Package: bvartools (via r-universe)

October 4, 2026

Title Bayesian Inference of Vector Autoregressive and Error Correction Models

Version 1.0.0

Date 2026-09-20

Description Assists in the set-up of algorithms for Bayesian inference of vector autoregressive (VAR) and error correction (VEC) models. Functions for posterior simulation, forecasting, impulse response analysis and forecast error variance decomposition are largely based on the introductory texts of Chan, Koop, Poirier and Tobias (2019, ISBN: 9781108437493), Koop and Korobilis (2010) <doi:10.1561/0800000013> and Luetkepohl (2006, ISBN: 9783540262398).

License GPL (>= 2)

Depends R (>= 4.0.0), coda

Imports grDevices, graphics, hdf5r, Matrix, methods, parallel, posterior (>= 1.7.0), Rcpp (>= 1.0.12), stats, utils

LinkingTo Rcpp, RcppArmadillo

SystemRequirements GNU make

Encoding UTF-8

URL https://github.com/franzmohr/bvartools, https://franzmohr.github.io/bvartools/

BugReports https://github.com/franzmohr/bvartools/issues

Suggests knitr, loo (>= 2.10.0), pkgload, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Collate 'RcppExports.R' 'add_dummy_variables.R' 'add_dummy_variables.bvarmodel.R' 'add_dummy_variables.bvecmodel.R' 'add_dummy_variables.expandingwindow.R' 'add_dummy_variables.modellist.R' 'add_forecast_errors.R' 'add_forecast_errors.bvarmodel.R'

'add_forecast_errors.bvecmodel.R'
'add_forecast_errors.expandingwindow.R'
'add_forecast_errors.modellist.R' 'add_forecast_input.R'
'add_forecast_input.bvarmodel.R'
'add_forecast_input.bvecmodel.R'
'add_forecast_input.expandingwindow.R'
'add_forecast_input.modellist.R' 'add_initial_values.R'
'add_initial_values.bvarmodel.R'
'add_initial_values.bvecmodel.R'
'add_initial_values.expandingwindow.R'
'add_initial_values.modellist.R' 'add_initial_values_errors.R'
'add_posterior_coefficients.R'
'add_posterior_coefficients.bvarmodel.R'
'add_posterior_coefficients.bvecmodel.R'
'add_posterior_coefficients.expandingwindow.R'
'add_posterior_coefficients.modellist.R'
'add_posterior_forecasts.R'
'add_posterior_forecasts.bvarmodel.R'
'add_posterior_forecasts.expandingwindow.R'
'add_posterior_forecasts.modellist.R' 'add_posterior_loglik.R'
'add_posterior_loglik.bvarmodel.R'
'add_posterior_loglik.bvecmodel.R'
'add_posterior_loglik.expandingwindow.R'
'add_posterior_loglik.modellist.R' 'add_predictive_loglik.R'
'add_predictive_loglik.bvarmodel.R'
'add_predictive_loglik.bvecmodel.R' 'add_prior_options.R'
'add_priors.R' 'add_priors.bvarmodel.R'
'add_priors.bvecmodel.R' 'add_priors.expandingwindow.R'
'add_priors.modellist.R' 'add_priors_checks.R'
'add_priors_sv_helper.R' 'add_seed.R' 'add_sign_restrictions.R'
'add_sign_restrictions.bvarmodel.R'
'add_sign_restrictions.expandingwindow.R'
'add_sign_restrictions.modellist.R'
'add_sign_zero_restrictions.R'
'add_sign_zero_restrictions.bvarmodel.R'
'add_sign_zero_restrictions.expandingwindow.R'
'add_sign_zero_restrictions.modellist.R'
'aggregate_forecasts.R' 'align_model_obs.R'
'align_model_obs.modellist.R'
'arias_rubio_ramirez_waggoner_2018.R' 'bayests_files.R'
'bayests_posterior.R' 'bayests_tree.R' 'bvar.R'
'bvar_fill_helper.R' 'bvartools-package.R' 'bvartools_model.R'
'bvec.R' 'chains.R' 'check_bvarpost_input.R'
'check_bvecpost_input.R' 'choose_best_model.R'
'choose_best_model.selcritlist.R' 'cointspace_prior.R'
'collect_draws.R' 'combine_models.R' 'create_bvarmodel.R'
'create_bvecmodel.R' 'create_external_forecast.R' 'data.R'
'discount_models.R' 'draw_error_bars.R'

'dummy_variables_helpers.R' 'expected_model_size.R'
'expected_model_size.bvarmodel.R'
'expected_model_size.bvecmodel.R'
'expected_model_size.expandingwindow.R'
'expected_model_size.modellist.R' 'external_forecast_methods.R'
'fevd.R' 'fevd.bvarmodel.R' 'folder_methods.R'
'forecast_plot_series.R' 'generate_artificial_helper.R'
'generate_artificial_var.R' 'generate_artificial_vec.R'
'get_forecast_errors.R' 'get_forecast_errors.bvarmodel.R'
'get_forecast_errors.bvecmodel.R' 'get_model_specifications.R'
'get_model_specifications.bvarmodel.R'
'get_model_specifications.bvecmodel.R'
'get_model_specifications.expandingwindow.R'
'get_model_specifications.selcrit.R' 'get_regressor_names.R'
'hdf5_groups.R' 'hdf5_write.R' 'historical_decomposition.R'
'iid_variables.R' 'inclusion_prior.R'
'inclusion_prior.bvarmodel.R' 'inclusion_prior.bvecmodel.R'
'irf.R' 'irf.bvarfile.R' 'irf.bvarmodel.R' 'map_models.R'
'minnesota_prior.R' 'minnesota_prior.bvarmodel.R'
'minnesota_prior.bvecmodel.R' 'minnesota_prior.modellist.R'
'multipliers.R' 'multipliers.bvarmodel.R'
'multipliers.bvecmodel.R' 'open_model.R' 'open_models.R'
'panel_constraints.R' 'parallel_models.R'
'persistence_profiles.R' 'persistence_profiles.bvarmodel.R'
'persistence_profiles.bvecmodel.R'
'persistence_profiles.modellist.R' 'plot.bvarfevd.R'
'plot.bvarirf.R' 'plot.bvarmodel.R' 'plot.bvarprd.R'
'plot.bvarprdlist.R' 'plot.bvarspillover.R' 'plot.bvecmodel.R'
'plot.expwindbvarprdlist.R' 'plot.modellist.R'
'plot.selcritlist.R' 'plot_blocks.R'
'plot_forecast_errors_by_period.R' 'plot_predictive_loglik.R'
'plot_selcrit_forecast_errors.R' 'plugin_deviance.R'
'pool_forecasts.R' 'pool_forecasts_methods.R'
'predict.bvarmodel.R' 'predict.expandingwindow.R'
'predict.modellist.R' 'prepare_forecast_input.R'
'prepare_forecast_input.bvarmodel.R'
'prepare_forecast_input.bvecmodel.R' 'print.bvarmodel.R'
'print.bvarspillover.R' 'print.bvecmodel.R' 'print.modellist.R'
'selection_criteria.R' 'print.selcrit.R' 'print.selcritlist.R'
'summary.bvarmodel.R' 'print.summary.bvarmodel.R'
'summary.bvecmodel.R' 'print.summary.bvecmodel.R' 'psis_loo.R'
'read_expanding_window_model_from_folder.R'
'read_model_from_hdf5.R' 'read_models_from_folder.R'
'rescale_error_correction.R' 'restrictions.R'
'scale_error_correction.R' 'selection_criteria.bvarmodel.R'
'selection_criteria.bvecmodel.R' 'selection_criteria.default.R'
'selection_criteria.expandingwindow.R'
'selection_criteria.externalforecast.R'

'selection_criteria.modellist.R' 'spillover.R'
'spillover.bvarmodel.R' 'spillover.expandingwindow.R'
'spillover.modellist.R' 'split_quantile_grid.R' 'ssvs_prior.R'
'ssvs_prior.bvarmodel.R' 'ssvs_prior.bvecmodel.R'
'summary.expandingwindow.R' 'summary.modellist.R'
'thin.bvarmodel.R' 'thin.bvecmodel.R' 'thin.expandingwindow.R'
'thin.modellist.R' 'time_variation_test.R'
'time_variation_test.modellist.R' 'transform_variables.R'
'use_expanding_window.R' 'use_expanding_window.bvarmodel.R'
'use_expanding_window.bvecmodel.R'
'use_expanding_window.modellist.R' 'vec_to_var.R'
'vec_to_var.bvecmodel.R' 'vec_to_var.modellist.R' 'waic.R'
'window.bvarmodel.R' 'window.bvecmodel.R' 'window.modellist.R'
'write_to_hdf5.R' 'write_to_hdf5.bvarmodel.R'
'write_to_hdf5.bvecmodel.R' 'write_to_hdf5.expandingwindow.R'
'write_to_hdf5.modellist.R' 'zzz.R'

LazyData true

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/pak/sysreqs make libhdf5-dev

Repository https://franzmohr.r-universe.dev

Date/Publication 2026-10-04 10:55:15 UTC

RemoteUrl https://github.com/franzmohr/bvartools

RemoteRef HEAD

RemoteSha 5aa5ba7e7ec975c6a451b28c915b3c485356bf72

Contents

add_dummy_variables 9
add_dummy_variables.bvarmodel 9
add_forecast_errors 11
add_forecast_errors.bvarmodel 12
add_forecast_errors.bvecmodel 13
add_forecast_errors.expandingwindow 14
add_forecast_errors.modellist 16
add_forecast_input 17
add_forecast_input.bvarmodel 17
add_forecast_input.bvecmodel 19
add_forecast_input.expandingwindow 20
add_forecast_input.modellist 21
add_initial_values 22
add_initial_values.bvarmodel 22
add_initial_values.bvecmodel 24
add_initial_values.expandingwindow 26
add_initial_values.modellist 27

add_posterior_coefficients . . . 28
add_posterior_coefficients.bvarmodel . . . 29
add_posterior_coefficients.bvecmodel . . . 31
add_posterior_coefficients.expandingwindow . . . 33
add_posterior_coefficients.modellist . . . 35
add_posterior_forecasts . . . 37
add_posterior_forecasts.bvarmodel . . . 37
add_posterior_forecasts.bvecmodel . . . 40
add_posterior_forecasts.expandingwindow . . . 41
add_posterior_forecasts.modellist . . . 43
add_posterior_loglik . . . 45
add_posterior_loglik.bvarmodel . . . 45
add_posterior_loglik.bvecmodel . . . 46
add_posterior_loglik.expandingwindow . . . 48
add_posterior_loglik.modellist . . . 49
add_predictive_loglik . . . 51
add_predictive_loglik.bvarmodel . . . 54
add_predictive_loglik.bvecmodel . . . 56
add_prior_options . . . 57
add_priors . . . 60
add_priors.bvarmodel . . . 60
add_priors.bvecmodel . . . 65
add_priors.expandingwindow . . . 73
add_priors.modellist . . . 74
add_seed . . . 75
add_sign_restrictions . . . 76
add_sign_restrictions.bvarmodel . . . 77
add_sign_zero_restrictions . . . 80
add_sign_zero_restrictions.bvarmodel . . . 80
aggregate_forecasts . . . 85
aggregation_weights . . . 88
align_model_obs . . . 89
align_model_obs.modellist . . . 90
analysis_of_stored_models . . . 91
arias_rubio_ramirez_waggoner_2018 . . . 92
at_macrodata . . . 94
bayests_files . . . 95
bayests_posterior . . . 96
bvar . . . 98
bvartools_model . . . 102
bvec . . . 105
chain_diagnostics . . . 110
choose_best_model . . . 113
choose_best_model.selcritlist . . . 113
coint_kls2010_reparameterise_two . . . 115
coint_prepare_sur_data . . . 116
cointspace_prior . . . 117
combine_models . . . 121

covar_prepare_data . . . 122
covar_vector_to_matrix . . . 124
create_bvarmodel . . . 125
create_bvecmodel . . . 131
create_external_forecast . . . 135
discount_models . . . 139
e1 . . . 140
e6 . . . 141
expected_model_size . . . 141
expected_model_size.bvarmodel . . . 142
fevd . . . 144
fevd.bvarmodel . . . 145
fevd.bvecmodel . . . 148
folder_steps . . . 149
from_bayests_tree . . . 151
generate_artificial_var . . . 152
generate_artificial_vec . . . 156
generate_lower_block_diagonal . . . 160
get_forecast_errors . . . 161
get_model_specifications . . . 162
historical_decomposition . . . 163
historical_decomposition.bvarmodel . . . 163
inclusion_prior . . . 165
inclusion_prior.bvarmodel . . . 166
inclusion_prior.bvecmodel . . . 167
irf . . . 169
irf.bvarmodel . . . 170
irf.bvecmodel . . . 173
kalman_durbin_koopman_2002 . . . 173
list_models_in_hdf5 . . . 175
loglik_normal . . . 176
map_draws . . . 178
map_models . . . 179
minnesota_prior . . . 181
minnesota_prior.bvarmodel . . . 182
minnesota_prior.bvecmodel . . . 184
minnesota_prior.modellist . . . 186
multipliers . . . 186
multipliers.bvarmodel . . . 187
multipliers.bvecmodel . . . 190
open_model . . . 191
open_models . . . 192
persistence_profiles . . . 194
persistence_profiles.bvarmodel . . . 195
plot.bvarfevd . . . 197
plot.bvarhd . . . 198
plot.bvarirf . . . 199
plot.bvarmodel . . . 200

plot.bvarprd . . . 201
plot.bvarprdlist . . . 202
plot.bvarspillover . . . 203
plot.bvarspilloverts . . . 204
plot.bvecmodel . . . 204
plot.expandwindbvarprdlist . . . 205
plot.modellist . . . 206
plot.selcritlist . . . 207
plot_forecast_errors_by_period . . . 208
plot_predictive_loglik . . . 210
pool_forecasts . . . 212
post_bvs . . . 216
post_coint_kls . . . 218
post_coint_kls_sur . . . 221
post_gamma_measurement_variance . . . 223
post_gamma_state_variance . . . 224
post_normal . . . 226
post_normal_sur . . . 227
predict.bvarmodel . . . 229
predict.bvecmodel . . . 230
predict.expandingwindow . . . 231
predict.modellist . . . 232
prepare_forecast_input . . . 233
prepare_forecast_input.bvarmodel . . . 234
prepare_forecast_input.bvecmodel . . . 235
print.bvarmodel . . . 236
print.bvarspillover . . . 237
print.bvecmodel . . . 237
print.modellist . . . 238
read_expanding_window_model_from_folder . . . 239
read_model_from_hdf5 . . . 239
read_models_from_folder . . . 240
rescale_error_correction . . . 242
rescale_error_correction.bvecmodel . . . 242
scale_error_correction . . . 243
scale_error_correction.bvecmodel . . . 244
selection_criteria . . . 245
selection_criteria.bvarmodel . . . 248
selection_criteria.bvecmodel . . . 249
selection_criteria.default . . . 251
selection_criteria.expandingwindow . . . 252
selection_criteria.externalforecast . . . 254
selection_criteria.modellist . . . 255
spillover . . . 256
spillover.bvarmodel . . . 257
spillover.bvecmodel . . . 260
spillover.expandingwindow . . . 261
spillover.modellist . . . 262

split_quantile_grid . . . 263
ssvs . . . 264
ssvs_prior . . . 266
ssvs_prior.bvarmodel . . . 267
ssvs_prior.bvecmodel . . . 268
stochvol_ksc_1998 . . . 269
stochvol_ocsn_2007 . . . 271
summary.bvarmodel . . . 272
summary.bvecmodel . . . 273
summary.expandingwindow . . . 274
summary.modellist . . . 275
sur_const_to_tvp . . . 275
thin.bvarmodel . . . 276
thin.bvecmodel . . . 277
thin.expandingwindow . . . 278
thin.modellist . . . 279
time_variation_test . . . 279
time_variation_test.bvarmodel . . . 280
transform_variables . . . 282
uk_macrodata . . . 284
us_macrodata . . . 284
use_expanding_window . . . 285
use_expanding_window.bvarmodel . . . 286
use_expanding_window.bvecmodel . . . 287
use_expanding_window.modellist . . . 288
vec_to_var . . . 289
vec_to_var.bvecmodel . . . 290
vec_to_var.expandingwindow . . . 292
vec_to_var.modellist . . . 293
window.bvarmodel . . . 293
window.bvecmodel . . . 294
window.modellist . . . 295
write_bayests_tree . . . 295
write_to_hdf5 . . . 297
write_to_hdf5.bvarmodel . . . 298
write_to_hdf5.bvecmodel . . . 299
write_to_hdf5.expandingwindow . . . 300
write_to_hdf5.modellist . . . 301

Index **303**

add_dummy_variables *Dummy Variables*

Description

A generic function used to add dummy variables to the deterministic terms of a model.

Usage

```
add_dummy_variables(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of object, as described on the pages of the methods.

See Also

Methods: add_dummy_variables.bvarmodel, add_dummy_variables.bvecmodel, add_dummy_variables.expandingwi
add_dummy_variables.modellist.

add_dummy_variables.bvarmodel *Dummy Variables*

Description

Adds dummy variables to the deterministic terms of a model: impulse dummies for single unusual periods, step dummies for level shifts, or any other series that is known in advance.

Usage

```
## S3 method for class 'bvarmodel'
add_dummy_variables(object, impulse = NULL, step = NULL, data = NULL, ...)

