin = 1,
  thinpara = thin,
  thinlatent = thin,
  keeptime = "all",
  quiet = FALSE,
  startpara = NULL,
  startlatent = NULL,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  n_chains = 1L,
  print_progress = "automatic",
  expert = NULL,
  ...
)

svsample2(
  y,
  draws = 10000,
  burnin = 1000,
  designmatrix = NA,
  priormu = c(0, 100),
  priorphi = c(5, 1.5),
  priorsigma = 1,
  priornu = 0,
  priorrho = NA,
  priorbeta = c(0, 10000),
  priorlatent0 = "stationary",
  thinpara = 1,
  thinlatent = 1,
  keeptime = "all",
  quiet = FALSE,
  startpara = NULL,
  startlatent = NULL
)
```

Arguments

`y`	numeric vector containing the data (usually log-returns), which must not contain zeros. Alternatively, `y` can be an `svsim` object. In this case, the returns will be extracted and a message is signalled.
`draws`	single number greater or equal to 1, indicating the number of draws after burn-in (see below). Will be automatically coerced to integer. The default value is 10000.
`burnin`	single number greater or equal to 0, indicating the number of draws discarded as burn-in. Will be automatically coerced to integer. The default value is 1000.
`designmatrix`	regression design matrix for modeling the mean. Must have `length(y)` rows. Alternatively, `designmatrix` may be a string of the form `"arX"`, where `X` is a nonnegative integer. To fit a constant mean model, use `designmatrix = "ar0"` (which is equivalent to `designmatrix = matrix(1,nrow = length(y))`). To fit an AR(1) model, use `designmatrix = "ar1"`, and so on. If some elements of `designmatrix` are `NA`, the mean is fixed to zero (pre-1.2.0 behavior of **stochvol**).
`priormu`	numeric vector of length 2, indicating mean and standard deviation for the Gaussian prior distribution of the parameter `mu`, the level of the log-volatility. The default value is `c(0,100)`, which constitutes a practically uninformative prior for common exchange rate datasets, stock returns and the like.
`priorphi`	numeric vector of length 2, indicating the shape parameters for the Beta prior distribution of the transformed parameter `(phi + 1) / 2`, where `phi` denotes the persistence of the log-volatility. The default value is `c(5,1.5)`, which constitutes a prior that puts some belief in a persistent log-volatility but also encompasses the region where `phi` is around 0.
`priorsigma`	single positive real number, which stands for the scaling of the transformed parameter `sigma^2`, where `sigma` denotes the volatility of log-volatility. More precisely, `sigma^2 ~ priorsigma * chisq(df = 1)`. The default value is `1`, which constitutes a reasonably vague prior for many common exchange rate datasets, stock returns and the like.
`priornu`	single non-negative number, indicating the rate parameter for the exponential prior distribution of the parameter; can be `Inf` `nu`, the degrees-of-freedom parameter of the conditional innovations t-distribution. The default value is `0`, fixing the degrees-of-freedom to infinity. This corresponds to conditional standard normal innovations, the pre-1.1.0 behavior of **stochvol**.
`priorrho`	either `NA` for the no-leverage case or a numeric vector of length 2 that specify the beta prior distribution for `(rho+1)/2`

`priorbeta`	numeric vector of length 2, indicating the mean and standard deviation of the Gaussian prior for the regression parameters. The default value is `c(0,10000)`, which constitutes a very vague prior for many common datasets. Not used if `designmatrix` is `NA`.
`priorlatent0`	either a single non-negative number or the string `'stationary'` (the default, also the behavior before version 1.3.0). When `priorlatent0` is equal to `'stationary'`, the stationary distribution of the latent AR(1)-process is used as the prior for the initial log-volatility `h_0`. When `priorlatent0` is equal to a number B, we have $h_0 \sim N(\mu, B\sigma^2)$ a priori.
`priorspec`	in case one needs different prior distributions than the ones specified by `priormu`, `...`, `priorrho`, a `priorspec` object can be supplied here. A smart constructor for this usecase is specify_priors.
`thin`	single number greater or equal to 1, coercible to integer. Every `thinpara`th parameter and latent draw is kept and returned. The default value is 1, corresponding to no thinning of the parameter draws i.e. every draw is stored.
`thinpara`	single number greater or equal to 1, coercible to integer. Every `thinpara`th parameter draw is kept and returned. The default value is `thin`.
`thinlatent`	single number greater or equal to 1, coercible to integer. Every `thinlatent`th latent variable draw is kept and returned. The default value is `thin`
`keeptime`	Either 'all' (the default) or 'last'. Indicates which latent volatility draws should be stored.
`quiet`	logical value indicating whether the progress bar and other informative output during sampling should be omitted. The default value is `FALSE`, implying verbose output.
`startpara`	*optional* named list, containing the starting values for the parameter draws. If supplied, `startpara` must contain three elements named `mu`, `phi`, and `sigma`, where `mu` is an arbitrary numerical value, `phi` is a real number between `-1` and `1`, and `sigma` is a positive real number. Moreover, if `priornu` is not `0`, `startpara` must also contain an element named `nu` (the degrees of freedom parameter for the t-innovations). The default value is equal to the prior mean. In case of parallel execution with `cl` provided, `startpara` can be a list of named lists that initialize the parallel chains.
`startlatent`	*optional* vector of length `length(y)`, containing the starting values for the latent log-volatility draws. The default value is `rep(-10,length(y))`. In case of parallel execution with `cl` provided, `startlatent` can be a list of named lists that initialize the parallel chains.
`parallel`	*optional* one of `"no"` (default), `"multicore"`, or `"snow"`, indicating what type of parallellism is to be applied. Option `"multicore"` is not available on Windows.
`n_cpus`	*optional* positive integer, the number of CPUs to be used in case of parallel computations. Defaults to `1L`. Ignored if parameter `cl` is supplied and `parallel != "snow"`.
`cl`	*optional* so-called SNOW cluster object as implemented in package `parallel`. Ignored unless `parallel == "snow"`.
`n_chains`	*optional* positive integer specifying the number of independent MCMC chains

`print_progress`	*optional* one of `"automatic"`, `"progressbar"`, or `"iteration"`, controls the output. Ignored if `quiet` is `TRUE`.
`expert`	*optional* named list of expert parameters. For most applications, the default values probably work best. Interested users are referred to the literature provided in the References section. If `expert` is provided, it may contain the following named elements: • interweaveLogical value. If `TRUE` (the default), then ancillarity-sufficiency interweaving strategy (ASIS) is applied to improve on the sampling efficiency for the parameters. Otherwise one parameterization is used. • correct_model_misspecificationLogical value. If `FALSE` (the default), then auxiliary mixture sampling is used to sample the latent states. If `TRUE`, extra computations are made to correct for model misspecification either ex-post by reweighting or on-line using a Metropolis-Hastings step.
`...`	Any extra arguments will be forwarded to `updatesummary`, controlling the type of statistics calculated for the posterior draws.

Details

Functions `svtsample`, `svlsample`, and `svtlsample` are wrappers around `svsample` with convenient default values for the SV model with t-errors, leverage, and both t-errors and leverage, respectively.

For details concerning the algorithm please see the paper by Kastner and Frühwirth-Schnatter (2014).

Value

The value returned is a list object of class `svdraws` holding

`para`	`mcmc.list` object containing the *parameter* draws from the posterior distribution.
`latent`	`mcmc.list` object containing the *latent instantaneous log-volatility* draws from the posterior distribution.
`latent0`	`mcmc.list` object containing the *latent initial log-volatility* draws from the posterior distribution.
`tau`	`mcmc.list` object containing the *latent variance inflation factors* for the sampler with conditional t-innovations *(optional)*.
`beta`	`mcmc.list` object containing the *regression coefficient* draws from the posterior distribution *(optional)*.
`y`	the left hand side of the observation equation, usually the argument `y`. In case of an AR(k) specification, the first `k` elements are removed.
`runtime`	`proc_time` object containing the run time of the sampler.
`priors`	a `priorspec` object containing the parameter values of the prior distributions for `mu`, `phi`, `sigma`, `nu`, `rho`, and `betas`, and the variance of specification for `latent0`.
`thinning`	`list` containing the thinning parameters, i.e. the arguments `thinpara`, `thinlatent` and `keeptime`.

summary	list containing a collection of summary statistics of the posterior draws for para, latent, and latent0.
meanmodel	character containing information about how designmatrix was employed.

To display the output, use print, summary and plot. The print method simply prints the posterior draws (which is very likely a lot of output); the summary method displays the summary statistics currently stored in the object; the plot method plot.svdraws gives a graphical overview of the posterior distribution by calling volplot, traceplot and densplot and displaying the results on a single page.

Note

If y contains zeros, you might want to consider de-meaning your returns or use designmatrix = "ar0".

References

Kastner, G. and Frühwirth-Schnatter, S. (2014). Ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC estimation of stochastic volatility models. *Computational Statistics & Data Analysis*, **76**, 408–423, http://dx.doi.org/10.1016/j.csda.2013.01.002.

See Also

svsim, specify_priors

Examples