## S3 method for class 'bvecmodel'
add_dummy_variables(object, impulse = NULL, step = NULL, data = NULL, ...)

## S3 method for class 'expandingwindow'
add_dummy_variables(object, impulse = NULL, step = NULL, data = NULL, ...)

## S3 method for class 'modellist'
add_dummy_variables(object, ...)
```

Arguments

`object`	an object of class 'bvarmodel' or 'bvecmodel', usually the output of `create_bvarmodel` or `create_bvecmodel`, or a list of such models of class 'modellist' or 'expandingwindow'.
`impulse`	a period, given the way `ts` takes `start`, e.g. `c(2020, 2)`, or a list of such periods. Each gets a dummy variable that is one in that period and zero otherwise.
`step`	a period or a list of periods as in `impulse`. Each gets a dummy variable that is zero before that period and one from it on.
`data`	an optional time-series object of further dummy variables, or of any other series known in advance, with named columns at the frequency of the model. It has to cover the estimation sample. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

A dummy variable takes up what a model otherwise has no explanation for: an impulse dummy the one period of a strike, a tax change or a pandemic, a step dummy a lasting shift in the level of the series, such as a change in how they are measured. Without one, a single extreme observation pulls the coefficients of every equation towards itself and inflates the estimated error variance for the whole sample.

The dummies are deterministic terms like the constant. They are appended to the end of the deterministic terms of the regressors, named after their type and period – `impulse.2020Q2`, `step.2008M09` – or after the columns of `data`, and their coefficients take the prior of the deterministic terms, `v_i_det` in `add_priors`. **The coefficient of an impulse dummy is estimated from a single observation**, so its prior matters more than that of the constant: a tight one leaves most of the observation to the other coefficients, which is what the dummy was meant to prevent.

The function has to be called before `add_priors`, since it changes the number of coefficients the priors are given for. A dummy variable that is zero in every period of the estimation sample, or that the other regressors already span – a step dummy that is one in every period is the constant again – is refused.

A forecast continues the dummies by their definition: an impulse dummy is zero after its period and a step dummy stays one. The series in `data` are continued with the values they were given, so for a forecast they have to reach the last forecast period, or all deterministic terms of the forecast have to be given in argument `deterministic` of `add_forecast_input`.

Of a list of models of class 'expandingwindow', each window gets the dummies whose periods it reaches and leaves out the rest: a forecast made at the end of a window that ends before a period could not have known about what happened in it. `use_expanding_window` applied to a model with dummy variables does the same.

In a VEC model the dummies enter the equations of the differences, not the cointegration term. **An impulse dummy there shifts the level of the series for good**, since a one-off change in a difference is a lasting change in the level. A one-off blip in the level is a dummy that is one in its period and minus one in the next, which can be given in `data`. `vec_to_var` carries the dummies into the VAR representation.

Value

The object in `object` with the dummy variables added to `data$train$x` and its SUR form `data$train$z`, to `data$original$deterministic` and to the names in `model$deterministic`, `model$n` increased by their number, and `model$dummy_variables`, a data frame with one row per dummy variable giving its `name`, its `type` (`"impulse"`, `"step"` or `"data"`) and the `time` of its period, which is what a forecast continues it by.

See Also

Other model set-up: `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvarmodel()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bv`
`use_expanding_window.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# An impulse dummy for the first quarter of 1975 and a step dummy from 1979
model <- add_dummy_variables(model, impulse = c(1975, 1), step = c(1979, 1))
model[["model"]][["deterministic"]]

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
```

---

add_forecast_errors *Add Forecast Errors*

Description

Generic function used to calculate forecast errors and add them to a model object.

Usage

```
add_forecast_errors(object, test_sample = NULL, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`test_sample`	a time-series object used as test data. If `NULL` (default), the values in `data$test$y` of the object are used, which is what a model carries after it has been scored once and what a model read from a file was written with.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_forecast_errors.bvarmodel`, `add_forecast_errors.expandingwindow`, `add_forecast_errors.model`

`add_forecast_errors.bvarmodel` *Add Forecast Errors*

Description

Calculates forecast errors and adds them to an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
add_forecast_errors(object, test_sample = NULL, ...)
```

Arguments

`object`	an object of class 'bvarmodel'.
`test_sample`	a time-series object used as test data. If `NULL` (default), the values in `data$test$y` of the object are used.
`...`	arguments passed forward to method.

Value

The object in `object` with `posterior$forecast$errors` added, a `mcmc` object with one row per draw and Kh columns in the order of `posterior$forecast$forecasts`, which it is stored beside. `selection_criteria` summarises them.

See Also

bvartools_model describes the object this returns, element by element.

Other model comparison: add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.b
add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(),
analysis_of_stored_models, choose_best_model.selcritlist(), create_external_forecast(),
folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(),
pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(),
selection_criteria.default(), selection_criteria.modellist()

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = orig)
```

---

add_forecast_errors.bvecmodel

Add Forecast Errors

Description

Calculates the forecast errors of a VEC model against a test sample, in levels.

Usage

```
## S3 method for class 'bvecmodel'
add_forecast_errors(object, test_sample = NULL, ...)
```

Arguments

object	an object of class 'bvecmodel', usually, the result of a call to add_posterior_forecasts.
test_sample	a time-series object of the endogenous variables, in levels, that covers the forecast periods. If NULL (default), the values in data$test$y of the object are used.
...	further arguments passed to add_forecast_errors.bvarmodel.

Details

The forecasts of a VEC model are of the levels, so the errors are taken against the levels of test_sample, exactly as for the VAR representation vec_to_var would give.

Value

The object in object with posterior$forecast$errors added, as described in add_forecast_errors.bvarmodel.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_predictive_loglik(), add_predictive_loglik.b
add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(),
analysis_of_stored_models, choose_best_model.selcritlist(), create_external_forecast(),
folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(),
pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(),
selection_criteria.default(), selection_criteria.modellist()

---

add_forecast_errors.expandingwindow
Add Forecast Errors

Description

Calculates and adds forecast errors for a list of Bayesian models.

Usage

```
## S3 method for class 'expandingwindow'
add_forecast_errors(object, test_sample = NULL, ...)
```

Arguments

object	an object of class 'expandingwindow'.
test_sample	a time-series object used as test data. If NULL (default), the values in data$test$y of the object are used.
...	arguments passed forward to method.

Value

The object in object with forecast errors added to each of its models, as described in add_forecast_errors.bvarmodel.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = orig)
```

`add_forecast_errors.modellist` *Add Forecast Errors*

Description

Calculates and adds forecast errors for a list of Bayesian models.

Usage

```
## S3 method for class 'modellist'
add_forecast_errors(object, test_sample = NULL, ...)
```

Arguments

`object`	an object of class 'modellist'.
`test_sample`	a time-series object used as test data. If `NULL` (default), the values in `data$test$y` of the object are used.
`...`	arguments passed forward to method.

Value

The object in `object` with forecast errors added to each of its models, as described in `add_forecast_errors.bvarmodel`.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 0:2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
```

```
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = orig)
```

`add_forecast_input` *Add Forecast Input Data*

Description

Generic function used to generate and add data matrices for forecast simulation to a model object.

Usage

```
add_forecast_input(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_forecast_input.bvarmodel`, `add_forecast_input.bvecmodel`, `add_forecast_input.expandingwindo`
`add_forecast_input.externalforecast`, `add_forecast_input.modellist`.

`add_forecast_input.bvarmodel`
Add Forecast Input Data

Description

Generates and adds data matrices for forecast simulation to the elements of an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
add_forecast_input(
  object,
  n_ahead = 10,
  deterministic = NULL,
  exogen = NULL,
  ...
)
```

Arguments

object	an object of class 'bvarmodel'.
n_ahead	number of steps ahead at which to predict.
deterministic	a time-series object with deterministic data. If not specified, the function will try to identify the deterministic terms automatically. If this is not successful, an error message we be returned.
exogen	a time-series object with the unmodelled, non-deterministic variables of the model. Required if the model has such variables. It has to cover the s periods before the first forecast period, which are the last s periods of the estimation sample, as well as the n_ahead forecast periods, because the regressors of a forecast period include the lags of these variables. See prepare_forecast_input.
...	arguments passed forward to method.

Value

The object in object with model$h set to n_ahead and data$forecast$x added, the regressors of the forecast periods in the layout of data$train$x, one row per period.

See Also

Other posterior simulation: add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecasts.bvarmodel(), add_posterior_forecasts.b
add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel(), add_seed(), bayests_files(), bayests_posterior(), bvar(), bvec(), chain_diagnostics(), predict.bvecmodel()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
```

```
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)
```

`add_forecast_input.bvecmodel` *Add Forecast Input Data*

Description

Prepares the input data of the forecast periods of a VEC model.

Usage

```
## S3 method for class 'bvecmodel'
add_forecast_input(
  object,
  n_ahead = 10,
  deterministic = NULL,
  exogen = NULL,
  ...
)
```

Arguments

`object`	an object of class 'bvecmodel', usually, the result of a call to `add_posterior_coefficients`.
`n_ahead`	an integer of the forecast horizon.
`deterministic, exogen`	the values of the deterministic terms and of the unmodelled variables in the forecast periods, as for `add_forecast_input.bvarmodel`.
`...`	further arguments passed to `prepare_forecast_input`.

Details

A VEC model is forecast in levels: it is the same model as its VAR representation, and the forecast of the differences follows from that of the levels. The regressors of the forecast periods are therefore those of the VAR in levels, and are assembled from the data and specification that `vec_to_var` builds. The posterior draws are not converted.

Value

The object in `object` with the forecast horizon in `model$h` and the regressors of the forecast periods, in levels, in `data$forecast$x`.

See Also

Other posterior simulation: add_forecast_input.bvarmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecasts.bvarmodel(), add_posterior_forecasts.b add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel(), add_seed(), bayests_files(), bayests_posterior(), bvar(), bvec(), chain_diagnostics(), predict.bvecmodel()

---

add_forecast_input.expandingwindow
Add Forecast Input Data

Description

Generates and adds data matrices for forecast simulation to the elements of an object of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
add_forecast_input(object, ...)
```

Arguments

object	an object of class 'expandingwindow' containing objects that can be passed on to their respective add_forecast_input method.
...	arguments passed forward to method.

Value

The object in object with forecast input added to each of its models, as described in add_forecast_input.bvarmodel.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
```

```
                    sigma = list(df = "k", scale = 1))

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)
```

`add_forecast_input.modellist` *Add Forecast Input Data*

Description

Generates and adds data matrices for forecast simulation to the elements of an object of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
add_forecast_input(object, ...)
```

Arguments

`object`	an object of class 'modellist' containing objects that can be passed on to their respective `add_forecast_input` method.
`...`	arguments passed forward to method.

Value

The object in `object` with forecast input added to each of its models, as described in `add_forecast_input.bvarmodel`.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 0:2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
```

```
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)
```

`add_initial_values` *Add Initial Values of an MCMC Chain*

Description

A generic function used to generate initial values of an MCMC chain. The function invokes particular methods which depend on the class of the first argument.

Usage

```
add_initial_values(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_initial_values.bvarmodel`, `add_initial_values.bvecmodel`, `add_initial_values.expandingwindo`
`add_initial_values.externalforecast`, `add_initial_values.modellist`.

`add_initial_values.bvarmodel`
Add Initial Values of an MCMC Chain

Description

Adds initial values to a VAR model.

Usage

```
## S3 method for class 'bvarmodel'
add_initial_values(object, method = "ols", ...)
```

Arguments

`object`	list of class 'bvarmodel', usually, the result of a call to `create_bvarmodel` in combination with `add_priors`.
`method`	character specifying the method of how initial values are generated. Default is `"ols"`. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

For argument `method` the following specifications are possible:

`"ols"` Inital values are equal to estimates from an unrestricted LS regression.

`"prior"` Initial values are drawn from the prior. Not possible for uninformative priors.

In case `method = "ols"`, the initial draw of a is the result of the unrestricted LS regression $(Z'Z)^{-1}Z'y$ of the model $y_t = Z_t a + u_t$, where Z is the matrix of regressors in SUR form and y is the vector of endogenous variables.

The initial draw of Σ^u is the sum of squared residuals divided by the number of observations, i.e. $\frac{uu'}{T}$, with u as the residuals of the LS regression.

In case of a model with time varying parameters (TVP), the initial states are obtained using the approach specified in argument `method`. The precisions of the state equations start at the means of their gamma priors in case `method = "ols"` and are drawn from those priors in case `method = "prior"`. So `method = "ols"` uses no random numbers, and a seed set after `add_initial_values` fixes the posterior draws.

Value

The object in `object` with the element `initial` added, a list of starting values with

`a` the coefficients, an $M \times 1$ matrix, or $TM \times 1$ for TVP models, whose initial states and state precisions are held in `a_init` and `a_sigma_inv`.

`u_sigma_inv` for `error = "wishart"`, the $K \times K$ inverse error covariance matrix.

`u_omega_inv` for gamma priors, the $K \times K$ diagonal matrix of error precisions.

`h`, `h_init` for stochastic volatility, the $T \times K$ log-volatilities and their initial states.

`psi` for `"gamma+covar"` and `"sv+covar"`, the error covariance coefficients.

`a_lambda` with variable selection, the inclusion indicators.

`u_scale`, `w` for `error = "ald"`, the $K \times 1$ scales and the $T \times K$ latent weights of the asymmetric Laplace distribution.

Elements that do not apply to a model are absent.

Seed

The function also stores the seed of the posterior simulation as element `seed` of `object$model`, unless the model has one already. It is drawn from R's random number generator, so `set.seed()` before this call makes it reproducible. `add_seed` replaces it.

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvecmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), aggregation_weights(), combine_models(), create_bvarmodel(), create_bvecmodel(), transform_variables(), use_expanding_window.bv use_expanding_window.bvecmodel()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 50, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)
```

---

add_initial_values.bvecmodel

Add Initial Values of an MCMC Chain

Description

Adds initial values to a VEC model, which was produced by function create_bvecmodel in combination with add_priors.

Usage

```
## S3 method for class 'bvecmodel'
add_initial_values(object, method = "maxlik", ...)
```

Arguments

object	list of class 'bvecmodel'.
method	character specifying the method of how initial values are generated. Defaults is "maxlik". Different approaches are used for TVP and SV models. See 'Details'.
...	further arguments passed to or from other methods.

Details

For argument method the following specifications are possible:

"maxlik" Inital values are equal to estimates from maximum likelihood regressions.

"prior" Initial values are drawn from the prior. Not possible for uninformative priors.

In case method = "maxlik", the initial draw of a is the result of a maximum likelihood estimation of the reduced rank model. When used with a Wishart prior, the initial draw of Σ^u is the sum of squared residuals of ML regression divided by the number of observations, i.e. $\frac{uu'}{T}$. In all other cases, the diagonal elements of Σ^u are set to the variances of the variables in u.

In case method = "prior", all initial draws in the model are random draws from the respective prior distributions.

In case of a model with time varying parameters (TVP), the initial states are obtained using the approach specified in argument method. The precisions of the state equations start at the means of their gamma priors in case method = "maxlik" and are drawn from those priors in case method = "prior". The autocorrelation coefficient rho of the state equation of β is taken from the prior specification, whether it is held there or drawn from the prior on it that coint$rho_min and coint$rho_max set up.

Value

The object in object with the element initial added, a list with the starting values beta of the cointegration coefficients, a $K_\beta r \times 1$ matrix, or $TK_\beta r \times 1$ with initial state beta_init for time varying cointegration, and the starting values of the remaining coefficients and the error term with the elements described in add_initial_values.bvarmodel.

Seed

The function also stores the seed of the posterior simulation as element seed of object$model, unless the model has one already. It is drawn from R's random number generator, so set.seed() before this call makes it reproducible. add_seed replaces it.

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvarmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), aggregation_weights(), combine_models(), create_bvarmodel(), create_bvecmodel(), transform_variables(), use_expanding_window.bv use_expanding_window.bvecmodel()

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 4, r = 1,
                          const = "unrestricted",
                          seasonal = "unrestricted",
```

```
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)
```

`add_initial_values.expandingwindow` *Add Initial Values of an MCMC Chain*

Description

Adds initial values to a list of models by passing each element to the respective method.

Usage

```
## S3 method for class 'expandingwindow'
add_initial_values(object, ...)
```

Arguments

`object`	a list of class 'expandingwindow', usually, the output of a call to `use_expanding_window`.
`...`	further arguments passed to or from other methods.

Value

The object in `object` with initial values added to each of its models, as described in `add_initial_values.bvarmodel`.

Examples

```
data("us_macrodata")

# Starting period of the forecasting exercise
start_period <- 2007

# AR(1) models as benchmark
model <- create_bvarmodel(data = us_macrodata, p = 1,
                          deterministic = "none",
                          error = "gamma",
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in draws should be much higher.
```

```
# Obtain objects for expanding window estimation
model <- use_expanding_window(model, start = start_period)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(shape = 3, rate = 1))

# Add initial values
model <- add_initial_values(model)
```

`add_initial_values.modellist` *Add Initial Values of an MCMC Chain*

Description

Adds initial values to a list of models by passing each element to the respective method.

Usage

```
## S3 method for class 'modellist'
add_initial_values(object, ...)
```

Arguments

`object`	a list of class 'modellist' `add_priors`.
`...`	further arguments passed to or from other methods.

Value

The object in `object` with initial values added to each of its models, as described in `add_initial_values.bvarmodel`.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 0:2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
```

```
# Add initial values
model <- add_initial_values(model)
```

`add_posterior_coefficients` *Posterior Simulation of Model Coefficients*

Description

Generic function used for posterior simulation of model coefficients.

Usage

```
add_posterior_coefficients(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Details

Before the method is called, the size the object will have once its draws are complete is compared with `options(bvartools.size_warning)`, and a warning is given if it is larger. See `expected_model_size`.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_posterior_coefficients.bvarmodel`, `add_posterior_coefficients.bvecmodel`, `add_posterior_coefficients.expandingwindow`, `add_posterior_coefficients.externalforecast`, `add_posterior_coefficients.modellist`.

add_posterior_coefficients.bvarmodel
Posterior Simulation of Model Coefficients

Description

Forwards model input to posterior simulation functions for vector autoregressive models.

Usage

```
## S3 method for class 'bvarmodel'
add_posterior_coefficients(
  object,
  posterior_function = NULL,
  chains = NULL,
  ...
)
```

Arguments

object	an object of class 'bvarmodel', usually, a result of a call to create_bvarmodel in combination with add_priors and add_initial_values.
posterior_function	the function to be applied to the model in argument object. If NULL (default), internal functions are used.
chains	the number of chains to simulate. If NULL (default), the value in object$model$chains is used, and one chain when there is none. Each chain is the same simulation with a seed of its own – the first with the seed of the model, so that one chain draws what it always has – and the chains are pooled, one after the other, in the draws of posterior, so that every later step uses all of them. Their number, when above one, is stored in object$model$chains, and chain_diagnostics compares them. A posterior_function is called once per chain. Not available for discounted models, whose posterior is not a chain.
...	further arguments passed to or from other methods.

Details

Unless posterior_function is specified, the function forwards the model input to the package's own posterior functions.

The internal samplers draw with the seed in object$model$seed, which add_initial_values sets and add_seed replaces. R's random number generator is set to that seed, with R's default kinds, for the simulation and put back as it was afterwards. A call of set.seed() between add_initial_values() and this function therefore does not change the draws. A model without a seed draws from R's generator as it stands. A posterior_function is called as it is and decides itself what to do with the seed; see bayests_posterior.

A sampler that cannot run raises its error rather than returning something. The message names what about the input it could not work with. Applied to a list of models – a 'modellist', an 'expanding-window' or, in **bgvars**, a 'gvarmodel' – that ends the run on the first specification that fails, rather than leaving that one without a posterior and carrying it into whatever reads the results.

Value

The object in `object` with the element `posterior` added. Each of its elements is a list whose element `coeffs` holds the draws after burn-in as a `mcmc` object with one row per draw and one column per parameter:

`a` the coefficients, M columns or TM for TVP models, in the order of the columns of `data$train$z` with the contemporaneous coefficients of structural models last. With variable selection element `lambda` holds the inclusion indicators, and for TVP models element `sigma` the state variances.

`u_sigma_inv` the inverse error covariance matrix, K^2 columns, or TK^2 if the error variances vary over time.

`u_omega_inv` for all errors but `"wishart"`, the error precisions, K columns or TK.

`psi` for `"gamma+covar"` and `"sv+covar"`, the error covariance coefficients.

`u_scale` for `error = "ald"`, the K scales of the asymmetric Laplace distribution.

`y` for a model estimated from a panel not observed whole, the completed panel, TK columns ordered by period, and `constraints_inv` the error precisions of its soft constraints. See `create_bvarmodel`.

`mu` under a steady-state prior, the K unconditional means. Under an adaptive prior, `a` holds the scales of its groups in element `shrinkage` and, for the horseshoe and the normal-gamma prior, the scales of its coefficients in `local`; where the normal-gamma prior draws θ, its draws are in `theta`. See `add_prior_options`.

Elements that do not apply to a model are absent or `NULL`. Note that `bvar` expects draws in the transposed orientation.

See Also

`bvartools_model` describes the object this returns, element by element.

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecasts.bvarmodel()`, `add_posterior_forecasts.b` `add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel()`, `add_seed()`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
```

```
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)
```

`add_posterior_coefficients.bvecmodel` *Posterior Simulation of Model Coefficients*

Description

Forwards model input to posterior simulation functions for vector error correction models.

Usage

```
## S3 method for class 'bvecmodel'
add_posterior_coefficients(
  object,
  posterior_function = NULL,
  chains = NULL,
  ...
)
```

Arguments

`object`	an object of class 'bvecmodel', usually, a result of a call to `create_bvecmodel` in combination with `add_priors` and `add_initial_values`.
`posterior_function`	the function to be applied to the model in argument `object`. If `NULL` (default), internal functions are used.
`chains`	the number of chains to simulate. If `NULL` (default), the value in `object$model$chains` is used, and one chain when there is none. Each chain is the same simulation with a seed of its own – the first with the seed of the model, so that one chain draws what it always has – and the chains are pooled, one after the other, in the draws of `posterior`, so that every later step uses all of them. Their number, when above one, is stored in `object$model$chains`, and `chain_diagnostics` compares them. A `posterior_function` is called once per chain. Not available for discounted models, whose posterior is not a chain.
`...`	further arguments passed to or from other methods.

Details

The function implements commonly used posterior simulation algorithms for Bayesian VAR models with both constant and time varying parameters (TVP) as well as stochastic volatility. It can produce posterior draws for standard BVAR models with independent normal-Wishart priors, which can be augmented by stochastic search variable selection (SSVS) as proposed by Geroge et al. (2008) or Bayesian variable selection (BVS) as proposed in Korobilis (2013). Both SSVS or BVS can also be applied to the covariances of the error term.

The implementation follows the descriptions in Chan et al. (2019), George et al. (2008) and Korobilis (2013). For all approaches the SUR form of a VAR model is used to obtain posterior draws. The algorithm is implemented in C++ to reduce calculation time.

The function also supports structural BVEC models, where the structural coefficients are estimated from contemporary endogenous variables, which corresponds to the so-called (A-model). Currently, only specifications are supported, where the structural matrix contains ones on its diagonal and all lower triangular elements are freely estimated. Since posterior draws are obtained based on the SUR form of the VEC model, the structural coefficients are drawn jointly with the other coefficients.

The internal samplers draw with the seed in `object$model$seed`, which `add_initial_values` sets and `add_seed` replaces. R's random number generator is set to that seed, with R's default kinds, for the simulation and put back as it was afterwards. A call of `set.seed()` between `add_initial_values()` and this function therefore does not change the draws. A model without a seed draws from R's generator as it stands. A `posterior_function` is called as it is and decides itself what to do with the seed; see `bayests_posterior`.

A sampler that cannot run raises its error rather than returning something. The message names what about the input it could not work with. Applied to a list of models – a 'modellist', an 'expanding-window' or, in **bgvars**, a 'gvecmodel' – that ends the run on the first specification that fails, rather than leaving that one without a posterior and carrying it into whatever reads the results.

Value

The object in `object` with the element `posterior` added, whose elements hold the draws after burn-in as `mcmc` objects with one row per draw and one column per parameter, in element `coeffs`: `beta`, the cointegration coefficients, $K_\beta r$ columns or $TK_\beta r$ for time varying cointegration, `a`, the loadings and the remaining coefficients, and the draws of the error term as described in `add_posterior_coefficients.bvarmodel`.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224–242. doi:10.1080/07474930903382208

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

See Also

bvartools_model describes the object this returns, element by element.

Other posterior simulation: add_forecast_input.bvarmodel(), add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_forecasts.bvarmodel(), add_posterior_forecasts.b add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel(), add_seed(), bayests_files(), bayests_posterior(), bvar(), bvec(), chain_diagnostics(), predict.bvecmodel()

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 1, r = 1, const = "restricted",
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)
```

---

add_posterior_coefficients.expandingwindow

Posterior Simulation of Model Coefficients

Description

Forwards model input to posterior simulation functions.

Usage

```
## S3 method for class 'expandingwindow'
add_posterior_coefficients(object, ..., cores = 1)
```

Arguments

`object`	an object of class 'expandingwindow'.
`...`	further arguments passed to or from other methods.
`cores`	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in `object` with posterior draws added to each of its models, as described in `add_posterior_coefficients.bvarr`

Parallel simulation

With `cores` above 1 the models are simulated on a cluster of that many worker processes, started with `makeCluster` and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that `use_expanding_window` returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets `OPENBLAS_NUM_THREADS`, `OMP_NUM_THREADS`, `MKL_NUM_THREADS` and `VECLIB_MAXIMUM_THREADS` to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as `posterior_function`, such as the one `bayests_posterior` returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see `add_seed`, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of `add_posterior_forecasts`, come from independent random number streams of the workers, set up with `clusterSetRNGStream` from R's generator. They are reproduced by `set.seed` for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))
```

```
# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 10, burnin = 2)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)
```

`add_posterior_coefficients.modellist`
Posterior Simulation of Model Coefficients

Description

Forwards model input to posterior simulation functions.

Usage

```
## S3 method for class 'modellist'
add_posterior_coefficients(object, ..., cores = 1)
```

Arguments

`object`	an object of class 'modellist'.
`...`	further arguments passed to or from other methods.
`cores`	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in `object` with posterior draws added to each of its models, as described in `add_posterior_coefficients.bvarr`

Parallel simulation

With `cores` above 1 the models are simulated on a cluster of that many worker processes, started with `makeCluster` and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that `use_expanding_window` returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets `OPENBLAS_NUM_THREADS`, `OMP_NUM_THREADS`, `MKL_NUM_THREADS` and `VECLIB_MAXIMUM_THREADS` to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as `posterior_function`, such as the one `bayests_posterior` returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see `add_seed`, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of `add_posterior_forecasts`, come from independent random number streams of the workers, set up with `clusterSetRNGStream` from R's generator. They are reproduced by `set.seed` for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model
model <- create_bvarmodel(e1, p = 1:2, deterministic = 2,
                          iterations = 20, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
```

```
model <- add_posterior_coefficients(model)


# The same on two worker processes
model <- add_posterior_coefficients(model, cores = 2)
```

`add_posterior_forecasts` *Add Forecasts*

Description

Generic function that calculates and adds forecasts.

Usage

```
add_posterior_forecasts(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_posterior_forecasts.bvarmodel`, `add_posterior_forecasts.expandingwindow`, `add_posterior_forecasts.externalforecast`, `add_posterior_forecasts.modellist`.

`add_posterior_forecasts.bvarmodel` *Add Forecasts*

Description

Calculates and adds forecasts to an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
add_posterior_forecasts(
  object,
  forecast_states = NULL,
  scenario = NULL,
  forecast_quantile = NULL,
  ...
)
```

Arguments

`object`	an object of class 'bvarmodel', usually, the result of a call to `add_posterior_coefficients` and `add_forecast_input`.
`forecast_states`	character, what a model with time varying coefficients or stochastic volatility does with them over the forecast horizon. `"simulate"` carries each draw's random walks forward, one step per period, so that the forecasts are draws from the posterior predictive distribution of the estimated model. `"hold"` keeps the coefficients and volatilities at their values in the last sample period, which gives forecasts conditional on no further drift and narrower intervals, and is what earlier versions of the package did. If `NULL` (default), the value in `object$model$forecast_states` is used, and `"simulate"` when there is none. Models with constant coefficients and volatility are unaffected.
`scenario`	an optional data frame with columns `period`, `variable` and `value`, one row per value a variable is held at in a forecast period: a conditional forecast. `period` counts forecast periods from one and `variable` is a name or a position among the endogenous variables. It is stored in `data$forecast$constraints`, which is where the forecast reads it from; set that element to `NULL` to forecast without it again. Available for models with constant coefficients and `error = "wishart"` or `"gamma"`, and for a grid of quantiles.
`forecast_quantile`	for a grid of quantiles (see `quantile_grid` in `create_bvarmodel`), a level in $(0, 1)$ every draw of the forecast is taken at, which gives quantile paths rather than draws from the predictive distribution: the paths of Chavleishvili and Manganelli (2019), whose difference with and without a `scenario` is an impulse response at that quantile. If `NULL` (default), the value in `object$model$forecast_quantile` is used, and the levels are drawn at random when there is none. Zero removes a stored one.
`...`	arguments passed forward to method.

Value

The object in `object` with `posterior$forecast$forecasts` added, a `mcmc` object with one row per draw and Kh columns, stacked by period: the K variables of the first forecast period, then those of the second, and so on. `posterior$forecast` is the group everything the forecast periods produce hangs below, the `errors` of `add_forecast_errors` beside these. `predict` summarises them. A `forecast_states` that was given is stored in `model$forecast_states`.

Simulating the volatility forward needs the variance of the log-volatility innovations, `posterior$u_sigma_inv$sigma`, which `add_posterior_coefficients` stores. A stochastic volatility model fitted with an earlier version of the package lacks it and stops with an error unless `forecast_states = "hold"`.

References

Chavleishvili, S., & Manganelli, S. (2019). Forecasting and stress testing with quantile vector autoregression. *ECB Working Paper*, 2330.

See Also

`bvartools_model` describes the object this returns, element by element.

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecast`
`add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel()`, `add_seed()`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)
```

`add_posterior_forecasts.bvecmodel` *Add Forecasts*

Description

Simulates forecasts of a VEC model, in levels, from its posterior draws.

Usage

```
## S3 method for class 'bvecmodel'
add_posterior_forecasts(object, forecast_states = NULL, ...)
```

Arguments

`object`	an object of class 'bvecmodel', usually, the result of a call to `add_posterior_coefficients` and `add_forecast_input`.
`forecast_states`	character, what a VEC model with time varying coefficients or stochastic volatility does with them over the forecast horizon. `"simulate"` carries each draw's random walks forward, one step per period – the loadings and short-run coefficients, the cointegration vectors through their state equation, the covariance block and the log-volatilities – and rebuilds the VAR in levels from them at every period, so that the forecasts are draws from the posterior predictive distribution of the estimated model. `"hold"` keeps them at their values in the last sample period. If `NULL` (default), the value in `object$model$forecast_states` is used, and `"simulate"` when there is none. Models with constant coefficients and volatility are unaffected.
`...`	further arguments passed to or from other methods.

Details

The forecasts are those of the VAR in levels the VEC model implies. For a model with constant coefficients they are the forecasts `vec_to_var` followed by `add_posterior_forecasts.bvarmodel` gives. For a model with time varying coefficients they are not: its VAR representation has no state equation of its own and holds its coefficients at the last period, while this method steps the states of the VEC model and converts them to levels anew in every forecast period.

Simulating the cointegration vectors forward depends on the units of the data. A step η_t of their state equation moves the error correction term by $\eta_t' w_t$, which for series far from zero – log levels times 100, say – is of the order of their levels rather than of their variation: the random walk intercept described in section 'Prior on the cointegration space' of `cointspace_prior`, carried forward with nothing in the data to restrain it. The forecast is then mostly that drift, and its intervals widen accordingly. A warning names the series in the error correction term that are more than 50 times their standard deviation per period away from zero at the end of the sample; `forecast_states = "hold"` is the alternative. Series near zero, such as interest rates and inflation in percent, do not draw it.

A time varying model estimated on series that `scale_error_correction` centred or scaled cannot be simulated forward and stops with an error unless `forecast_states = "hold"`: the state equation of its cointegration vectors belongs to the transformed series, while `rescale_error_correction` has put the vectors back on the scale of the data and moved the means into the constant of the last sample period only.

Simulating the volatility forward needs `posterior$u_sigma_inv$sigma`, the variance of the log-volatility innovations, which `add_posterior_coefficients` stores. A model with stochastic volatility estimated with an earlier version of the package lacks it and stops with an error unless `forecast_states = "hold"`.

Value

The object in `object` with `posterior$forecast$forecasts` added, a `mcmc` object with one row per draw and Kh columns of forecasts of the levels, stacked by period. `predict` summarises them. A `forecast_states` that was given is stored in `model$forecast_states`.

See Also

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecast`
`add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel()`, `add_seed()`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

---

add_posterior_forecasts.expandingwindow
Add Forecasts

Description

Calculates and adds forecasts to the elements of an object of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
add_posterior_forecasts(object, ..., cores = 1)
```

Arguments

`object`	an object of class 'expandingwindow', usually, the result of a call to `add_posterior_coefficients` and `add_forecast_input`.
`...`	arguments passed forward to method.
`cores`	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in `object` with forecast draws added to each of its models, as described in `add_posterior_forecasts.bvarmode`

Parallel simulation

With `cores` above 1 the models are simulated on a cluster of that many worker processes, started with `makeCluster` and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that `use_expanding_window` returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets `OPENBLAS_NUM_THREADS`, `OMP_NUM_THREADS`, `MKL_NUM_THREADS` and `VECLIB_MAXIMUM_THREADS` to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as `posterior_function`, such as the one `bayests_posterior` returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see `add_seed`, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of `add_posterior_forecasts`, come from independent random number streams of the workers, set up with `clusterSetRNGStream` from R's generator. They are reproduced by `set.seed` for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 10, burnin = 2)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
```

```
# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)
```

`add_posterior_forecasts.modellist` *Add Forecasts*

Description

Calculates and adds forecasts to the elements of an object of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
add_posterior_forecasts(object, ..., cores = 1)
```

Arguments

`object`	an object of class 'modellist', usually, the result of a call to `add_posterior_coefficients` and `add_forecast_input`.
`...`	arguments passed forward to method.
`cores`	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in `object` with forecast draws added to each of its models, as described in `add_posterior_forecasts.bvarmode`

Parallel simulation

With `cores` above 1 the models are simulated on a cluster of that many worker processes, started with `makeCluster` and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that `use_expanding_window` returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets `OPENBLAS_NUM_THREADS`, `OMP_NUM_THREADS`, `MKL_NUM_THREADS` and `VECLIB_MAXIMUM_THREADS` to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread

per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as `posterior_function`, such as the one `bayests_posterior` returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see `add_seed`, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of `add_posterior_forecasts`, come from independent random number streams of the workers, set up with `clusterSetRNGStream` from R's generator. They are reproduced by `set.seed` for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 0:2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)
```

add_posterior_loglik *Add Log-Likelihood*

Description

Generic function that calculates the posterior log-likelihoods of a model.

Usage

```
add_posterior_loglik(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: add_posterior_loglik.bvarmodel, add_posterior_loglik.bvecmodel, add_posterior_loglik.expandin
add_posterior_loglik.externalforecast, add_posterior_loglik.modellist.

add_posterior_loglik.bvarmodel *Add Log-Likelihood*

Description

Calculates and adds posterior log-likelihoods to an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
add_posterior_loglik(object, ...)
```

Arguments

object	an object of class 'bvarmodel', usually, the result of a call to add_posterior_coefficients.
...	additional arguments.

Value

The object in `object` with `posterior$loglik` added, a `mcmc` object with one row per draw and one column per period of the training sample, holding the pointwise log-likelihood that `selection_criteria` uses.

See Also

`bvartools_model` describes the object this returns, element by element.

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecas`
`add_posterior_forecasts.bvecmodel()`, `add_posterior_loglik.bvecmodel()`, `add_seed()`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Add log-likelihoods
object <- add_posterior_loglik(object)
```

`add_posterior_loglik.bvecmodel` *Add Log-Likelihood*

Description

Calculates and adds posterior log-likelihoods to an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
add_posterior_loglik(object, ...)
```

Arguments

object	an object of class 'bvecmodel', usually, the result of a call to add_posterior_coefficients.
...	additional arguments.