```
###############
# Full examples
###############

# Example 1
## Simulate a short and highly persistent SV process
sim <- svsim(100, mu = -10, phi = 0.99, sigma = 0.2)

## Obtain 5000 draws from the sampler (that's not a lot)
draws <-
  svsample(sim, draws = 5000, burnin = 100,
           priormu = c(-10, 1), priorphi = c(20, 1.5), priorsigma = 0.2)

## Check out the results
summary(draws)
plot(draws)



# Example 2
## Simulate an asymmetric and conditionally heavy-tailed SV process
sim <- svsim(150, mu = -10, phi = 0.96, sigma = 0.3, nu = 10, rho = -0.3)

## Obtain 10000 draws from the sampler
## Use more advanced prior settings
## Run two parallel MCMC chains
advanced_draws <-
  svsample(sim, draws = 10000, burnin = 5000,
           priorspec = specify_priors(mu = sv_normal(-10, 1),
                                      sigma2 = sv_gamma(0.5, 2),
                                      rho = sv_beta(4, 4),
                                      nu = sv_constant(5)),
           parallel = "snow", n_chains = 2, n_cpus = 2)

## Check out the results
summary(advanced_draws)
plot(advanced_draws)


# Example 3
## AR(1) structure for the mean
data(exrates)
len <- 3000
ahead <- 100
y <- head(exrates$USD, len)

## Fit AR(1)-SVL model to EUR-USD exchange rates
res <- svsample(y, designmatrix = "ar1")

## Use predict.svdraws to obtain predictive distributions
preddraws <- predict(res, steps = ahead)

## Calculate predictive quantiles
predquants <- apply(preddraws$y[[1]], 2, quantile, c(.1, .5, .9))

## Visualize
expost <- tail(head(exrates$USD, len+ahead), ahead)
ts.plot(y, xlim = c(length(y)-4*ahead, length(y)+ahead),
       ylim = range(c(predquants, expost, tail(y, 4*ahead))))
for (i in 1:3) {
  lines((length(y)+1):(length(y)+ahead), predquants[i,],
        col = 3, lty = c(2, 1, 2)[i])
}
lines((length(y)+1):(length(y)+ahead), expost,
      col = 2)


# Example 4
## Predicting USD based on JPY and GBP in the mean
data(exrates)
len <- 3000
ahead <- 30
## Calculate log-returns
logreturns <- apply(exrates[, c("USD", "JPY", "GBP")], 2,
                    function (x) diff(log(x)))
logretUSD <- logreturns[2:(len+1), "USD"]
regressors <- cbind(1, as.matrix(logreturns[1:len, ]))  # lagged by 1 day
```