Value

The object in object with posterior$loglik added, a mcmc object with one row per draw and one column per period of the training sample, holding the pointwise log-likelihood that selection_criteria uses.

See Also

Other posterior simulation: add_forecast_input.bvarmodel(), add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecas add_posterior_forecasts.bvecmodel(), add_posterior_loglik.bvarmodel(), add_seed(), bayests_files(), bayests_posterior(), bvar(), bvec(), chain_diagnostics(), predict.bvecmodel()

Examples

```
# Load data
data("e6")

# Create model
model <- create_bvecmodel(e6, p = 4, r = 1,
                          const = "unrestricted",
                          seasonal = "unrestricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add log-likelihoods
model <- add_posterior_loglik(model)
```

`add_posterior_loglik.expandingwindow` *Add Log-Likelihood*

Description

Forwards its elements to methods used to calculate posterior log-likelihoods.

Usage

```
## S3 method for class 'expandingwindow'
add_posterior_loglik(object, ..., cores = 1)
```

Arguments

`object`	an object of class 'expandingwindow', usually, the result of a call to `add_posterior_coefficients`.
`...`	further arguments passed to or from other methods.
`cores`	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in `object` with log-likelihood draws added to each of its models, as described in `add_posterior_loglik.bvarmodel`.

Parallel simulation

With `cores` above 1 the models are simulated on a cluster of that many worker processes, started with `makeCluster` and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that `use_expanding_window` returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets `OPENBLAS_NUM_THREADS`, `OMP_NUM_THREADS`, `MKL_NUM_THREADS` and `VECLIB_MAXIMUM_THREADS` to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as `posterior_function`, such as the one `bayests_posterior` returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see `add_seed`, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds

some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of add_posterior_forecasts, come from independent random number streams of the workers, set up with clusterSetRNGStream from R's generator. They are reproduced by set.seed for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add log-likelihoods
model <- add_posterior_loglik(model)
```

---

add_posterior_loglik.modellist
Add Log-Likelihood

Description

Forwards its elements to methods used to calculate posterior log-likelihoods.

Usage

```
## S3 method for class 'modellist'
add_posterior_loglik(object, ..., cores = 1)
```

Arguments

object	a list of class 'modellist', usually, the result of a call to add_posterior_coefficients.
...	further arguments passed to or from other methods.
cores	the number of worker processes the models are simulated on. Defaults to 1, which simulates them one after the other in this session. See section 'Parallel simulation'.

Value

The object in object with log-likelihood draws added to each of its models, as described in add_posterior_loglik.bvarmodel.

Parallel simulation

With cores above 1 the models are simulated on a cluster of that many worker processes, started with makeCluster and stopped when the call returns, but on no more workers than there are models. Nested lists, such as the 'modellist' of 'expandingwindow' lists that use_expanding_window returns for several specifications, are shared out model by model.

Each worker runs with a single BLAS thread: the function sets OPENBLAS_NUM_THREADS, OMP_NUM_THREADS, MKL_NUM_THREADS and VECLIB_MAXIMUM_THREADS to 1 while the workers start, whatever this session has set, and puts them back afterwards. An optimised BLAS would otherwise start one thread per core in every worker, and the samplers do not become faster with more threads. The thread count of this session is not changed.

The workers use the library paths of this session and load the installed version of the package, together with every package that provides a method the models of the list need – **dfmtools** for its dynamic factor models, for example. A function passed as posterior_function, such as the one bayests_posterior returns, is sent to the workers along with the variables of the environment it was created in, and must not rely on anything else of this session.

Coefficient draws do not depend on the number of workers. Every model is simulated with its own seed, see add_seed, and a model without one is given one from R's random number generator before it is sent off. The draws equal those of a simulation on one core in a session whose BLAS runs on one thread, as R's reference BLAS always does. An optimised BLAS such as OpenBLAS rounds some results differently in the last digits on one thread than on several, and a Markov chain carries such a difference forward, so a session running it on several threads gets different draws from the same posterior. Draws that are not seeded per model, such as those of add_posterior_forecasts, come from independent random number streams of the workers, set up with clusterSetRNGStream from R's generator. They are reproduced by set.seed for a given number of workers, but differ from those of a simulation on one core.

Starting the workers takes a moment, so a cluster pays off for lists of models that take longer than that to simulate.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model
model <- create_bvarmodel(e1, p = 1:2, deterministic = 2,
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Add log-likelihoods
object <- add_posterior_loglik(object)
```

---

add_predictive_loglik *Add Predictive Log-Likelihood*

Description

Adds the draws of the one-step-ahead log predictive density to the windows of an expanding window exercise, which `selection_criteria` sums to the log predictive likelihood.

Usage

```
add_predictive_loglik(object, ...)

## S3 method for class 'expandingwindow'
add_predictive_loglik(object, ...)

## S3 method for class 'modellist'
add_predictive_loglik(object, ...)
```

Arguments

`object`	an object of class `"expandingwindow"` whose windows hold posterior draws, or a `"modellist"` of such objects, as returned by `use_expanding_window`.
`...`	further arguments passed to or from other methods.

Details

The windows of an expanding window exercise grow by one period at a time, so window $i + 1$ holds exactly one observation that window i has not seen. For every window but the last, the function evaluates the density of that observation, Δy_t, given the regressors of period t, at each posterior draw of the window, and stores the draws in element `predictive` of the window, together with the period t. The log of their mean is the log predictive density $\ln p(\Delta y_t | y_{t-1}, \ldots, y_1)$, and its sum over the windows is the log predictive likelihood of Geweke and Amisano (2011), which Koop, León-González and Strachan (2011) use to choose between time varying cointegration models and their ranks.

A draw of a model with constant coefficients and a constant error covariance describes period t as it describes the sample. Everything that follows a state equation is carried one period forward with that equation first:

- time varying coefficients by one step of their random walk with the drawn state variance, and a coefficient that variable selection excluded stays at zero;
- a time varying cointegration space by $\beta_t = \rho(I_r \otimes P_\tau)\beta_{t-1} + \eta_t$, $\eta_t \sim N(0, I)$, with the drawn ρ or the one of the prior, where P_τ is the identity unless the prior centres the space;
- time varying error covariances by one step of their random walk;
- stochastic volatilities by one step of the random walk of the log variances. Its state variance is taken from `posterior$u_sigma_inv$sigma` if the sampler stored it, and is otherwise drawn from its conditional posterior given the drawn path of the log variances and the prior in `priors$u_sigma`.

Unlike the pointwise log-likelihood of `add_posterior_loglik`, whose states have seen the observation they are evaluated at, the predictive density conditions only on the data before period t. This is what makes it the criterion for models whose coefficients or variances follow a state equation, where leave-one-out importance sampling fails for exactly the periods a path bends towards.

The function is available for VEC models, which includes the rank zero models in differences that ranks are compared with, and for VAR models, whose regressors are the lags of the endogenous variables, the exogenous variables and the deterministic terms of the period that is predicted. A VAR model is the case of a VEC model without an error correction term, so the density is the same expression with the cointegration block left out. The windows of a VEC model must have been simulated on error correction terms that are neither scaled nor centred, or put back with `rescale_error_correction` first, and structural models are not supported.

The windows need not share their regressors. `use_expanding_window` leaves an impulse, step or data dummy of `add_dummy_variables` out of the windows that end before its event, so the window just before the event has fewer regressors than the observation the next window adds. The density of that observation uses only the regressors of the window that predicts it: the regressors of the observation are matched to the ones of the window by name, and a dummy the window does not have is left out, because a forecast made at the end of the window could not have known about the event. The event period is therefore scored as an ordinary forecast, and the dummy only enters the densities of the windows that contain its event. This applies to VAR and VEC models alike.

The expression is the normal density of the observation, so the function is available for the algorithms whose observation is normal given the parameters of a draw, and refuses the others. The asymmetric Laplace algorithms of quantile estimation, `"VarNormalAld"` and `"VarTvpAld"`, are refused: the precision their samplers store is the one of the normal that their scale mixture conditions

on period by period, not the density of an observation, whose mixing variable would have to be integrated out.

The methods for a single fitted model, `add_predictive_loglik.bvarmodel` and `add_predictive_loglik.bvecmodel`, score a forecast against the observations its horizon realised instead. That is the same statistic over a different set of periods – one step ahead densities, each conditioning on the realised history before it – and it is BayesTS that computes it rather than the R code here. It is stored in `posterior$forecast$loglik`, beside the forecasts it scores, rather than in `predictive`.

Value

The object in `object`, with element `predictive` added to every window but the last. It is a list with `loglik`, the draws of the log predictive density, and `period`, the time of the predicted observation.

References

Geweke, J., & Amisano, G. (2011). Hierarchical Markov normal mixture models with applications to financial asset returns. *Journal of Applied Econometrics, 26*(1), 1–29. doi:10.1002/jae.1119

Koop, G., León-González, R., & Strachan, R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

See Also

Methods for a single fitted model: `add_predictive_loglik.bvarmodel`, `add_predictive_loglik.bvecmodel`.

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `choose_best_model.selcritlist()`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `map_models()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Create model
model <- create_bvecmodel(e6, p = 2, r = 1, const = "unrestricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 0.0001))

# Estimate the model on the last four expanding windows
model <- use_expanding_window(model, start = c(1998, 1))
model <- add_initial_values(model)
model <- add_posterior_coefficients(model)
```

```
# One-step-ahead log predictive densities
model <- add_predictive_loglik(model)

# The log predictive likelihood is criterion "LPL"
selection_criteria(model)
```

`add_predictive_loglik.bvarmodel` *Add the Log Predictive Density of a Forecast*

Description

Scores the forecast of an object of class 'bvarmodel' against the observations its horizon realised.

Usage

```
## S3 method for class 'bvarmodel'
add_predictive_loglik(object, test_sample = NULL, ...)
```

Arguments

`object`	an object of class 'bvarmodel', usually, the result of a call to `add_posterior_forecasts`.
`test_sample`	a time-series object used as test data. If `NULL` (default), the values in `data$test$y` of the object are used, which is what a model carries after it has been scored once and what a model read from a file was written with.
`...`	further arguments passed to or from other methods.

Details

The draws of the log predictive density of period $T+i$ are stored in `posterior$forecast$loglik`, one row per draw and one column per scored period, beside the paths in `posterior$forecast$forecasts` that they score. The numbers are BayesTS's own: this method fills `data$test$y` and hands the object to the same C++ that computes the score when the `bayests` programme is run on a model file.

Each column conditions on the observations the periods before it realised, not on the path the forecast simulated. The log of the mean of the draws of a column is therefore the one step ahead predictive density given everything known up to that period, and those sum over the columns to $\ln p(y_{T+1}, \ldots, y_{T+h} | y_1, \ldots, y_T)$, the log predictive likelihood of the whole realised stretch, which `selection_criteria` reports as `LPL`. Scoring against a simulated history instead would give the marginal density of each period on its own, which is a different quantity and one whose columns could not be added up.

With the history realised rather than simulated, the regressors of the scored periods do not depend on the draw. The score is then the model's own pointwise log-likelihood over those periods – the same expression as `add_posterior_loglik` evaluates, over a different sample – so nothing

is written down twice and the two cannot drift apart. What still moves with the draw is everything that follows a state equation: time varying coefficients, time varying error covariances and stochastic volatilities take one step of their random walk per scored period, in the order and subject to the same `model$forecast_states` that `add_posterior_forecasts` steps them in. Under `"simulate"` those steps are drawn, so the score is drawn too and two runs give two answers, exactly as two runs of a forecast do.

Fewer realised periods than the horizon is not an error. The periods that are there are the ones that can be scored, and the rest of the forecast is left alone. A `test_sample` that does not reach the forecast at all leaves the object unchanged, which is what a model estimated to the end of a series looks like.

Structural models are refused: their regressors include the contemporaneous observations, so the realised row is not built from its lags alone, and their density carries the Jacobian of A_0 besides. So are the asymmetric Laplace algorithms of quantile estimation, which do not forecast in the first place.

Value

The object in `object` with `posterior$forecast$loglik` added, a `mcmc` object with one row per draw and one column per scored period, and with the values it was scored against in `data$test$y`.

See Also

`bvartools_model` describes the object this returns, element by element.

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `choose_best_model.selcritlist()`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `map_models()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1978, 4))

# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

model <- add_initial_values(model)
model <- add_posterior_coefficients(model)

# Forecast the four periods that were held back
```

```
model <- add_forecast_input(model, n_ahead = 4)
model <- add_posterior_forecasts(model)

# Score them against what those periods realised
model <- add_predictive_loglik(model, test_sample = orig)
dim(model[["posterior"]][["forecast"]][["loglik"]])

# The log predictive likelihood is criterion "LPL"
selection_criteria(model)[["LPL"]]
```

`add_predictive_loglik.bvecmodel`

Add the Log Predictive Density of a Forecast

Description

Scores the forecast of an object of class 'bvecmodel' against the levels its horizon realised.

Usage

```
## S3 method for class 'bvecmodel'
add_predictive_loglik(object, test_sample = NULL, ...)
```

Arguments

`object`	an object of class 'bvecmodel', usually, the result of a call to `add_posterior_forecasts`.
`test_sample`	a time-series object of the endogenous variables, in levels, that covers the forecast periods. If `NULL` (default), the values in `data$test$y` of the object are used.
`...`	further arguments passed to or from other methods.

Details

The forecasts of a VEC model are of the levels, so the score is of the levels too, and it is taken in the VAR representation `vec_to_var` would give: every draw is converted to its level VAR coefficients and the realised row is evaluated against the lags of the realised rows before it. That is the same route `add_forecast_errors.bvecmodel` takes and the same one the forecast itself took, so the three describe one model rather than three.

Everything else is as `add_predictive_loglik.bvarmodel` describes it, including what each column conditions on and how a cointegration space or a volatility that moves with time is carried across the horizon. A model whose error correction term was scaled or centred has to be put back with `rescale_error_correction` first, since the score is taken against the levels themselves.

Value

The object in `object` with `posterior$forecast$loglik` added, a `mcmc` object with one row per draw and one column per scored period, and with the levels it was scored against in `data$test$y`.

See Also

bvartools_model describes the object this returns, element by element.

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcritlist(), create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
# Load data
data("e6")
e6 <- e6 * 100
train <- window(e6, end = c(1997, 4))

# Create model
model <- create_bvecmodel(train, p = 2, r = 1, const = "unrestricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 0.0001))

model <- add_initial_values(model)
model <- add_posterior_coefficients(model)

# Forecast the periods that were held back and score them
model <- add_forecast_input(model, n_ahead = 4)
model <- add_posterior_forecasts(model)
model <- add_predictive_loglik(model, test_sample = e6)

dim(model[["posterior"]][["forecast"]][["loglik"]])
```

---

add_prior_options *Adaptive Priors, Stationarity and the Steady State*

Description

Adds the options of the prior that act on top of the one add_priors sets: hyperparameters estimated from the data, a restriction of the coefficients to a stationary model, a prior on the unconditional mean in place of one on the intercept, and the prior of the error of soft constraints.

Usage

```
add_prior_options(
  object,
  shrinkage = "none",
  stationary = FALSE,
  steady_state = NULL,
  constraints = NULL
)
```

Arguments

object	an object of class 'bvarmodel' or 'modellist', usually the result of add_priors.
shrinkage	a character, "none" (default), "minnesota", "horseshoe" or "normal_gamma", or a list with element type naming one of them and elements group, shape, rate and, for "normal_gamma", theta or theta_rate. See 'Details'.
stationary	logical. If TRUE, every draw of the coefficients is a stationary model. Defaults to FALSE.
steady_state	an optional list with elements mu, the prior mean of the unconditional mean of the endogenous variables, and v_i, its prior precision, each a number or one value per variable. See 'Details'.
constraints	an optional list with elements shape and rate, the gamma prior of the error precision of soft constraints, a number or one value per series named in argument soft of create_bvarmodel. Defaults to list(shape = 3, rate = 0.01) where the model has soft constraints.

Details

Argument shrinkage lets the data decide how tightly the prior of add_priors shrinks the coefficients of the lagged endogenous variables. Their prior variances are multiplied by scales that are drawn with the coefficients:

- "minnesota": one scale per group of coefficients with an inverse gamma prior of shape and rate, the hierarchical Minnesota prior of Chan (2021). By default own lags form group 1 and the lags of the other variables group 2, with shape = 3 and rate = 2, a prior mean of one: the prior of add_priors is the centre of the one estimated.
- "horseshoe": the horseshoe prior of Carvalho, Polson and Scott (2010), a half-Cauchy scale per group and one per coefficient, drawn as in Makalic and Schmidt (2016). By default all lags form one group.
- "normal_gamma": the normal-gamma prior of Griffin and Brown (2010) as Huber and Feldkircher (2019) put it on a VAR. Each coefficient's scale ψ_j has a gamma prior of shape theta and rate $\theta\lambda_g/2$, and each group's λ_g a gamma prior of shape and rate. By default the lags of order l form group l, with theta = 0.1 and shape = rate = 0.01. theta is held fixed; the smaller it is, the more the prior pushes small coefficients to zero while leaving large ones alone. Give theta_rate instead to draw θ under an exponential prior of that rate, by a Metropolis-Hastings step tuned during the burn-in, as Huber and Feldkircher (2019) do; the draws come back as posterior$a$theta, and the chain needs a burn-in to tune the step.

Coefficients in group 0 – by default the deterministic terms and the exogenous variables – keep the prior of `add_priors`. A `group` of one's own has one element per coefficient, in the order of the columns of `data$train$z`. The prior must be diagonal wherever it shrinks.

Argument `stationary` redraws a draw of the coefficients that is not stationary up to 100 times, and keeps the previous draw if none of them is, which leaves the posterior restricted to the stationary region invariant.

Argument `steady_state` puts the prior on the unconditional mean μ of the endogenous variables rather than on the intercept, which is $(I - \sum_i A_i)\mu$ in every draw, following Villani (2009). A prior belief about the level a series returns to is more often available than one about an intercept. The model must have lags and an intercept and nothing else, and cannot be combined with `shrinkage`. The draws of μ come back as `posterior$mu`.

The three are available for models with constant coefficients and `error = "wishart"`, `"gamma"`, `"gamma+covar"`, `"sv"` or `"sv+covar"`, without variable selection or a structural form.

Value

The object in `object` with `model$shrinkage`, `model$stationary` and `model$steady_state` set where they apply, and the priors in `priors$a$shrinkage`, `priors$mu` and `priors$constraints`.

References

Carvalho, C. M., Polson, N. G., & Scott, J. G. (2010). The horseshoe estimator for sparse signals. *Biometrika, 97*(2), 465–480. doi:10.1093/biomet/asq017

Griffin, J. E., & Brown, P. J. (2010). Inference with normal-gamma prior distributions in regression problems. *Bayesian Analysis, 5*(1), 171–188. doi:10.1214/10BA507

Huber, F., & Feldkircher, M. (2019). Adaptive shrinkage in Bayesian vector autoregressive models. *Journal of Business & Economic Statistics, 37*(1), 27–39. doi:10.1080/07350015.2016.1256217

Chan, J. C. C. (2021). Minnesota-type adaptive hierarchical priors for large Bayesian VARs. *International Journal of Forecasting, 37*(3), 1212–1226. doi:10.1016/j.ijforecast.2021.01.002

Makalic, E., & Schmidt, D. F. (2016). A simple sampler for the horseshoe estimator. *IEEE Signal Processing Letters, 23*(1), 179–182. doi:10.1109/LSP.2015.2503725

Villani, M. (2009). Steady-state priors for vector autoregressions. *Journal of Applied Econometrics, 24*(4), 630–650. doi:10.1002/jae.1065

See Also

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_priors.bvarmodel()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bv`
`use_expanding_window.bvecmodel()`

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100

model <- create_bvarmodel(e1, p = 2, iterations = 20, burnin = 10)
```

```
model <- add_priors(model, coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
model <- add_prior_options(model, shrinkage = "minnesota", stationary = TRUE)
model <- add_initial_values(model)
model <- add_posterior_coefficients(model)
```

`add_priors` *Add Priors to Bayesian Models*

Description

A generic function used to generate prior vectors and matrices. The function invokes particular methods which depend on the class of the first argument.