```
## Fit SV model to EUR-USD exchange rates
res <- svsample(logretUSD, designmatrix = regressors)

## Use predict.svdraws to obtain predictive distributions
predregressors <- cbind(1, as.matrix(logreturns[(len+1):(len+ahead), ]))
preddraws <- predict(res, steps = ahead,
                     newdata = predregressors)
predprice <- exrates[len+2, "USD"] * exp(t(apply(preddraws$y[[1]], 1, cumsum)))

## Calculate predictive quantiles
predquants <- apply(predprice, 2, quantile, c(.1, .5, .9))

## Visualize
priceUSD <- exrates[3:(len+2), "USD"]
expost <- exrates[(len+3):(len+ahead+2), "USD"]
ts.plot(priceUSD, xlim = c(len-4*ahead, len+ahead+1),
       ylim = range(c(expost, predquants, tail(priceUSD, 4*ahead))))
for (i in 1:3) {
  lines(len:(len+ahead), c(tail(priceUSD, 1), predquants[i,]),
        col = 3, lty = c(2, 1, 2)[i])
}
lines(len:(len+ahead), c(tail(priceUSD, 1), expost),
      col = 2)


########################
# Further short examples
########################

y <- svsim(50, nu = 10, rho = -0.1)$y

# Supply initial values
res <- svsample(y,
                startpara = list(mu = -10, sigma = 1))


# Supply initial values for parallel chains
res <- svsample(y,
                startpara = list(list(mu = -10, sigma = 1),
                                 list(mu = -11, sigma = .1, phi = 0.9),
                                 list(mu = -9, sigma = .3, phi = 0.7)),
                parallel = "snow", n_chains = 3, n_cpus = 2)

# Parallel chains with with a pre-defined cluster object
cl <- parallel::makeCluster(2, "PSOCK", outfile = NULL)  # print to console
res <- svsample(y,
                parallel = "snow", n_chains = 3, cl = cl)
parallel::stopCluster(cl)


# Turn on correction for model misspecification
## Since the approximate model is fast and it is working very
##   well in practice, this is turned off by default
res <- svsample(y,
                expert = list(correct_model_misspecification = TRUE))
print(res)

## Not run:
# Parallel multicore chains (not available on Windows)
res <- svsample(y, draws = 30000, burnin = 10000,
                parallel = "multicore", n_chains = 3, n_cpus = 2)

# Plot using a color palette
palette(rainbow(coda::nchain(para(res, "all"))))
plot(res)

# Use functionality from package 'coda'
## E.g. Geweke's convergence diagnostics
coda::geweke.diag(para(res, "all")[, c("mu", "phi", "sigma")])

# Use functionality from package 'bayesplot'
bayesplot::mcmc_pairs(res, pars = c("sigma", "mu", "phi", "h_0", "h_15"))

## End(Not run)
```

`svsample_fast_cpp` *Bindings to* `C++` *Functions in* `stochvol`

Description

All the heavy lifting in `stochvol` is implemented in `C++` with the help of `R` packages `Rcpp` and `RcppArmadillo`. These functions call the MCMC samplers in `C++` directly without any any validation and transformations, expert use only!

Usage

```
svsample_fast_cpp(
  y,
  draws = 1,
  burnin = 0,
  designmatrix = matrix(NA),
  priorspec = specify_priors(),
  thinpara = 1,
  thinlatent = 1,
  keeptime = "all",
  startpara,
  startlatent,
  keeptau = !inherits(priorspec$nu, "sv_infinity"),
  print_settings = list(quiet = TRUE, n_chains = 1, chain = 1),
  correct_model_misspecification = FALSE,
  interweave = TRUE,
  myoffset = 0,
  fast_sv = get_default_fast_sv()
)

svsample_general_cpp(
  y,
  draws = 1,
  burnin = 0,
  designmatrix = matrix(NA),
  priorspec = specify_priors(),
  thinpara = 1,
  thinlatent = 1,
  keeptime = "all",
  startpara,
  startlatent,
  keeptau = !inherits(priorspec$nu, "sv_infinity"),
  print_settings = list(quiet = TRUE, n_chains = 1, chain = 1),
  correct_model_misspecification = FALSE,
  interweave = TRUE,
  myoffset = 0,
  general_sv = get_default_general_sv(priorspec)
)
```

Arguments

`y`	numeric vector of the observations
`draws`	single positive integer, the number of draws to return (after the burn-in)
`burnin`	single positive integer, length of warm-up period, this number of draws are discarded from the beginning
`designmatrix`	numeric matrix of covariates. Dimensions: `length(y)` times the number of covariates. If there are no covariates then this should be `matrix(NA)`
`priorspec`	a `priorspec` object created by `specify_priors`
`thinpara`	single number greater or equal to 1, coercible to integer. Every `thinpara`th parameter draw is kept and returned. The default value is 1, corresponding to no thinning of the parameter draws i.e. every draw is stored.
`thinlatent`	single number greater or equal to 1, coercible to integer. Every `thinlatent`th latent variable draw is kept and returned. The default value is 1, corresponding to no thinning of the latent variable draws, i.e. every draw is kept.
`keeptime`	Either 'all' (the default) or 'last'. Indicates which latent volatility draws should be stored.
`startpara`	named list, containing the starting values for the parameter draws. It must contain elements • mu: an arbitrary numerical value • phi: real number between `-1` and `1` • sigma: a positive real number

- nu: a number larger than `2`; can be `Inf`
- rho: real number between `-1` and `1`
- beta: a numeric vector of the same length as the number of covariates
- latent0: a single number, the initial value for `h0`

`startlatent` vector of length `length(y)`, containing the starting values for the latent log-volatility draws.

`keeptau` Logical value indicating whether the 'variance inflation factors' should be stored (used for the sampler with conditional t innovations only). This may be useful to check at what point(s) in time the normal disturbance had to be 'upscaled' by a mixture factor and when the series behaved 'normally'.

`print_settings` List of three elements:

- quiet: logical value indicating whether the progress bar and other informative output during sampling should be omitted
- n_chains: number of independent MCMC chains
- chain: index of this chain

Please note that this function does not run multiple independent chains but `svsample` offers different printing functionality depending on whether it is executed as part of several MCMC chains in parallel (chain specific messages) or simply as a single chain (progress bar).

`correct_model_misspecification`

Logical value. If `FALSE`, then auxiliary mixture sampling is used to sample the latent states. If `TRUE`, extra computations are made to correct for model misspecification either ex-post by reweighting or on-line using a Metropolis-Hastings step.

`interweave` Logical value. If `TRUE`, then ancillarity-sufficiency interweaving strategy (ASIS) is applied to improve on the sampling efficiency for the parameters. Otherwise one parameterization is used.

`myoffset` Single non-negative number that is used in `log(y^2 + myoffset)` to prevent `-Inf` values in the auxiliary mixture sampling scheme.

`fast_sv` named list of expert settings. We recommend the use of `get_default_fast_sv`.

`general_sv` named list of expert settings. We recommend the use of `get_default_general_sv`.

Details

The sampling functions are separated into fast SV and general SV. See more details in the sections below.

Fast SV

Fast SV was developed in Kastner and Fruehwirth-Schnatter (2014). Fast SV estimates an approximate SV model without leverage, where the approximation comes in through auxiliary mixture approximations to the exact SV model. The sampler uses the ancillarity-sufficiency interweaving strategy (ASIS) to improve on the sampling efficiency of the model parameters, and it employs all-without-a-loop (AWOL) for computationally efficient Kalman filtering of the conditionally Gaussian state space. Correction for model misspecification happens as a post-processing step.

Fast SV employs sampling strategies that have been fine-tuned and specified for vanilla SV (no leverage), and hence it can be fast and efficient but also more limited in its feature set. The conditions for the fast SV sampler: `rho == 0`; `mu` has either a normal prior or it is also constant `0`; the prior for `phi` is a beta distribution; the prior for `sigma^2` is either a gamma distribution with shape `0.5` or a mean- and variance-matched inverse gamma distribution; either `keeptime == 'all'` or `correct_model_misspecification == FALSE`. These criteria are NOT VALIDATED by fast SV on the `C++` level!

General SV

General SV also estimates an approximate SV model without leverage, where the approximation comes in through auxiliary mixture approximations to the exact SV model. The sampler uses both ASIS and AWOL.

General SV employs adapted random walk Metropolis-Hastings as the proposal for the parameters `mu`, `phi`, `sigma`, and `rho`. Therefore, more general prior distributions are allowed in this case.

Examples

```
# Draw one sample using fast SV and general SV
y <- svsim(40)$y
params <- list(mu = -10, phi = 0.9, sigma = 0.1,
               nu = Inf, rho = 0, beta = NA,
               latent0 = -10)
res_fast <- svsample_fast_cpp(y,
  startpara = params, startlatent = rep(-10, 40))
res_gen <- svsample_general_cpp(y,
  startpara = params, startlatent = rep(-10, 40))

# Embed SV in another sampling scheme
## vanilla SV
len <- 40L
draws <- 1000L
burnin <- 200L
param_store <- matrix(NA, draws, 3,
                      dimnames = list(NULL,
                                      c("mu", "phi", "sigma")))
startpara <- list(mu = 0, phi = 0.9, sigma = 0.1,
                  nu = Inf, rho = 0, beta = NA,
                  latent0 = 0)
startlatent <- rep(0, len)
for (i in seq_len(burnin+draws)) {
  # draw the data in the bigger sampling scheme
  # now we simulate y from vanilla SV
  y <- svsim(len, mu = 0, phi = 0.9, sigma = 0.1)$y
  # call SV sampler
  res <- svsample_fast_cpp(y, startpara = startpara,
                           startlatent = startlatent)
  # administrate values
  startpara[c("mu","phi","sigma")] <-
    as.list(res$para[, c("mu", "phi", "sigma")])
  startlatent <- drop(res$latent)
```

```
    # store draws after the burnin
    if (i > burnin) {
      param_store[i-burnin, ] <-
        res$para[, c("mu", "phi", "sigma")]
    }
  }
  ### quick look at the traceplots
  ts.plot(param_store, col = 1:3)
```

---

svsample_roll *Rolling Estimation of Stochastic Volatility Models*

Description

svsample_roll performs rolling window estimation based on svsample. A convenience function for backtesting purposes.

Usage

```
svsample_roll(
  y,
  designmatrix = NA,
  n_ahead = 1,
  forecast_length = 500,
  n_start = NULL,
  refit_every = 1,
  refit_window = c("moving", "expanding"),
  calculate_quantile = c(0.01),
  calculate_predictive_likelihood = TRUE,
  keep_draws = FALSE,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  ...
)

svtsample_roll(
  y,
  designmatrix = NA,
  n_ahead = 1,
  forecast_length = 500,
  n_start = NULL,
  refit_every = 1,
  refit_window = c("moving", "expanding"),
  calculate_quantile = c(0.01),
  calculate_predictive_likelihood = TRUE,
  keep_draws = FALSE,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  ...
)

svlsample_roll(
  y,
  designmatrix = NA,
  n_ahead = 1,
  forecast_length = 500,
  n_start = NULL,
  refit_every = 1,
  refit_window = c("moving", "expanding"),
  calculate_quantile = c(0.01),
  calculate_predictive_likelihood = TRUE,
  keep_draws = FALSE,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  ...
)