Usage

```
add_priors(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_priors.bvarmodel`, `add_priors.bvecmodel`, `add_priors.expandingwindow`, `add_priors.externalfor`
`add_priors.modellist`.

`add_priors.bvarmodel` *Add Priors to Bayesian Models*

Description

Adds prior specifications to a BVAR model, which was produced by function `create_bvarmodel`.

Usage

```
## S3 method for class 'bvarmodel'
add_priors(object, coef, sigma, varsel = NULL, ...)
```

Arguments

`object`	a list of class 'bvarmodel'.
`coef`	a named list of prior specifications for the coefficients. It has no default and must contain at least `v_i` or `minnesota`. Variances are specified as precisions, i.e. as inverses of the variances. See 'Details'.
`sigma`	a named list of prior specifications for the error term. It has no default, and the elements it must contain depend on argument `error` of `create_bvarmodel`. See 'Details'.
`varsel`	a named list of prior specifications for the variable selection algorithm. Required if the model was created with `varsel = "ssvs"` or `"bvs"`, and not allowed otherwise. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

None of the arguments `coef`, `sigma` and `varsel` provides default hyperparameters: every value that a model needs must be given in the list it belongs to. A missing required element raises an error, as does an element that is not listed below.

Argument `coef` can contain the following elements:

`v_i` a non-negative numeric specifying the prior precision of the coefficients, where 0 gives an uninformative prior. Required unless `minnesota` is given. It is also required together with `minnesota` if `error` is `"gamma+covar"` or `"sv+covar"`, where it sets the prior precision of the error covariance coefficients. The precisions of the other coefficients are taken from `minnesota` if it is given, and from `varsel` for SSVS.

`v_i_det` a numeric specifying the prior precision of coefficients corresponding to deterministic terms. If it is not given, `v_i` is used. Not used if `minnesota` is given or SSVS is applied.

`coint_var` a logical specifying whether the prior mean of the first own lag of an endogenous variable should be set to 1, which is commonly used for cointegrated VAR models. Default is `FALSE`. Ignored, with a warning, under SSVS, which centres the prior of every coefficient it selects at zero; BVS keeps the prior mean.

`const` a numeric or character specifying the prior mean of coefficients, which correspond to the intercept. If a numeric is provided, all prior means are set to this value. If `coef$const = "mean"`, the mean of the respective endogenous variable is used as prior mean. If `coef$const = "first"`, the first values of the respective endogenous variable is used as prior mean. This can be useful for trend estimation when using a TVP model.

`minnesota` a named list containing the parameters for the calculation of the Minnesota prior. It must contain `kappa1`, `kappa2` and `kappa4`, and `kappa3` if the model has exogenous variables. For the endogenous variable i the prior variance of the lth lag of regressor j is obtained as

$$\frac{\kappa_1}{l^2} \text{ for own lags of endogenous variables,}$$

$$\frac{\kappa_1 \kappa_2}{l^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for endogenous variables other than own lags,}$$

$$\frac{\kappa_1 \kappa_3}{(l+1)^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for exogenous variables,}$$

$\kappa_1\kappa_4\sigma_i^2$ for deterministic terms,

where σ_i is the residual standard deviation of variable i of an unrestricted LS estimate. For exogenous variables σ_i is the sample standard deviation. If the model does not contain exogenous variables, `kappa3` will be ignored.

`max_var` a positive numeric specifying the maximum prior variance of the coefficients of non-deterministic variables in the Minnesota prior. Larger prior variances are set to this value. Only used if `minnesota` is given.

`shape` a numeric specifying the shape of the gamma prior on the precisions, the inverse error variances, of the state equation, whose mean is `shape / rate`. Required for models with time varying parameters unless `omega_v` is given, and not used otherwise.

`rate` a numeric specifying the rate of the gamma prior on the precisions of the state equation. Required together with `shape`.

`rate_det` a numeric specifying the prior rate parameter of the error variances of the state equation for coefficients, which correspond to deterministic terms. If it is not given, `rate` is used. Only used for models with time varying parameters.

`omega_v` a positive numeric, in place of `shape` and `rate`: the variance of a normal prior on the signed standard deviation ω of the state innovations, $\omega \sim N(0, V_\omega)$, which is also the prior mean of the state variance ω^2. This is the non-centred parameterisation of Frühwirth-Schnatter and Wagner (2010), which makes a constant coefficient an interior point of the prior and lets the posterior carry the Savage-Dickey test for time variation of Chan (2018); see Details. It applies to the covariance coefficients as well. Only for models with time varying parameters, `tvp = TRUE`, whatever their `error`.

Argument `sigma` must contain the elements that belong to the `error` of the model:

- `"wishart"`: `df` and `scale`. Not available for structural models.
- `"gamma"` and `"gamma+covar"`: `shape` and `rate`.
- `"sv"` and `"sv+covar"`: `mu`, `v_i`, `shape`, `rate`, `state_variance` and `offset`; with `tvp = TRUE`, `omega_v` may take the place of `shape` and `rate`.
- `"ald"`: `shape` and `rate`.

The elements are

`df` a positive integer, or a character expression in `k`, the number of endogenous variables, such as `"k"` or `"k + 3"`, specifying the prior degrees of freedom of the inverse Wishart prior.

`scale` a positive numeric specifying the prior error variance of the endogenous variables in the inverse Wishart prior.

`shape` for `"gamma"` and `"gamma+covar"` a non-negative numeric, or a character expression in `k` as for `df`, specifying the shape of the gamma prior on the error precisions, the inverse error variances, whose mean is `shape / rate`. For `"ald"` a positive numeric specifying the shape of the inverse gamma prior on the scale of the asymmetric Laplace distribution. For both either one value or one per endogenous variable. For models with stochastic volatility a numeric specifying the shape of the gamma prior on the precision of the state equation of the log-volatilities.

`rate` a positive numeric specifying the rate that corresponds to `shape`, for `"gamma"`, `"gamma+covar"` and `"ald"` either one value or one per endogenous variable.

`mu` numeric of the prior mean of the initial state of the log-volatilities. Only used for models with time varying volatility.

`v_i` numeric of the prior precision of the initial state of the log-volatilities. Only used for models with time varying volatility.

`state_variance` numeric of the initial draw for the variance of the log-volatilities. Only used for models with time varying volatility.

`offset` numeric of the constant, which is added before taking the log of the squared errors. Only used for models with time varying volatility.

`omega_v` a positive numeric, in place of `shape` and `rate`: the variance of a normal prior on the signed standard deviation of the log-volatility innovations, as `coef$omega_v` is for the coefficients. Only for models with time varying parameters and stochastic volatility.

Under `omega_v` a random walk $x_t = x_{t-1} + v_t$, $v_t \sim N(0, \omega^2)$, is estimated as $x_t = x_0 + \omega\tilde{x}_t$ with $\tilde{x}_t$ a standard random walk. The hypothesis that x_t does not move is then $\omega = 0$, a point inside the prior, and its Bayes factor is the Savage-Dickey density ratio $p(\omega = 0)/p(\omega = 0|y)$, which a single run of the time varying model estimates (Chan 2018). Each block of the posterior estimated this way holds, beside `sigma`, the draws of ω in `omega`, and the log density at zero of its conditional posterior, per state in `omega_log_zero` and for the whole block in `omega_log_zero_joint`. The log Bayes factor in favour of time variation in state i is $\log N(0; 0, V_{\omega,i})$ less the log of the average over the draws of `exp(omega_log_zero[, i])`. The joint one compares "every state of the block moves" with "none does" rather than "at least one moves". Each block chooses its prior for itself, so the volatilities can be tested while the coefficients keep `shape` and `rate`. For structural models only a gamma prior or stochastic volatility specification is allowed.

Argument `varsel` can contain the following elements:

`inprior` a numeric between 0 and 1 specifying the prior probability of a variable to be included in the model.

`covar` logical indicating if the variable selection algorithm should also be applied to the error covariance matrix.

`exclude_det` logical indicating if deterministic terms should be excluded from the variable selection algorithm.

`minnesota` a numeric vector of length 4 containing parameters for the calculation of the Minnesota-like inclusion priors. See below.

`tau` a numeric vector of two elements containing the prior standard errors of restricted variables (τ_0) as its first element and unrestricted variables (τ_1) as its second. Only used for SSVS.

`semiautomatic` an numeric vector of two elements containing the factors by which the standard errors associated with an unconstrained least squares estimate of the model are multiplied to obtain the prior standard errors of restricted (τ_0) and unrestricted (τ_1) variables, respectively. This is the semiautomatic approach described in George et al. (2008). Only used for SSVS.

In the case of SSVS, either `tau` or `semiautomatic` must be specified.

If `varsel$minnesota` is specified, prior inclusion probabilities are calculated in a Minnesota-like fashion as

$\frac{\kappa_1}{l}$ for own lags of endogenous variables,
$\frac{\kappa_2}{l}$ for other endogenous variables,

$\frac{\kappa_3}{1+l}$ for exogenous variables,
κ_2 for contemporaneous endogenous variables of a structural model,
κ_4 for deterministic variables,

for lag l with κ_1, κ_2, κ_3, κ_4 as the first, second, third and forth element in `varsel$minnesota`, respectively.

Value

The object in `object` with the element `priors` added, a list with

`a` the prior of the coefficients: `type`, the $M \times 1$ prior means `mu` and the $M \times M$ prior precision matrix `v_inv`, where M is the number of coefficients. With variable selection also `inprior` and `include`, for SSVS `tau0` and `tau1`, and for TVP models `shape` and `rate` of the state variances, or `omega_v`.

`psi` for `error = "gamma+covar"` or `"sv+covar"`, the prior of the error covariance coefficients with `type`, `mu`, `v_inv` and `varsel`, and the variable selection and state variance elements as for `a`.

`u_sigma` the prior of the error variances with its `type`: `"wishart"` with `df` and `scale`, `"gamma"` with `shape` and `rate`, or `"sv"` with `mu`, `v_inv`, `shape`, `rate` (or `omega_v`), `sigma` and `offset`. With a Minnesota prior it also holds `u_sigma_inv`, the inverse error covariance matrix the prior was scaled with. Not present for `error = "ald"`.

`u_scale` for `error = "ald"`, the prior of the scales of the asymmetric Laplace distribution with `type`, `shape` and `rate`.

References

Chan, J. C. C. (2018). Specification tests for time-varying parameter models with stochastic volatility. *Econometric Reviews, 37*(8), 807–823. doi:10.1080/07474938.2016.1167948

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Frühwirth-Schnatter, S., & Wagner, H. (2010). Stochastic model specification search for Gaussian and partial non-Gaussian state space models. *Journal of Econometrics, 154*(1), 85–100. doi:10.1016/j.jeconom.2009.07.003

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

See Also

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bv`
`use_expanding_window.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burn-in should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
```

`add_priors.bvecmodel` *Add Priors to Bayesian Models*

Description

Adds prior specifications to a list of models, which was produced by function `create_bvecmodel`.

Usage

```
## S3 method for class 'bvecmodel'
add_priors(object, coef, coint = NULL, sigma, varsel = NULL, ...)
```

Arguments

`object`	a list of class 'bvecmodel'.
`coef`	a named list of prior specifications for coefficients that do not determine the cointegration space. It has no default and must contain at least `v_i` or `minnesota`. Variances are specified as precisions, i.e. as inverses of the variances. See 'Details'.
`coint`	a named list of prior specifications for coefficients determining the cointegration space of VEC models. It has no default and is required for a positive cointegration rank. A model of rank zero has no cointegration space and needs none; one that is given is checked and ignored, so that a list of models over several ranks can share the same arguments. See 'Details'.
`sigma`	a named list of prior specifications for the error term. It has no default, and the elements it must contain depend on argument `error` of `create_bvecmodel`. See 'Details'.
`varsel`	a named list of prior specifications for the variable selection algorithm. Required if the model was created with `varsel = "ssvs"` or `"bvs"`, and not allowed otherwise. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

None of the arguments `coef`, `coint`, `sigma` and `varsel` provides default hyperparameters: every value that a model needs must be given in the list it belongs to. A missing required element raises an error, as does an element that is not listed below.

Argument `coef` can contain the following elements:

`v_i` a non-negative numeric specifying the prior precision of the coefficients, where 0 gives an uninformative prior. Required unless `minnesota` is given. It is also required together with `minnesota` if `error` is `"gamma+covar"` or `"sv+covar"`. The precisions of the other coefficients are taken from `minnesota` if it is given, and from `varsel` for SSVS.

`v_i_det` a numeric specifying the prior precision of coefficients corresponding to deterministic terms. If it is not given, `v_i` is used. Not used if `minnesota` is given or SSVS is applied.

`const` a numeric or character specifying the prior mean of coefficients, which correspond to the intercept. If a numeric is provided, all prior means are set to this value. If `coef$const = "mean"`, the mean of the respective endogenous variable is used as prior mean. If `coef$const = "first"`, the first values of the respective endogenous variable is used as prior mean.

`minnesota` a named list containing the parameters for the calculation of the Minnesota prior. It must contain `kappa1`, `kappa2` and `kappa4`, and `kappa3` if the model has exogenous variables. For the endogenous variable i the prior variance of the lth lag of regressor j is obtained as

$$\frac{\kappa_1}{l^2} \text{ for own lags of endogenous variables,}$$

$$\frac{\kappa_1 \kappa_2}{l^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for endogenous variables other than own lags,}$$

$$\frac{\kappa_1 \kappa_3}{(l+1)^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for exogenous variables,}$$

$$\kappa_1 \kappa_4 \sigma_i^2 \text{ for deterministic terms,}$$

where σ_i is the residual standard deviation of variable i of an unrestricted LS estimate. For exogenous variables σ_i is the sample standard deviation. If the model does not contain exogenous variables, `kappa3` will be ignored. The function only provides priors for the non-cointegration part of the model. However, the residual standard errors σ_i are based on an unrestricted LS regression of the endogenous variables on the error correction term and the non-cointegration regressors.

`max_var` a positive numeric specifying the maximum prior variance of the coefficients of non-deterministic variables in the Minnesota prior. Larger prior variances are set to this value. Only used if `minnesota` is given.

`shape` a numeric specifying the shape of the gamma prior on the precisions, the inverse error variances, of the state equation, whose mean is `shape / rate`. Required for models with time varying parameters and not used otherwise.

`rate` a numeric specifying the rate of the gamma prior on the precisions of the state equation. Required for models with time varying parameters and not used otherwise.

`rate_det` a numeric specifying the prior rate parameter of the error variances of the state equation for coefficients, which correspond to deterministic terms. If it is not given, `rate` is used. Only used for models with time varying parameters.

`rate_alpha` a numeric specifying the rate of the gamma prior on the precisions of the state equation of the loadings. If it is not given, `rate` is used. Only used for models with time varying parameters and a positive rank. The loadings multiply the levels in the error correction term, so a drift that is negligible for a coefficient on a differenced regressor moves the fitted value by much more; a rate several orders of magnitude below `rate` keeps their drift from absorbing the residuals while the other coefficients vary. It does not reach the steps of the cointegration vectors, which can absorb the residuals as well; see section 'Prior on the cointegration space'.

`omega_v` a positive numeric, in place of `shape`, `rate`, `rate_det` and `rate_alpha`: the variance of a normal prior on the signed standard deviation of the state innovations of the coefficients, loadings included unless `omega_v_alpha` is given, and of the covariance coefficients – the non-centred parameterisation that lets the posterior carry the test for time variation of `time_variation_test`; see `add_priors.bvarmodel` for the model and the draws it adds. The cointegration space keeps its state equation. Only for models with time varying parameters, `tvp = TRUE`, whatever their `error`.