svtlsample_roll(
  y,
  designmatrix = NA,
  n_ahead = 1,
  forecast_length = 500,
  n_start = NULL,
  refit_every = 1,
  refit_window = c("moving", "expanding"),
  calculate_quantile = c(0.01),
  calculate_predictive_likelihood = TRUE,
  keep_draws = FALSE,
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1L,
  cl = NULL,
  ...
)
```

Arguments

`y`	numeric vector containing the data (usually log-returns), which must not contain zeros. Alternatively, `y` can be an `svsim` object. In this case, the returns will be extracted and a message is signalled.
`designmatrix`	regression design matrix for modeling the mean. Must have `length(y)` rows. Alternatively, `designmatrix` may be a string of the form `"arX"`, where `X` is a nonnegative integer. To fit a constant mean model, use `designmatrix = "ar0"`

	(which is equivalent to `designmatrix = matrix(1,nrow = length(y))`). To fit an AR(1) model, use `designmatrix = "ar1"`, and so on. If some elements of `designmatrix` are `NA`, the mean is fixed to zero (pre-1.2.0 behavior of **stochvol**).
`n_ahead`	number of time steps to predict from each time window.
`forecast_length`	the time horizon at the end of the data set that is used for backtesting.
`n_start`	*optional* the starting time point for backtesting. Computed from `forecast_length` if omitted.
`refit_every`	the SV model is refit every `refit_every` time steps. Only the value `1` is allowed.
`refit_window`	one of `"moving"` or `"expanding"`. If `"expanding"`, then the start of the time window stays at the beginning of the data set. If `"moving"`, then the length of the time window is constant throughout backtesting.
`calculate_quantile`	vector of numbers between 0 and 1. These quantiles are predicted using `predict.svdraws` for each time window.
`calculate_predictive_likelihood`	boolean. If `TRUE`, the `n_ahead` predictive density is evaluated at the `n_ahead` time observation after each time window.
`keep_draws`	boolean. If `TRUE`, the `svdraws` and the `svpredict` objects are kept from each time window.
`parallel`	one of `"no"` (default), `"multicore"`, or `"snow"`, indicating what type of parallellism is to be applied. Option `"multicore"` is not available on Windows.
`n_cpus`	*optional* positive integer, the number of CPUs to be used in case of parallel computations. Defaults to `1L`. Ignored if parameter `cl` is supplied and `parallel != "snow"`.
`cl`	*optional* so-called SNOW cluster object as implemented in package `parallel`. Ignored unless `parallel == "snow"`.
`...`	Any extra arguments will be forwarded to `svsample`, controlling the prior setup, the starting values for the MCMC chains, the number of independent MCMC chains, thinning and other expert settings.

Details

Functions `svtsample_roll`, `svlsample_roll`, and `svtlsample_roll` are wrappers around `svsample_roll` with convenient default values for the SV model with t-errors, leverage, and both t-errors and leverage, respectively.

Value

The value returned is a list object of class `svdraws_roll` holding a list item for every time window. The elements of these list items are

`indices`	a list object containing two elements: `train` is the vector of indices used for fitting the model, and `test` is the vector of indices used for prediction. The latter is mainly useful if a `designmatrix` is provided.
`quantiles`	the input parameter `calculate_quantiles`.

`refit_every`	the input parameter `refit_every`.
`predictive_likelihood`	present only if `calculate_predictive_likelihood` is `TRUE`. Then it is a number, the expected predictive density of the observation. The expecation is taken over the joint `n_ahead` predictive distribution of all model parameters.
`predictive_quantile`	present only if `calculate_quantile` is a non-empty vector. Then it is a vector of quantiles from the `n_ahead` predictive distribution of `y`. It is based on MCMC simulation by using `predict`.
`fit`	present only if `keep_draws` is `TRUE`. Then it is an `svdraws` object as returned by `svsample`.
`prediction`	present only if `keep_draws` is `TRUE`. Then it is an `svpredict` object as returned by `predict.svdraws`.

To display the output, use `print` and `summary`. The `print` method simply prints a short summary of the setup; the `summary` method displays the summary statistics of the backtesting.

Note

The function executes `svsample` `(length(y) -arorder -n_ahead -n_start + 2) %/% refit_every` times.

See Also

`svsim`, `specify_priors`, `svsample`

Examples

```
# Simulate from the true model
sim <- svsim(200)

# Perform rolling estimation using the vanilla SV
# model and default priors
roll <- svsample_roll(sim, draws = 5000, burnin = 2000,
                      keep_draws = TRUE,
                      forecast_length = 10,
                      n_ahead = 1, refit_every = 1,
                      refit_window = "moving",
                      calculate_predictive_likelihood = TRUE,
                      calculate_quantile = c(0.01, 0.05))

# Perform rolling estimation by making use
# of two CPU cores, advanced priors, and multiple
# chains with pre-set initial values. Let us combine
# that with an AR(2) specification
prior_beta <- sv_multinormal(c(1,0,-1), rbind(c(1, 0, 0.1),
                                              c(0, 0.3, -0.04),
                                              c(0.1, -0.04, 0.1)))
priorspec <- specify_priors(rho = sv_beta(4, 4),
                             latent0_variance = sv_constant(1),
                             beta = prior_beta,
                             nu = sv_exponential(0.05))
startpara <- list(list(mu = -9, phi = 0.3),
                  list(mu = -11, sigma = 0.1, phi = 0.95),
                  list(phi = 0.99))
roll <- svsample_roll(sim, draws = 5000, burnin = 2000,
                      designmatrix = "ar2",
                      priorspec = priorspec,
                      startpara = startpara,
                      parallel = "snow", n_cpus = 2,
                      n_chains = 3,
                      keep_draws = TRUE,
                      forecast_length = 10,
                      n_ahead = 1, refit_every = 1,
                      refit_window = "expanding",
                      calculate_predictive_likelihood = TRUE,
                      calculate_quantile = c(0.01, 0.05))
```

svsim — *Simulating a Stochastic Volatility Process*

Description

svsim is used to produce realizations of a stochastic volatility (SV) process.

Usage

```
svsim(len, mu = -10, phi = 0.98, sigma = 0.2, nu = Inf, rho = 0)
```

Arguments

len	length of the simulated time series.
mu	level of the latent log-volatility AR(1) process. The defaults value is -10.
phi	persistence of the latent log-volatility AR(1) process. The default value is 0.98.
sigma	volatility of the latent log-volatility AR(1) process. The default value is 0.2.
nu	degrees-of-freedom for the conditional innovations distribution. The default value is Inf, corresponding to standard normal conditional innovations.
rho	correlation between the observation and the increment of the log-volatility. The default value is 0, corresponding to the basic SV model with symmetric "log-returns".

Details

This function draws an initial log-volatility `h_0` from the stationary distribution of the AR(1) process defined by `phi`, `sigma`, and `mu`. Then the function jointly simulates the log-volatility series `h_1,...,h_n` with the given AR(1) structure, and the "log-return" series `y_1,...,y_n` with mean 0 and standard deviation `exp(h/2)`. Additionally, for each index `i`, `y_i` can be set to have a conditionally heavy-tailed residual (through `nu`) and/or to be correlated with `(h_{i+1}-h_i)` (through `rho`, the so-called leverage effect, resulting in asymmetric "log-returns").

Value

The output is a list object of class `svsim` containing

`y`	a vector of length `len` containing the simulated data, usually interpreted as "log-returns".
`vol`	a vector of length `len` containing the simulated instantaneous volatilities `exp(h_t/2)`.
`vol0`	The initial volatility `exp(h_0/2)`, drawn from the stationary distribution of the latent AR(1) process.
`para`	a named list with five elements `mu`, `phi`, `sigma`, `nu`, and `rho`, containing the corresponding arguments.

Note

The function generates the "log-returns" by `y <-exp(-h/2)*rt(h,df=nu)`. That means that in the case of `nu < Inf` the (conditional) volatility is `sqrt(nu/(nu-2))*exp(h/2)`, and that corrected value is shown in the `print`, `summary` and `plot` methods.

To display the output use `print`, `summary` and `plot`. The `print` method simply prints the content of the object in a moderately formatted manner. The `summary` method provides some summary statistics (in %), and the `plot` method plots the the simulated 'log-returns' `y` along with the corresponding volatilities `vol`.

Author(s)

Gregor Kastner <gregor.kastner@wu.ac.at>

See Also

`svsample`

Examples

```
## Simulate a highly persistent SV process of length 500
sim <- svsim(500, phi = 0.99, sigma = 0.1)

print(sim)
summary(sim)
plot(sim)

## Simulate an SV process with leverage
sim <- svsim(200, phi = 0.94, sigma = 0.15, rho = -0.6)
```

```
print(sim)
summary(sim)
plot(sim)

## Simulate an SV process with conditionally heavy-tails
sim <- svsim(250, phi = 0.91, sigma = 0.05, nu = 5)

print(sim)
summary(sim)
plot(sim)
```

sv_constant — *Prior Distributions in* stochvol

Description

The functions below can be supplied to `specify_priors` to overwrite the default set of prior distributions in `svsample`. The functions have `mean`, `range`, `density`, and `print` methods.

Usage

```
sv_constant(value)

sv_normal(mean = 0, sd = 1)

sv_multinormal(mean = 0, precision = NULL, sd = 1, dim = NA)

sv_gamma(shape, rate)

sv_inverse_gamma(shape, scale)

sv_beta(shape1, shape2)

sv_exponential(rate)

sv_infinity()
```

Arguments

`value`	The constant value for the degenerate constant distribution
`mean`	Expected value for the univariate normal distribution or mean vector of the multivariate normal distribution
`sd`	Standard deviation for the univariate normal distribution or constant scale of the multivariate normal distribution
`precision`	Precision matrix for the multivariate normal distribution

`dim`	(optional) Dimension of the multivariate distribution
`shape`	Shape parameter for the distribution
`rate`	Rate parameter for the distribution
`scale`	Scale parameter for the distribution
`shape1`	First shape parameter for the distribution
`shape2`	Second shape parameter for the distribution

Multivariate Normal

Multivariate normal objects can be specified several ways. The most general way is by calling `sv_multinormal(mean,precision)`, which allows for arbitrary mean and (valid) precision arguments. Constant mean vectors and constant diagonal precision matrices of dimension `D` can be created two ways: either `sv_multinormal(mean,sd,dim = D)` or `rep(sv_normal(mean,sd),length.out = D)`.

See Also

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

Other priors: `specify_priors()`

`updatesummary` *Updating the Summary of MCMC Draws*

Description

Creates or updates a summary of an `svdraws` object.

Usage