`omega_v_alpha` a positive numeric, the `omega_v` of the loadings, as `rate_alpha` is their rate under the centred prior. If it is not given, `omega_v` is used. The loadings multiply the levels in the error correction term, so a standard deviation of their innovations that is small for a coefficient on a differenced regressor moves the fitted value by much more, and the test for time variation of `time_variation_test` compares each loading's posterior at zero with its own prior: a prior on the wrong scale makes the Bayes factor of a loading say more about the prior than about the loading. Only used with `omega_v` and a positive rank.

Argument `coint` specifies the prior on the cointegration space. Its elements, and the priors they give, are described in section 'Prior on the cointegration space' below, which is shared with `cointspace_prior`.

Argument `sigma` must contain the elements that belong to the `error` of the model:

- `"wishart"`: `df` and `scale`. Not available for structural models.
- `"gamma"` and `"gamma+covar"`: `shape` and `rate`.
- `"sv"` and `"sv+covar"`: `mu`, `v_i`, `shape`, `rate`, `state_variance` and `offset`; with `tvp = TRUE`, `omega_v` may take the place of `shape` and `rate`, as `coef$omega_v` does for the coefficients.

The elements are

`df` a positive integer, or a character expression in `k`, the number of endogenous variables, such as `"k"` or `"k + 3"`, specifying the prior degrees of freedom of the inverse Wishart prior. The samplers add the rank r of the cointegration matrix to the posterior degrees of freedom, as the prior of the loadings requires.

`scale` a positive numeric specifying the prior error variance of the endogenous variables in the inverse Wishart prior.

`shape` for `"gamma"` and `"gamma+covar"` a non-negative numeric, or a character expression in `k` as for `df`, specifying the shape of the gamma prior on the error precisions, the inverse error variances, whose mean is `shape / rate`, either one value or one per endogenous variable. For models with stochastic volatility a numeric specifying the shape of the gamma prior on the precision of the state equation of the log-volatilities.

`rate` a positive numeric specifying the rate that corresponds to `shape`, for `"gamma"` and `"gamma+covar"` either one value or one per endogenous variable.

`mu` numeric of the prior mean of the initial state of the log-volatilities. Only used for models with time varying volatility.

`v_i` numeric of the prior precision of the initial state of the log-volatilities. Only used for models with time varying volatility.

`state_variance` numeric of the initial draw for the variance of the log-volatilities. Only used for models with time varying volatility.

`offset` numeric of the constant, which is added before taking the log of the squared errors. Only used for models with time varying volatility.

For structural models only a gamma prior or stochastic volatility specification is allowed.

Argument `varsel` can contain the following elements:

`inprior` a numeric between 0 and 1 specifying the prior probability of a variable to be included in the model.

`covar` logical indicating if the variable selection algorithm should also be applied to the error covariance matrix.

`exclude_det` logical indicating if deterministic terms should be excluded from the variable selection algorithm.

`minnesota` a numeric vector of length 4 containing parameters for the calculation of the Minnesota-like inclusion priors. See below.

`tau` a numeric vector of two elements containing the prior standard errors of restricted variables (τ_0) as its first element and unrestricted variables (τ_1) as its second. Only used for SSVS.

`semiautomatic` an numeric vector of two elements containing the factors by which the standard errors associated with an unconstrained least squares estimate of the model are multiplied to obtain the prior standard errors of restricted (τ_0) and unrestricted (τ_1) variables, respectively. This is the semiautomatic approach described in George et al. (2008). Only used for SSVS.

In the case of SSVS, either `tau` or `semiautomatic` must be specified.

If `varsel$minnesota` is specified, prior inclusion probabilities are calculated in a Minnesota-like fashion as

$\frac{\kappa_1}{l}$	for own lags of endogenous variables,
$\frac{\kappa_2}{l}$	for other endogenous variables,
$\frac{\kappa_3}{1+l}$	for exogenous variables,
κ_2	for contemporaneous endogenous variables of a structural model,
κ_4	for deterministic variables,

for lag l with κ_1, κ_2, κ_3, κ_4 as the first, second, third and forth element in `varsel$minnesota`, respectively.

Value

The object in `object` with the element `priors` added, a list with

`beta` the prior of the cointegration space with `type "cointspace"`: `v_inv` and `p_tau_inv` for constant cointegration parameters, with `g_inv` if `coint$g_i` was given, or `rho`, `mu` and `v_inv` of the state equation for time varying ones, together with the elements added by `p_tau_i = "ml"` or a uniform prior on ρ.

`a` the prior of the loadings and the remaining coefficients, with the same elements as for a VAR model in `add_priors.bvarmodel`.

`psi`, `u_sigma` the priors of the error covariance coefficients and error variances, as for a VAR model.

Prior on the cointegration space

Argument `coint` can contain the following elements. Which of them are required depends on whether the cointegration vectors are constant or time varying, since the two are given different kinds of prior: a matric-variate prior on the cointegration space in the first case, and a state equation in the second. Any other element raises an error.

`v_i` non-negative numeric specifying the shrinkage of the cointegration space prior, or `"ml"`. See below. Required for models with constant cointegration parameters and not used otherwise.

`p_tau_i` the inverse of the matrix P_τ, which determines the central location of the cointegration space $sp(\beta)$. Either a numeric of its diagonal elements, a full symmetric matrix, or `"ml"`, with one row and column per series in the error correction term `data$train$w`. See below. Required for models with constant cointegration parameters. For models with time varying cointegration parameters only `"ml"` is used.

`weight` positive numeric specifying the weight of a prior centred on the maximum likelihood estimate in units of the information of the sample. Default is 1. Only used if `p_tau_i = "ml"`.

`rho` a numeric specifying the autocorrelation coefficient of the state equation of β. It must be smaller than 1. Required for models with time varying cointegration parameters and not used otherwise. If `rho_min` and `rho_max` are given as well, this is the value the chain starts ρ at rather than the value it keeps, and it must lie between them.

`rho_min`, `rho_max` numerics specifying the support of a uniform prior on ρ, which makes it a drawn parameter rather than a fixed hyperparameter. Optional, and either both or neither; they must satisfy $0 <$`rho_min`$<$`rho_max`≤ 1. Koop et al. (2011) use $(0.999, 1)$. Only used for models with time varying cointegration parameters.

`g_i` the inverse of the matrix G that scales the prior of the loadings, for models with constant cointegration parameters and stochastic volatility, `error = "sv"` or `"sv+covar"`, and refused for every other model. Either a numeric of its diagonal elements, a full symmetric positive definite matrix with one row and column per endogenous variable, or `"ml"` for the inverse of the maximum likelihood estimate of the error covariance. Optional. See below.

For a model with constant cointegration parameters the prior is that of Koop et al. (2010). The sampler uses `v_i` and `p_tau_i` only through their product, so with `v_i = 0` the prior on the cointegration space is uniform whatever `p_tau_i` is. An informative prior on the space therefore needs a positive `v_i`, which also shrinks the loadings: for β close to the centre of the space they have prior $N(0, \Sigma/v)$.

In Koop et al. (2010) the loadings' prior is scaled by a matrix G, which may be the error covariance Σ or any fixed, known matrix. The models with a constant error covariance take $G = \Sigma$. With stochastic volatility the covariance differs from period to period, so G is fixed for the whole run instead: G^{-1} is `g_i` if it is given, and otherwise the error precision implied by the starting values of the log-volatilities, averaged over the sample once before the first draw. That fallback depends on `add_initial_values`, so giving `g_i` makes the prior independent of how the chain is started. `g_i = "ml"` uses Johansen's (1995) estimate of the error covariance, the same one `v_i = "ml"` is based on.

With `p_tau_i = "ml"` the prior is centred on the space spanned by Johansen's (1995) maximum likelihood estimate $\hat{\beta}$, computed from the error correction term as it is stored in the model – so call `scale_error_correction` first if the series should be scaled; scaling is refused afterwards. Let H be an orthonormal basis of that space, $H_\perp$ one of its orthogonal complement and $\beta = H + H_\perp\delta$. The matrix

$$P_\tau^{-1} = HH' + H_\perp T^{-1} H_\perp', \quad T = \frac{v}{w}(H_\perp' S_{11} H_\perp)^{-1},$$

where S_{11} is the cross product of the residuals of a regression of the error correction term on the short-run regressors and w is `weight`, makes the prior of δ given α $N(0, (\alpha'\Sigma^{-1}\alpha)^{-1} \otimes w^{-1}(H_\perp' S_{11} H_\perp)^{-1})$. This is the asymptotic distribution of the maximum likelihood estimator with its precision multiplied by w: `weight = 1` adds as much information about the space as the sample itself holds. Since the centre is estimated from the same sample, the posterior then overstates the precision of $sp(\beta)$. Eigenvalues of T are capped at one, which is the uniform prior.

With `v_i = "ml"` the shrinkage is chosen so that the prior mean of $tr(\alpha'\Sigma^{-1}\alpha)$ is 100 times its maximum likelihood estimate. It can be combined with a numeric `p_tau_i`.

For a model with time varying cointegration parameters the state equation is $\beta_t = \rho\beta_{t-1} + \eta_t$ with $\eta_t \sim N(0, I)$, and the prior on the state before the sample is that equation's own stationary distribution, $N(0, I/(1-\rho^2))$. This is what makes the prior proper, so a ρ close to one is intended and one further from it draws a warning. `add_priors` gives the loadings the compensating scale: only the product $\alpha\beta'$ is identified, so their prior variance is shrunk by $1-\rho^2$, leaving the product on the scale `coef$v_i` asks for.

When ρ is drawn, those last two are computed from `coint$rho` once and do not follow the draw: the state before the sample keeps the normal prior built here, and the loadings keep the shrinkage of `add_priors`. This is a deliberate departure from Koop et al. (2011), in whom the state before the sample is the stationary distribution of whatever ρ currently is. It is also what makes the draw an exact Gibbs block rather than their Metropolis-within-Gibbs step: with the state before the sample free of ρ, the conditional posterior of ρ is a normal truncated to the prior's support. Draws of ρ are returned in `object$posterior$beta$rho`.

How far the space moves is not set by the prior on any variance. The steps η_t have unit variance whatever `coef$rate` and `coef$rate_alpha` are, and how far a step moves $\Pi_t = \alpha_t\beta_t'$ depends on the scale of β_t. Only the product is identified, so the posterior can pair small loadings with large cointegration vectors, which a unit step barely turns, or large loadings with small ones, which it turns a lot. Two things decide which. One is the prior precision of the loadings, `coef$v_i` $/(1-\rho^2)$, which restrains the drift only if it is tight relative to the scale of Π the data call for, and that scale depends on the units of the data. The other is the series in the error correction term: a step moves $\beta_t' w_t$ by $\eta_t' w_t$, which grows with the levels of the series and not only with their variation. For log levels far from zero the shift is of the order of the levels themselves, and the sampler either lets it act as a random walk intercept, which no prior on the deterministic terms controls and which can absorb the residuals of an equation, or draws loadings close to zero, which switches the error correction term off.

Both have been seen. In the US sub-model of the **bgvars** data set `gvar2023`, with the oil price among its endogenous variables and a rank of one, a prior that pinned the state variances of all coefficients, the loadings and the constant included, at 1e-14 still left the residual standard deviation of one equation as low as 0.1 to 0.5 of the maximum likelihood one in some chains, with which equation and how low depending on the chain, and a saved chain reported a posterior mean log-likelihood several hundred above the maximum of the constant coefficient model. The chains do not move between these outcomes within a few thousand draws, so a single chain can report any

of them. The Austrian sub-model under the same prior lost fit that the constant coefficient model has, and the chains that were inspected drew loadings close to zero. On data simulated from a constant coefficient VEC model, whose posterior under such a prior should reproduce the maximum likelihood fit, single chains did so only with the series centred and `coef$v_i = 10`, or around zero and `coef$v_i = 100`. Around 12, where log levels usually are, a chain drew loadings close to zero, and around zero with `coef$v_i = 1` one fitted part of the residuals.

The same mechanism makes `scale_error_correction` risky in a model whose coefficients vary over time. It divides the series by the standard deviation of their differences, which makes them, and with them both a drift in a loading and a step of $\beta_t' w_t$, larger by the same factor. The coefficient paths can then absorb the residuals of an equation almost entirely: its residual variance – or, under stochastic volatility, its volatility path – is reported far below that of a least squares fit of the same regressors and nearly constant over the sample. In the model of the vignette on time varying parameters and stochastic volatility in error correction models, `vignette("tvp-sv-vec", package = "bvartools")`, a scaled error correction term did that to one equation in each of four chains with different seeds, and the unscaled one with a small `coef$rate` in none.

It is safer to leave the series unscaled, to set `coef$v_i` with the scale of Π in mind as well as choosing small rates, and to compare the residual variances with those of a least squares fit of the same regressors, in chains with different seeds, for a model that is to be used. Small rates alone do not guarantee that a model passes that check, and passing it is what says that neither the coefficients nor the cointegration vectors fit the residuals. `scale_error_correction(object, scale = FALSE, centre = TRUE)` centres the series before posterior simulation, which removes the part of a step of $\beta_t' w_t$ that comes from the levels of the series rather than from their variation, and `rescale_error_correction` writes the draws back in terms of the series as they are, so that `vec_to_var` and the forecasts can use them. It does not reach the other channel, the scale of the loadings, and it is not a full remedy: in the US sub-model above, centring alone roughly halved how far the error correction term and the constant moved over the sample and still left the oil price equation at about 0.8 of the maximum likelihood residual standard deviation in two chains. Whether a centred model is right is again what the comparison with least squares shows.

For a model with time varying cointegration parameters `p_tau_i = "ml"` centres the marginal prior of the cointegration space on Johansen's estimate as well, using the informative marginal prior of Koop et al. (2011, working paper version). The state equation becomes $\beta_t = \rho(I_r \otimes P_\tau)\beta_{t-1} + \eta_t$ with $P_\tau = HH' + H_\perp T H_\perp'$: the part of β_t along $sp(H)$ keeps ρ, the part off it decays faster, and the mode of the marginal distribution of $sp(\beta_t)$ is $sp(H)$ in every period. T is chosen so that the prior spread of the tilt of β_t away from $sp(H)$ in a period, approximately $T^* = (1-\rho^2)(I-\rho^2 T^2)^{-1}$, is the sampling variance of Johansen's estimator with its precision multiplied by `weight`. The state before the sample is given the stationary distribution the transition implies. The transition is stored as element `p_tau`.

ρ limits how informative this prior can be: even $T = 0$ leaves the tilt a spread of $1 - \rho^2$ per period, and a `weight` asking for more is floored there with a warning.

A larger `weight` does not always give a tighter posterior. It narrows the prior spread of the tilt in each period by lowering T, but T is also how much of the tilt carries over from one period to the next. As T approaches zero the tilt of each period becomes independent of the last, the data of neighbouring periods stop informing it, and the posterior spread of $sp(\beta_t)$ can widen again. On data set E6 with $\rho = 0.999$, for example, `weight = 1`, which gives $T \approx 0.44$, produced a tighter posterior than `weight = 100`, which reaches $T = 0$. The useful range of `weight` is the one that keeps T clearly above zero, and a ρ closer to one widens it.

A `weight` so small that T would be the identity in every direction leaves the noninformative prior

unchanged. Both P_τ and the prior on the state before the sample are computed at `coint$rho`, and do not follow the draw if ρ is drawn. `coint$v_i` has no counterpart for these models.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Johansen, S. (1995). *Likelihood-based inference in cointegrated vector autoregressive models*. Oxford: Oxford University Press.

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224–242. doi:10.1080/07474930903382208

Koop, G., León-González, R., & Strachan R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

See Also

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvarmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bv`
`use_expanding_window.bvecmodel()`

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 1, r = 1, const = "restricted",
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# The same model with time varying parameters. The cointegration vectors are
# then a state path rather than a draw from a cointegration space prior, so
# 'coint' takes the autocorrelation of that path instead of its shrinkage and
# central location, and 'coef' takes the prior of the state error variances.
model <- create_bvecmodel(e6, p = 4, r = 1, tvp = TRUE,
```

```
                          const = "unrestricted",
                          seasonal = "unrestricted",
                          iterations = 10, burnin = 10)

model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10,
                                shape = 3, rate = 0.0001),
                    coint = list(rho = 0.999),
                    sigma = list(df = "k", scale = 1))
```

`add_priors.expandingwindow` *Add Priors to Bayesian Models*

Description

Adds prior specifications to a list of models by passing each element to the respective method.

Usage

```
## S3 method for class 'expandingwindow'
add_priors(object, ...)
```

Arguments

`object`	a list of class 'expandingwindow', usually, the output of a call to `use_expanding_window`.
`...`	further arguments passed to or from other methods.

Value

The object in `object` with priors added to each of its models, as described in `add_priors.bvarmodel`.

Examples

```
data("us_macrodata")

# Starting period of the forecasting exercise
start_period <- 2007

# AR(1) models as benchmark
model <- create_bvarmodel(data = us_macrodata, p = 1,
                          deterministic = "none",
                          error = "gamma",
                          iterations = 10, burnin = 10)

# Obtain objects for expanding window estimation
model <- use_expanding_window(model, start = start_period)
```

```
# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(shape = 3, rate = 1))
```

`add_priors.modellist` *Add Priors to Bayesian Models*

Description

Adds prior specifications to a list of models by passing each element to the respective method.

Usage

```
## S3 method for class 'modellist'
add_priors(object, ...)
```

Arguments

`object`	a list of class 'modellist', usually, the output of a call to `create_bvarmodel` or `create_bvecmodel`.
`...`	further arguments passed to or from other methods.

Value

The object in `object` with priors added to each of its models, as described in `add_priors.bvarmodel`.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 0:2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
```

add_seed *Seed of the Posterior Simulation*

Description

Sets the seed with which the posterior of a model is simulated.

Usage

```
add_seed(object, seed, ...)

## S3 method for class 'bvarmodel'
add_seed(object, seed, ...)

## S3 method for class 'bvecmodel'
add_seed(object, seed, ...)

## S3 method for class 'modellist'
add_seed(object, seed, ...)

## S3 method for class 'expandingwindow'
add_seed(object, seed, ...)
```

Arguments

object	a model, at any point before the posterior is simulated: an object of class 'bvarmodel' or 'bvecmodel', a model of another package that provides an add_seed method for it, or a list of such models of class 'modellist' or 'expandingwindow'.
seed	a non-negative whole number no larger than .Machine$integer.max.
...	further arguments passed to or from other methods.