```
updatesummary(
  x,
  quantiles = c(0.05, 0.5, 0.95),
  esspara = TRUE,
  esslatent = FALSE
)
```

Arguments

`x`	`svdraws` object.
`quantiles`	numeric vector of posterior quantiles to be computed. The default is `c(0.05,0.5,0.95)`.
`esspara`	logical value which indicates whether the effective sample size (ESS) should be calculated for the *parameter draws*. This is achieved by calling `effectiveSize` from the `coda` package. The default is `TRUE`.
`esslatent`	logical value which indicates whether the effective sample size (ESS) should be calculated for the *latent log-volatility* draws. This is achieved by calling `effectiveSize` from the `coda` package. The default is `FALSE`, because this can be quite time-consuming when many latent variables are present.

Details

`updatesummary` will always calculate the posterior mean and the posterior standard deviation of the raw draws and some common transformations thereof. Moroever, the posterior quantiles, specified by the argument `quantiles`, are computed. If `esspara` and/or `esslatent` are `TRUE`, the corresponding effective sample size (ESS) will also be included.

Value

The value returned is an updated list object of class `svdraws` holding

`para`	`mcmc` object containing the *parameter* draws from the posterior distribution.
`latent`	`mcmc` object containing the *latent instantaneous log-volatility* draws from the posterior distribution.
`latent0`	`mcmc` object containing the *latent initial log-volatility* draws from the posterior distribution.
`y`	argument `y`.
`runtime`	`"proc_time"` object containing the run time of the sampler.
`priors`	`list` containing the parameter values of the prior distribution, i.e. the arguments `priormu`, `priorphi`, `priorsigma` (and potentially `nu`).
`thinning`	`list` containing the thinning parameters, i.e. the arguments `thinpara`, `thinlatent` and `keeptime`.
`summary`	`list` containing a collection of summary statistics of the posterior draws for `para`, `latent`, and `latent0`.

To display the output, use `print`, `summary` and `plot`. The `print` method simply prints the posterior draws (which is very likely a lot of output); the `summary` method displays the summary statistics currently stored in the object; the `plot` method gives a graphical overview of the posterior distribution by calling `volplot`, `traceplot` and `densplot` and displaying the results on a single page.

Note

`updatesummary` does not actually overwrite the object's current summary, but in fact creates a new object with an updated summary. Thus, don't forget to overwrite the old object if this is want you intend to do. See the examples below for more details.

See Also

svsample

Examples

```
## Here is a baby-example to illustrate the idea.
## Simulate an SV time series of length 51 with default parameters:
sim <- svsim(51)

## Draw from the posterior:
res <- svsample(sim$y, draws = 2000, priorphi = c(10, 1.5))

## Check out the results:
summary(res)
plot(res)

## Look at other quantiles and calculate ESS of latents:
newquants <- c(0.01, 0.05, 0.25, 0.5, 0.75, 0.95, 0.99)
res <- updatesummary(res, quantiles = newquants, esslatent = TRUE)

## See the difference?
summary(res)
plot(res)
```

---

update_fast_sv *Single MCMC Update Using Fast SV*

Description

Samples the mixture indicators, the latent variables, and the model independent parameters mu, phi, and sigma. The input is the logarithm of the squared de-meaned observations. An approximate SV model is estimated instead of the exact SV model by the use of auxiliary mixture sampling. Depending on the prior specification, mu might not be updated. Depending on the expert settings, the function might follow the ancillarity-sufficiency interweaving strategy (ASIS, Yu and Meng, 2011) for sampling mu, phi, and sigma. Furthermore, the user can turn off the sampling of the parameters, the latents, or the mixture indicators in the expert settings.

Usage

```
update_fast_sv(log_data2, mu, phi, sigma, h0, h, r, prior_spec, expert)
```

Arguments

log_data2	log(data^2), where data is the vector of de-meaned observations
mu	parameter mu. Level of the latent process h. Updated in place
phi	parameter phi, persistence of the latent process h. Updated in place

sigma	parameter sigma, volatility of the latent process h, also called volvol. Updated in place
h0	parameter h0, the initial value of the latent process h. Updated in place
h	the vector of the latent process. Updated in place
r	the vector of the mixture indicators. Updated in place
prior_spec	prior specification object. See type_definitions.h
expert	expert settings for this function. See type_definitions.h

See Also

Other stochvol_cpp: `update_general_sv()`, `update_regressors()`, `update_t_error()`

---

update_general_sv *Single MCMC Update Using General SV*

Description

Samples the latent variables and the model independent parameters mu, phi, sigma, and rho. The observations need to be provided in different formats for efficiency. An approximate SV model is as the default posterior distribution for the latent vector; however, there is the option to correct for model misspecification through the expert settings. Depending on the prior specification, some of mu, phi, sigma, and rho might not be updated. Depending on the expert settings, the function might follow the ancillarity-sufficiency interweaving strategy (ASIS, Yu and Meng, 2011) for sampling mu, phi, sigma, and rho. Also controlled by the expert settings, Furthermore, the user can turn off the sampling of the parameters, the latents, or the mixture indicators in the expert settings.

Usage

```
update_general_sv(
  data,
  log_data2,
  sign_data,
  mu,
  phi,
  sigma,
  rho,
  h0,
  h,
  adaptation,
  prior_spec,
  expert
)
```

Arguments

`data`	the vector of de-meaned observations
`log_data2`	log(data^2), where data is the vector of de-meaned observations
`sign_data`	the sign of the data
`mu`	parameter mu. Level of the latent process h. Updated in place
`phi`	parameter phi, persistence of the latent process h. Updated in place
`sigma`	parameter sigma, volatility of the latent process h, also called volvol. Updated in place
`rho`	parameter rho. Accounts for asymmetry/the leverage effect. Updated in place
`h0`	parameter h0, the initial value of the latent process h. Updated in place
`h`	the vector of the latent process. Updated in place
`adaptation`	object implementing the adaptive Metropolis-Hastings scheme. Updated in place. See adaptation.hpp
`prior_spec`	prior specification object. See type_definitions.h
`expert`	expert settings for this function. See type_definitions.h

See Also

Other stochvol_cpp: `update_fast_sv()`, `update_regressors()`, `update_t_error()`

`update_regressors` *Single MCMC update of Bayesian linear regression*

Description

Samples the coefficients of a linear regression. The prior distribution is multivariate normal and it is specified in prior_spec.

Usage

```
update_regressors(dependent_variable, independent_variables, beta, prior_spec)
```

Arguments

`dependent_variable`	the left hand side
`independent_variables`	the matrix of the independent variables. Has to be of same height as the length of the dependent variable
`beta`	the vector of the latent states used in MDA. Updated in place
`prior_spec`	prior specification object. See type_definitions.h

See Also

Other stochvol_cpp: `update_fast_sv()`, `update_general_sv()`, `update_t_error()`

`update_t_error` *Single MCMC update to Student's t-distribution*

Description

Samples the degrees of freedom parameter of standardized and homoskedastic t-distributed input variates. Marginal data augmentation (MDA) is applied, tau is the vector of auxiliary latent states. Depending on the prior specification, nu might not be updated, just tau.

Usage

```
update_t_error(
  homosked_data,
  tau,
  mean,
  sd,
  nu,
  prior_spec,
  do_tau_acceptance_rejection = TRUE
)
```

Arguments

`homosked_data`	de-meaned and homoskedastic observations
`tau`	the vector of the latent states used in MDA. Updated in place
`mean`	the vector of the conditional means // TODO update docs in R
`sd`	the vector of the conditional standard deviations
`nu`	parameter nu. The degrees of freedom for the t-distribution. Updated in place
`prior_spec`	prior specification object. See type_definitions.h
`do_tau_acceptance_rejection`	boolean. If `TRUE`, there is a correction for non-zero `mean` and non-unit `sd`, otherwise the proposal distribution is used

Details

The function samples tau and nu from the following hierarchical model: homosked_data_i = sqrt(tau_i) * (mean_i + sd_i * N(0, 1)) tau_i ~ InvGamma(.5*nu, .5*(nu-2)) Naming: The data is homoskedastic ex ante in the model, mean_i and sd_i are conditional on some other parameter in the model. The prior on tau corresponds to a standardized t-distributed heavy tail on the data.

See Also

Other stochvol_cpp: `update_fast_sv()`, `update_general_sv()`, `update_regressors()`

validate_and_process_expert *Validate and Process Argument 'expert'*

Description

A helper function that validates the input and extends it with default values if there are missing parts for argument 'expert'.

Usage

```
validate_and_process_expert(expert = NULL, priorspec = specify_priors())
```

Arguments

`expert`	list, the input values for expert.
`priorspec`	a `priorspec` object created by `specify_priors`

Value

A list that is the input extended by default values. If the input is invalid, an error is thrown.

See Also

`specify_priors`

volplot *Plotting Quantiles of the Latent Volatilities*

Description

Displays quantiles of the posterior distribution of the volatilities over time as well as predictive distributions of future volatilities.

Usage

```
volplot(
  x,
  forecast = 0,
  dates = NULL,
  show0 = FALSE,
  forecastlty = NULL,
  tcl = -0.4,
  mar = c(1.9, 1.9, 1.9, 0.5),
  mgp = c(2, 0.6, 0),
  simobj = NULL,
  newdata = NULL,
  ...
)
```

Arguments

`x`	`svdraws` object.
`forecast`	nonnegative integer or object of class `svpredict`, as returned by `predict.svdraws`. If an integer greater than 0 is provided, `predict.svdraws` is invoked to obtain the `forecast`-step-ahead prediction. The default value is `0`.
`dates`	vector of length `ncol(x$latent)`, providing optional dates for labeling the x-axis. The default value is `NULL`; in this case, the axis will be labeled with numbers.
`show0`	logical value, indicating whether the initial volatility `exp(h_0/2)` should be displayed. The default value is `FALSE`. Only available for inputs `x` of class `svdraws`.
`forecastlty`	vector of line type values (see `par`) used for plotting quantiles of predictive distributions. The default value `NULL` results in dashed lines.
`tcl`	The length of tick marks as a fraction of the height of a line of text. See `par` for details. The default value is `-0.4`, which results in slightly shorter tick marks than usual.
`mar`	numerical vector of length 4, indicating the plot margins. See `par` for details. The default value is `c(1.9,1.9,1.9,0.5)`, which is slightly smaller than the R-defaults.
`mgp`	numerical vector of length 3, indicating the axis and label positions. See `par` for details. The default value is `c(2,0.6,0)`, which is slightly smaller than the R-defaults.
`simobj`	object of class `svsim` as returned by the SV simulation function `svsim`. If provided, "true" data generating values will be added to the plot(s).
`newdata`	corresponds to parameter `newdata` in `predict.svdraws`. *Only if* `forecast` *is a positive integer and* `predict.svdraws` *needs a* `newdata` *object.* Corresponds to input parameter `designmatrix` in `svsample`. A matrix of regressors with number of rows equal to parameter `forecast`.
`...`	further arguments are passed on to the invoked `ts.plot` function.

Value

Called for its side effects. Returns argument `x` invisibly.

Note

In case you want different quantiles to be plotted, use `updatesummary` on the `svdraws` object first. An example of doing so is given below.

Author(s)

Gregor Kastner <`gregor.kastner@wu.ac.at`>

See Also

updatesummary, predict.svdraws

Other plotting: paradensplot(), paratraceplot.svdraws(), paratraceplot(), plot.svdraws(), plot.svpredict()

Examples

```
## Simulate a short and highly persistent SV process
sim <- svsim(100, mu = -10, phi = 0.99, sigma = 0.2)

## Obtain 5000 draws from the sampler (that's not a lot)
draws <- svsample(sim$y, draws = 5000, burnin = 100,
  priormu = c(-10, 1), priorphi = c(20, 1.5),
  priorsigma = 0.2)

## Plot the latent volatilities and some forecasts
volplot(draws, forecast = 10)

## Re-plot with different quantiles
newquants <- c(0.01, 0.05, 0.25, 0.5, 0.75, 0.95, 0.99)
draws <- updatesummary(draws, quantiles = newquants)

volplot(draws, forecast = 10)
```

Index