Details

Calling this function is optional. add_initial_values already stores a seed, drawn from R's random number generator, in element seed of object$model, unless the model has one. add_seed sets that element, for example to give a model a seed that depends neither on the state of R's generator nor on the worker of a cluster that happens to simulate it. It can be called before add_initial_values, which then keeps the seed it is given, or after it, which replaces the drawn one.

The seed is part of the model. write_to_hdf5 writes it as attribute seed of group /model, where the BayesTS executable reads it, and read_model_from_hdf5 reads it back. The internal samplers of add_posterior_coefficients draw with it too, so a model with a given seed gives the same draws however R's generator stands. BayesTS and this package use different generators, so the same seed gives different draws in the two.

A list of models gets the seeds `seed`, `seed + 1`, ..., one per model in the order of its elements, counting through nested lists, so that no two of its models draw the same numbers. A model is any element with an `add_seed` method of its own, so the models of other packages that provide one, such as the dynamic factor models of **dfmtools**, are counted with the rest. Elements that are not estimated, such as external forecasts, and elements without a method are left as they are and are not counted.

Value

`object` with the seed set.

See Also

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecast`, `add_posterior_forecasts.bvecmodel()`, `add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model
model <- create_bvarmodel(data = e1, p = 2, iterations = 100, burnin = 50)

# Set the seed of the posterior simulation
model <- add_seed(model, 20260916)
model[["model"]][["seed"]]
```

`add_sign_restrictions` *Sign Restrictions*

Description

A generic function used to identify the shocks of a model by the signs of the impulse responses they produce.

Usage

```
add_sign_restrictions(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_sign_restrictions.bvarmodel`, `add_sign_restrictions.expandingwindow`, `add_sign_restrictions`

`add_sign_restrictions.bvarmodel` *Sign Restrictions*

Description

Identifies the shocks of an object of class 'bvarmodel' by the signs of the impulse responses they produce.

Usage

```
## S3 method for class 'bvarmodel'
add_sign_restrictions(
  object,
  restrictions,
  max_tries = 1000,
  period = NULL,
  ...
)

## S3 method for class 'expandingwindow'
add_sign_restrictions(object, ...)

## S3 method for class 'modellist'
add_sign_restrictions(object, ...)
```

Arguments

`object`	an object of class 'bvarmodel', containing posterior draws of the coefficients and the error covariance.
`restrictions`	a data frame of the restrictions, with one row per restriction. See 'Details'.
`max_tries`	the number of rotations that are tried per posterior draw before the draw is given up on. Defaults to 1000.
`period`	integer. Index of the period, whose draws should be identified. Only used for TVP or SV models. Default is `NULL`, so that the posterior draws of the last time period are used.
`...`	further arguments passed to or from other methods.

Details

The function identifies the structural shocks of the VAR model

$$y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t,$$

with $u_t \sim N(0, \Sigma)$, by searching for rotations of its Choleski factor whose impulse responses carry the signs that argument `restrictions` asks for.

Write P for the lower triangular Choleski factor of Σ and Q for an orthogonal matrix. Then $PQ(PQ)' = \Sigma$ for every such Q, so a model rotated by Q fits the data exactly as well as the one the posterior draw describes. The likelihood is therefore silent about which rotation is the right one, and the restrictions choose among the models it cannot tell apart. The identification is in consequence *set valued*: what is obtained is not one impulse response per posterior draw but the collection of those that the restrictions admit.

For every posterior draw the function draws rotations uniformly over the orthogonal group and keeps the first one whose responses satisfy every restriction. Since a sign restriction cannot distinguish a shock from its own negative, a column that fails is retried with its sign flipped before the rotation is discarded. A draw for which no admissible rotation is found within `max_tries` attempts is dropped from the identified sample, and `summary` reports how many were. A low acceptance rate is a statement about the restrictions, not a technicality: it says the model rarely produces the pattern that was asked of it.

Argument `restrictions` is a data frame with the columns

`impulse` name of the endogenous variable the shock is named after.

`response` name of the endogenous variable whose response is restricted.

`sign` either `1` for a response that must be positive or `-1` for one that must be negative.

`horizon` optional integer of the horizon the restriction applies to, counted from zero for the impact period. Defaults to `0`.

Restrictions on several shocks may be combined in one data frame. A shock that no row mentions is left as it is drawn, since nothing in the restrictions distinguishes one rotation of it from another; only the restricted shocks should be interpreted.

Only sign restrictions are supported. A zero restriction cannot be imposed by rejection, because the set of rotations that satisfies one has probability zero, so no number of tries finds a member of it. Use `add_sign_zero_restrictions`, which draws rotations that satisfy the zero restrictions by construction and reweights them.

The accepted rotations are added to the object as element `q` of its posterior draws, from where `irf`, `fevd` and `spillover` use them under `type = "sign"`.

The result depends on the state of the random number generator, so `set.seed` is needed to reproduce it.

Value

The object of class 'bvarmodel' with the accepted rotations in element `q` of its `posterior`, one row per posterior draw, and the specification of the restrictions in element `sign_restrictions` of its `model`. Draws for which no admissible rotation was found are recorded as `NA`.

References

Arias, J. E., Rubio-Ramirez, J. F., Waggoner, D. F. (2018). Inference based on structural vector autoregressions identified with sign and zero restrictions: Theory and applications. *Econometrica, 86*(2), 685-720. doi:10.3982/ECTA14468

Rubio-Ramirez, J. F., Waggoner, D. F., Zha, T. (2010). Structural vector autoregressions: Theory of identification and algorithms for inference. *The Review of Economic Studies, 77*(2), 665-696. doi:10.1111/j.1467937X.2009.00578.x

Uhlig, H. (2005). What are the effects of monetary policy on output? Results from an agnostic identification procedure. *Journal of Monetary Economics, 52*(2), 381-419. doi:10.1016/j.jmoneco.2004.05.007

See Also

Other post-estimation analysis: `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# A demand shock raises investment and consumption on impact
restrictions <- data.frame(impulse = "invest",
                           response = c("invest", "cons"),
                           sign = c(1, 1),
                           horizon = 0)

set.seed(1234)
model <- add_sign_restrictions(model, restrictions)

# Obtain the identified impulse response
```

```
ir <- irf(model, impulse = "invest", response = "cons", type = "sign")
```

`add_sign_zero_restrictions` *Sign and Zero Restrictions*

Description

A generic function used to identify the shocks of a model by the signs of the impulse responses they produce and by responses that are restricted to be zero.

Usage

```
add_sign_zero_restrictions(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `add_sign_zero_restrictions.bvarmodel`, `add_sign_zero_restrictions.expandingwindow`, `add_sign_zero_restrictions.modellist`.

`add_sign_zero_restrictions.bvarmodel` *Sign and Zero Restrictions*

Description

Identifies the shocks of an object of class 'bvarmodel' by the signs of the impulse responses they produce together with responses that are restricted to be exactly zero.

Usage

```
## S3 method for class 'bvarmodel'
add_sign_zero_restrictions(
  object,
  restrictions,
  draws = NULL,
  one_sided = FALSE,
  smooth = TRUE,
  period = NULL,
  max_tries = 1,
  ...
)

## S3 method for class 'expandingwindow'
add_sign_zero_restrictions(object, ...)

## S3 method for class 'modellist'
add_sign_zero_restrictions(object, ...)
```

Arguments

`object`	an object of class 'bvarmodel', containing posterior draws of the coefficients and the error covariance.
`restrictions`	a data frame of the restrictions, with one row per restriction. See 'Details'.
`draws`	integer. The number of draws to resample. Defaults to `NULL`, which uses the effective sample size of the importance sampler, the number of independent draws it actually produced. Asking for more than that is allowed and duplicates draws.
`one_sided`	logical. Should the numerical derivative behind the importance weights be taken on one side only? It is about forty percent faster and correspondingly less accurate. Defaults to `FALSE`.
`smooth`	logical. Should the importance weights be Pareto smoothed? Defaults to `TRUE`. `FALSE` uses the raw weights of the paper's Algorithm 3, which is what to set to reproduce it exactly.
`period`	integer. Index of the period, whose draws should be identified. Only used for TVP or SV models. Default is `NULL`, so that the posterior draws of the last time period are used.
`max_tries`	integer. The largest number of rotations drawn for each posterior draw. Defaults to 1, the algorithm of the paper. More tries find rotations where the sign restrictions leave a region too small for a single try to hit, and the importance weights account for them. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

A sign restriction can be imposed by trying rotations until one of them carries the signs that were asked for, which is what `add_sign_restrictions` does. A zero restriction cannot: the rotations

that satisfy one form a set of probability zero, so no number of tries finds a single member of it. This function uses the algorithms of Arias, Rubio-Ramirez and Waggoner (2018) instead, and needs at least one zero restriction to be worth using – with only signs, `add_sign_restrictions` produces the same draws more cheaply, and this function says so rather than running.

The restrictions are imposed on $F(A_0, A_+)$, the responses stacked over the horizons the restrictions mention. That function satisfies $F(A_0Q, A_+Q) = F(A_0, A_+)Q$ for every orthogonal Q, so a zero restriction on the response to the jth shock is a *linear* restriction on the jth column of Q once the reduced form is fixed. The columns are therefore built one at a time, each drawn uniformly from the sphere in the subspace that the zero restrictions and the columns already placed leave over. Every rotation the function draws satisfies the zero restrictions exactly rather than approximately.

That construction does not draw from the posterior conditional on the zero restrictions but from a distribution that differs from it by a volume element, which the function computes numerically and divides out. The result is an importance sample: each draw carries a weight, and the draws are resampled with replacement according to those weights so that what the function returns is an ordinary, equally weighted sample. A draw whose rotation fails the sign restrictions is given weight zero and so never resampled, and a shock is not retried with its sign flipped, since the sphere its column is drawn from already covers both of its signs.

By default each posterior draw gets one rotation, as in the paper. With many sign restrictions that can leave nothing: when one rotation in tens of thousands satisfies them all, a few thousand posterior draws produce no admissible rotation, and the function stops. `max_tries` draws more rotations per draw, but not the way `add_sign_restrictions` does. Stopping at the first rotation that satisfies the signs would bias an importance sampler: the single try is what weights each posterior draw by the probability p that its rotations satisfy the signs, and a draw whose admissible set is tiny would count as much as one whose set is large. Instead the rotations are drawn until two of them satisfy the signs, or until `max_tries` have been drawn. The first of them is kept, and its weight is multiplied by an unbiased estimate of p from the tries (Girshick, Mosteller and Savage, 1946): $1/(N-1)$ when the second came at try N, and $1/T$ when only one came in all T tries. The rotation kept does not depend on how many tries it took, so the weight stays an unbiased estimate of the paper's and the sample stays exact. The price is variance: an estimated p spreads the weights, and the effective sample size says by how much. Looking for the second success also means drawing about twice the rotations a first success needs.

With more than one try a column whose sign restrictions all hold with the opposite sign is also flipped rather than rejected. That is exact too: the proposal does not change when a column changes sign, so the rotation kept has the same distribution, and the probability that a try succeeds grows by 2^s, s the number of shocks carrying sign restrictions – the same factor for every draw, which cancels when the weights are normalised. It cuts the tries needed by that factor, 64 for six sign restricted shocks. `max_tries = 1` flips nothing: it is the paper's algorithm and draws exactly what the function drew before the argument existed.

The draws that come back are a resample and no longer a chain. Their order carries no information, several of them may be copies of the same original draw, and convergence diagnostics computed on them mean nothing. How much independent information they carry is the effective sample size, which `summary` reports.

Argument `restrictions` is a data frame with the columns

`impulse` name of the endogenous variable the shock is named after.

`response` name of the endogenous variable whose response is restricted.

`sign` `1` for a response that must be positive, `-1` for one that must be negative, and `0` for one restricted to be exactly zero.

`horizon` optional horizon the restriction applies to, counted from zero for the impact period. `Inf` restricts the long-run response. Defaults to `0`.

The order of the endogenous variables matters here in a way it does not for sign restrictions alone. Columns of Q are built in the order the variables appear, and the jth column has to keep at least one dimension after the $j-1$ columns before it and its own zero restrictions have been taken out. A shock carrying many zero restrictions therefore has to be named early. The function refuses an ordering that leaves a column with nothing to draw and says which shock it was.

The result depends on the state of the random number generator, so `set.seed` is needed to reproduce it. It depends on it in two places rather than one: the rotations are drawn at random, and so are the matrices that complete each shock's constraints to a square system. Appendix A.3 of the paper says any draw of the latter defines a valid algorithm, and that is true – but not that any draw defines an equally efficient one. Holding a model, its posterior draws and every rotation fixed, different completions have moved the effective sample size by a factor of four. A run reporting a poor effective sample size is therefore worth repeating under a different seed before the restrictions themselves are blamed.

The weights are Pareto smoothed before they are used. Nothing in the algorithm bounds the ratio of two volume elements, so a draw landing where the proposal put almost no probability and the target a great deal carries an enormous weight, the effective sample size collapses, and the resample is a few copies of that one draw. Pareto smoothing, of Vehtari et al. (2024), fits a generalised Pareto distribution to the largest few weights and replaces them by its quantiles. On a model where the raw weights gave an effective sample size of 4 out of 487 accepted draws, with one of them holding 46 percent of the total weight, smoothing returned 138 and the largest share fell to 6 percent; on the same model under three other seeds, where nothing was wrong, it changed the effective sample size by less than three percent either way. It buys that at the cost of a small bias, which is the trade the paper argues for. `smooth = FALSE` takes the raw weights of Algorithm 3 instead.

The shape k of the fitted distribution is worth more than the smoothing. It estimates how heavy the tail of the weights is, and so says when the sample cannot be trusted rather than leaving that to be guessed from an effective sample size: the estimator has a finite variance only for k below one half, and above about 0.7 neither it nor its effective sample size means much. It is reported by `summary` and kept in element `pareto_k` of `sign_zero_restrictions`.

The function warns when the importance sample is not fit to summarise: when k is 0.7 or above, when the effective sample size is below twenty, so that the draws returned have no percentiles worth reading, or when it keeps less than a quarter of the information in the draws that satisfied the signs. The warning says which repair applies. A well-behaved tail with a small effective sample size is an efficiency problem that more candidate draws fix, since the effective sample size grows roughly in proportion to them. A heavy tail is not fixed that way, because the trouble is where the proposal put its probability rather than how much of it was drawn; a different seed, a different variable ordering or weaker restrictions are the things worth trying.

Applied to a 'modellist' or an 'expandingwindow' the function identifies each member on its own and returns the collection. Each therefore gets its own effective sample size, and unless `draws` is given the members come back with different numbers of draws – which is what it means for one model, or one window of the sample, to support the restrictions less well than another. Pass `draws` to give them all the same number.

Value

The object of class 'bvarmodel' with its posterior draws resampled, the accepted rotations in element `q` of its `posterior`, one row per resampled draw, and the specification of the restrictions in element `sign_zero_restrictions` of its `model`, together with `max_tries`, the number of rotations drawn in all (`tries`), the number of draws that satisfied the sign restrictions, the effective sample size of the importance sampler and the share of the total weight held by its single largest draw, the shape `pareto_k` of the distribution fitted to the tail of the weights and whether they were `smoothed`. Element `sign_restrictions` is set alongside it, so that `irf`, `fevd` and `spillover` read the rotations under `type = "sign"` as they do for `add_sign_restrictions`.

References

Arias, J. E., Rubio-Ramirez, J. F., Waggoner, D. F. (2018). Inference based on structural vector autoregressions identified with sign and zero restrictions: Theory and applications. *Econometrica, 86*(2), 685-720. doi:10.3982/ECTA14468

Girshick, M. A., Mosteller, F., Savage, L. J. (1946). Unbiased estimates for certain binomial sampling problems with applications. *The Annals of Mathematical Statistics, 17*(1), 13-23.

Rubio-Ramirez, J. F., Waggoner, D. F., Zha, T. (2010). Structural vector autoregressions: Theory of identification and algorithms for inference. *The Review of Economic Studies, 77*(2), 665-696. doi:10.1111/j.1467937X.2009.00578.x

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., Gabry, J. (2024). Pareto smoothed importance sampling. *Journal of Machine Learning Research, 25*(72), 1-58.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)
```

```
# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Consumption does not respond to the investment shock on impact, and income
# rises. Investment is the first variable of the data set, which is what
# leaves its shock a column to draw: see 'Details' on the ordering.
restrictions <- data.frame(impulse = "invest",
                           response = c("cons", "income"),
                           sign = c(0, 1),
                           horizon = 0)

set.seed(1234)
model <- add_sign_zero_restrictions(model, restrictions)

# Obtain the identified impulse response
ir <- irf(model, impulse = "invest", response = "income", type = "sign")
```

`aggregate_forecasts` *Aggregate Forecasts to Annual Figures*

Description

Turns the forecast draws of models estimated on quarterly or monthly data into draws of the annual figures they imply, so that the models can be compared with annual forecasts, such as the projections of international institutions, on the same annual basis.

Usage

```
aggregate_forecasts(object, code, target = "average", levels = NULL, scale = 1)
```

Arguments

`object`	an object of class 'bvarmodel', 'bvecmodel', 'expandingwindow' or 'modellist', whose models contain forecasts, i.e. `add_posterior_forecasts` was already applied to them.
`code`	a named integer vector of the transformation codes of FRED-MD and FRED-QD, 1 : 7, which were applied to the variables of the models – see `transform_variables` and 'Details'. The names are endogenous variables of the models. Endogenous variables, which are not named, are dropped from the comparison.
`target`	either `"average"` (default) or `"q4q4"`, the annual figure that is compared. See 'Details'.
`levels`	an optional time-series object with the untransformed series, to which `code` was applied, i.e. the argument `x` of `transform_variables`. Its columns are named after the variables. Required for codes 2, 3, 6 and 7. See 'Details'.
`scale`	the factor, by which the transformed series of codes 4 to 7 were multiplied before they were passed to the models, such as 100 for log differences in percent. The default of 1 corresponds to the result of `transform_variables`.

Details

Every draw of a forecast is turned back into a path of the untransformed series by reversing the transformation of its code, and continued from the periods, which were observed at the end of the training sample. The periods of a year, which were already observed, are thus taken from the data and the remaining periods from the draw, so that every draw of the forecast becomes a draw of the annual figure.

The codes determine the annual figure. The multiplicative codes 4 to 7 – logarithms, their differences and differences of growth rates – describe a series such as GDP or a price index, whose annual figure is a growth rate in percent. The additive codes 1 to 3 describe a series such as an unemployment rate or an interest rate, whose annual figure is a level. Argument `target` determines, which growth rate and which level:

`"average"` the growth of the annual average of the levels over the annual average of the year before for codes 4 to 7, and the annual average for codes 1 to 3. This is the convention of, e.g., the World Economic Outlook of the IMF. The growth of the annual average is also the growth of the annual sum, since both years have the same number of periods, so it serves flows such as GDP and averages such as prices alike.

`"q4q4"` the growth of the level of the last period of the year over the last period of the year before, i.e. the fourth quarter over the fourth quarter or December over December, for codes 4 to 7, and the level of the last period of the year for codes 1 to 3. This is the convention of, e.g., the Summary of Economic Projections of the Federal Reserve.

Reversing a difference requires the level, from which it starts. For codes 2, 3, 6 and 7 it is taken from argument `levels`, which is therefore required for them. Codes 1, 4 and 5 can be reversed from the data of the models, because the unknown level of a logarithm cancels from a growth rate. If `levels` is provided, it must reproduce the data of the models under `code` and `scale`, which guards against a wrong `scale`, and it also provides the realised annual figures.

Horizon 1 is the year of the forecast origin, i.e. the year of the first period after the training sample, and horizon 2 the year after it. The number of annual horizons is the number of years that the forecast horizon covers from every origin, `floor(n_ahead / frequency)`, so that a quarterly model, which forecasts eight quarters, provides the current and the next year.

The realised annual figures, against which the aggregated forecasts are scored, are obtained from `levels` or, without it, from the data of the models in the same way, and are put in `data$test$y`. `add_forecast_errors` uses them, if its argument `test_sample` is omitted. Otherwise, `test_sample` must be an annual time series, for example of official annual figures.

The result can be passed as argument `object` to `create_external_forecast`, which then reads the periods of the external forecasts as years, but matches their publications to the training samples of the models at the frequency of the data. The external forecasts are scored against the same realised annual figures as the models.

The aggregated models only contain the forecasts and the data, which are required to evaluate them. Therefore, aggregation is the last step before `add_forecast_errors` and `selection_criteria`, which only provides out-of-sample statistics for them. Aggregated VEC models become objects of class 'bvarmodel', because their forecasts are those of the levels.

Value

An object of the same class as `object`, whose models contain the draws of the annual forecasts in `posterior$forecast$forecasts`.

References

McCracken, M. W., & Ng, S. (2021). FRED-QD: A quarterly database for macroeconomic research. *Federal Reserve Bank of St. Louis Review, 103*(1), 1–44.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcritlist(), create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("us_macrodata")

# Inflation and the interest rate enter the model as they are
model <- create_bvarmodel(data = us_macrodata, p = 1, deterministic = "const",
                          iterations = 10, burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.

model <- use_expanding_window(model, start = 2005)
model <- add_priors(model, coef = list(v_i = 0.1, v_i_det = 0.01),
                    sigma = list(df = "k", scale = 1))
model <- add_initial_values(model)
model <- add_posterior_coefficients(model)
model <- add_forecast_input(model, n_ahead = 8)
model <- add_posterior_forecasts(model)

# Annual averages of inflation and the interest rate
annual <- aggregate_forecasts(model, code = c(Dp = 1, r = 1))

# Artificial annual forecasts of the current and the next year, published in
# the middle of each quarter
fcst <- expand.grid(origin = 2005 + (0:7) / 4 + 0.1, year = 0:1,
                    variable = c("Dp", "r"), stringsAsFactors = FALSE)
fcst[["period"]] <- floor(fcst[["origin"]]) + fcst[["year"]]
fcst[["value"]] <- 2

ext <- create_external_forecast(fcst, annual, data_lag = 1)

# Both are scored against the annual averages of the data
models <- add_forecast_errors(combine_models(annual, ext))
selection_criteria(models)
```

`aggregation_weights` *Weights of a Temporal Aggregate*

Description

The weights with which a series observed at a lower frequency aggregates the high-frequency periods of a model, for argument `aggregate` of `create_bvarmodel`.

Usage

```
aggregation_weights(type = c("average", "sum", "growth"), n = 3)
```

Arguments

`type`	a character naming the aggregation: `"average"` for a series that is the mean of the periods it covers, such as a quarterly unemployment rate in a monthly model, `"sum"` for a flow that is their total, and `"growth"` for the growth rate of an average, following Mariano and Murasawa (2003). See 'Details'.
`n`	an integer, the number of high-frequency periods one low-frequency period covers: 3 for quarters of months, 12 for years of months, 4 for years of quarters.

Details

An observation of the low-frequency series in period t is

$$x_t = \sum_{j=1}^{L} w_j y_{t-L+j},$$

where y_t is the high-frequency series the model contains and $w_1, \ldots, w_L$ are the weights returned, oldest first.

For `"average"` they are n weights of $1/n$, and for `"sum"` n weights of one. For `"growth"` the high-frequency series is the growth rate – the log difference – of a series whose average the low-frequency series is the growth rate of. Approximating the log of an average by the average of the logs, that growth rate is a triangular combination of $2n - 1$ high-frequency growth rates, which for quarters of months is $(1, 2, 3, 2, 1)/3$.

Value

A numeric vector of weights, oldest period first.

References

Mariano, R. S., & Murasawa, Y. (2003). A new coincident index of business cycles based on monthly and quarterly series. *Journal of Applied Econometrics, 18*(4), 427–443. doi:10.1002/jae.695

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvarmodel(), add_initial_values.bvecmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), combine_models(), create_bvarmodel(), create_bvecmodel(), transform_variables(), use_expanding_window.bv use_expanding_window.bvecmodel()

Examples

```
aggregation_weights("growth", 3)
```

`align_model_obs` *Align Observations Across Models*

Description

Generic function used to restrict each model in a list to the set of observations common to all models, ensuring that comparisons are computed on the same underlying sample.

Usage

```
align_model_obs(object, ...)
```

Arguments

`object`	a list of models.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: align_model_obs.modellist.

`align_model_obs.modellist` *Align Observations Across Models*

Description

Restricts each model in an object of class 'modellist' to the set of observations common to all models, ensuring that comparisons are computed on the same underlying sample.

Usage

```
## S3 method for class 'modellist'
align_model_obs(object, ...)
```

Arguments

`object`	an object of class 'modellist'.
`...`	further arguments passed to or from other methods.

Value

An object of class 'modellist'.

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `analysis_of_stored_models`, `choose_best_model.selcritlist()`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `map_models()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data(e6)

# Create multiple VAR models
model_1 <- create_bvarmodel(diff(e6), p = 0:2, deterministic = "const")

# Create multiple VEC models
model_2 <- create_bvecmodel(e6, p = 1, r = 0:1, const = "unrestricted")

# Combine the models in one object
model <- combine_models(model_1, model_2)

model <- align_model_obs(model)
```

analysis_of_stored_models

Impulse Responses and Variance Decompositions of a Stored Model

Description

Impulse response functions and forecast error variance decompositions of a model in an HDF5 file, computed a piece of the chain at a time.

Usage

```
## S3 method for class 'bvarfile'
irf(x, ..., chunk = 100)

## S3 method for class 'bvarfile'
fevd(x, ..., chunk = 100)
```

Arguments

x	an object of class 'bvarfile', from open_model.
...	arguments of irf.bvarmodel or fevd.bvarmodel, which do the work on each piece.
chunk	how many draws are read at a time. Defaults to 100.

Details

Both quantities are sums over the draws: an impulse response is a set of quantiles of the responses of the draws, a variance decomposition their mean. Neither needs the draws together, so the chain is read in pieces of chunk, each piece contributes what it has, and the pieces are put together at the end. What comes out is what the same call on the model in memory gives.

For a variance decomposition the pieces are combined before the shares are normalised and before groups are collapsed, so normalise_gir and max_groups describe the whole chain rather than the piece they were applied to.

Value

What irf.bvarmodel and fevd.bvarmodel return: an object of class 'bvarirf' or 'bvarfevd'.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), choose_best_model.selcritlist(), create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = 1, scale = .0001))
model <- add_posterior_coefficients(add_initial_values(model))

file <- file.path(tempdir(), "bvartools-example-irf.h5")
unlink(file)
write_to_hdf5(model, filename = file)
stored <- open_model(file)

ir <- irf(stored, impulse = "invest", response = "income", n_ahead = 5,
          chunk = 25)
shares <- fevd(stored, response = "income", n_ahead = 5, chunk = 25)
```

---

arias_rubio_ramirez_waggoner_2018

Sign and Zero Restricted Rotations of Posterior Draws

Description

The identification step of `add_sign_zero_restrictions` on its own: rotations of the reduced-form draws of a VAR that satisfy sign and zero restrictions on the impulse responses, with the importance weights of Arias, Rubio-Ramirez and Waggoner (2018). It is exported for packages that estimate a VAR as part of something larger – the state equation of a factor augmented VAR, for example – and want to identify it the same way.

Usage

```
arias_rubio_ramirez_waggoner_2018(
  draws,
  restrictions,
  variables,
  lags,
  max_tries = 1,
  smooth = TRUE,
  one_sided = FALSE
)
```

Arguments

`draws`	a list with one element per posterior draw, each a list with elements `A`, the $K \times M$ coefficient matrix with the p lag blocks first and any deterministic terms after them, and `Sigma`, the $K \times K$ error covariance.
`restrictions`	a data frame of the restrictions, as in `add_sign_zero_restrictions`: columns `impulse`, `response`, `sign` and optionally `horizon`, the first two naming elements of `variables`. A table without a zero restriction is accepted here; the rotations are then drawn uniformly and the weights are equal.
`variables`	the names of the K variables, in the order of the rows of `A`. Shocks are named after them and their columns are drawn in this order.
`lags`	the lag order p.
`max_tries`	integer. The largest number of rotations drawn for each posterior draw. See 'Details' of `add_sign_zero_restrictions`.
`smooth`	logical. Should the importance weights be Pareto smoothed?
`one_sided`	logical. Should the numerical derivative behind the importance weights be taken on one side only?

Details

The rotation Q of a draw maps into the impact matrix LQ, with L the lower Cholesky factor of `Sigma`: the response of the variables to the shocks on impact.

The function draws from the random number generator in the order `add_sign_zero_restrictions` does, so the same seed gives the same rotations through either. It warns when the importance sample is not fit to summarise, and stops when no draw has an admissible rotation.

Value

A list with

`q`	a matrix with one row per draw holding its rotation, by column, and `NA` for a draw with no admissible rotation;
`weights`	the normalised importance weights, zero for such a draw;
`restrictions`	the restrictions, with positions in place of names;
`tries`	the number of rotations drawn in all;
`accepted`	the number of draws with an admissible rotation;
`effective_sample_size`, `max_weight_share`, `pareto_k`	the diagnostics of the importance sample, as `add_sign_zero_restrictions` records them.

References

Arias, J. E., Rubio-Ramirez, J. F., Waggoner, D. F. (2018). Inference based on structural vector autoregressions identified with sign and zero restrictions: Theory and applications. *Econometrica, 86*(2), 685-720. doi:10.3982/ECTA14468

See Also

add_sign_zero_restrictions, which applies it to a 'bvarmodel' and resamples the draws.

---

at_macrodata *Austrian sub-model of a global VAR*

Description

The data set contains quarterly time series for Austria, the corresponding foreign (star) variables and global commodity prices from 1979Q2 to 2023Q3. It was produced from data set gvar2023 of package bgvars, which contains the GVAR database of Mohaddes and Raissi (2024), as the Austrian sub-model of a global VAR model of all 33 countries of the database.

Usage

```
data("at_macrodata")
```

Format

A named list with two elements, which are quarterly time-series objects from 1979Q2 to 2023Q3 with 178 rows each:

domestic a time-series object with 6 columns, the domestic (endogenous) variables:

- y log real GDP.
- Dp rate of inflation, the quarterly change in the log CPI.
- eq log real equity prices.
- ep log exchange rate against the US dollar, deflated by the CPI.
- r short-term interest rate, $0.25 \ln(1 + R^S/100)$, where R^S is the annual short-term rate in percent.
- lr long-term interest rate, $0.25 \ln(1 + R^L/100)$, where R^L is the annual long-term rate in percent.

foreign a time-series object with 9 columns, the foreign and global (weakly exogenous) variables:

- y.s, Dp.s, eq.s, ep.s, r.s, lr.s foreign counterparts of the domestic variables.
- poil log oil prices.
- pmat log agricultural raw material prices.
- pmetal log metals prices.

Details

The foreign variables are trade weighted averages of the series of the other countries of the database for which the respective variable is available. The weights are constant over the whole sample and equal the shares of the countries in Austria's trade summed over the years 2014 to 2016.

In a sub-model of a global VAR the domestic variables are endogenous, and the foreign and global variables are weakly exogenous. The two elements of the list can therefore be passed to arguments data and exogen of create_bvarmodel or create_bvecmodel, e.g. create_bvarmodel(data = at_macrodata$domestic, exogen = at_macrodata$foreign).

References

Dees, S., di Mauro, F., Pesaran, M. H., & Smith, L. V. (2007). Exploring the international linkages of the euro area: A global VAR analysis. *Journal of Applied Econometrics, 22*(1), 1–38. doi:10.1002/jae.932

Mohaddes, K., & Raissi, M. (2024). *Compilation, revision and updating of the global VAR (GVAR) database, 1979Q2–2023Q3* (mimeo). University of Cambridge: Judge Business School.

`bayests_files` *Run BayesTS on Stored Models*

Description

Returns a function that runs the standalone BayesTS executable on model files or on directories of them, several at a time, leaving the results in the files.

Usage

```
bayests_files(executable = NULL, library_path = NULL)
```

Arguments

`executable`	the path to the BayesTS executable. If `NULL` (default), option `bvartools.bayests_executable` or, if that is not set, environment variable `BAYESTS_EXECUTABLE`.
`library_path`	a character vector of directories the executable needs on its search path for its runtime libraries, put in front of `PATH` while it runs. A packaged BayesTS carries its libraries and does not need it; an executable from a build tree may.

Details

`bayests_posterior` draws one model that is in the session: it writes the model to a scratch file, runs BayesTS on it and reads the draws back. For models that are already stored – a folder written with `write_to_hdf5`, which is how a model too large to hold is worked on – that is three copies of the draws per model and a session holding them, for no purpose. The function returned here runs BayesTS on the stored files themselves and leaves what it produces where it wrote it, so the draws never pass through R.

The returned function takes paths, which are model files or directories of them, the name of a BayesTS command, and how many to run at once:

`"posterior"` draws the posterior, and with it the pointwise log-likelihood unless `--no-loglik` is passed in `args`.

`"loglik"` the log-likelihood of a model that has draws.

`"forecasts"` the forecasts of a model that has draws.

`"check"` reads and validates the models without running them, which is what to call before an estimation that will take hours.

A model is drawn with the seed in its file, so it does not matter whether it is run on its own, as one of a directory, or through `bayests_posterior`: the draws are the same.

Value

A function of paths, a command, further arguments for the executable and the number of processes to run at once. It returns the paths invisibly and fails with the end of BayesTS's output if any of the runs does.

Running several at once

The executable works through a directory one model at a time and gains nothing from `OMP_NUM_THREADS` on models of the size a country model of a global VAR has, so the way to use a machine is to give it several paths and a `jobs` above one.

They are run as processes of the operating system rather than on a cluster of R workers, because an idle R worker is not cheap: on Windows a fresh one commits about 2 GB before it does anything and about 4 GB once it has loaded a package of this kind, which for six workers is more memory than the six BayesTS processes they would be waiting for. Here one R session starts the processes, each writing its output to a log of its own, and waits for them.

See Also

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecast`
`add_posterior_forecasts.bvecmodel()`, `add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel`
`add_seed()`, `bayests_posterior()`, `bvar()`, `bvec()`, `chain_diagnostics()`, `predict.bvecmodel()`

Examples

```
## Not run:
run <- bayests_files(executable = "/opt/bayests/bin/bayests")

# One directory of models, drawn in place
run("models/gvar/submodels/US")

# Every country, six processes at a time
run(list.dirs("models/gvar/submodels", recursive = FALSE), jobs = 6)

# Validate first, draw without the log-likelihood
run("models/gvar/submodels/US", command = "check")
run("models/gvar/submodels/US", args = "--no-loglik")

## End(Not run)
```

`bayests_posterior` *Posterior Simulation with the BayesTS Executable*

Description

Returns a function that simulates the posterior of a model with the standalone BayesTS executable rather than with the samplers compiled into this package.

Usage

```
bayests_posterior(executable = NULL, library_path = NULL, scratch = tempdir())
```

Arguments

executable	the path to the BayesTS executable. If NULL (default), option bvartools.bayests_executable or, if that is not set, environment variable BAYESTS_EXECUTABLE.
library_path	a character vector of directories the executable needs on its search path for its runtime libraries, put in front of PATH while it runs. A packaged BayesTS carries its libraries and does not need it; an executable from a build tree may.
scratch	the directory where the model files are written while BayesTS draws into them. They are removed afterwards.

Details

The result is meant for argument posterior_function of add_posterior_coefficients:

```
model <- add_posterior_coefficients(model,
  posterior_function = bayests_posterior())
```

It writes the model with write_to_hdf5, runs bayests posterior on the file without its log-likelihood and forecast steps, reads the draws back with read_model_from_hdf5 and adds them to the model as element posterior. Nothing else about the model is changed. The draws have the elements, dimensions and thinning of those of the internal samplers, whose C++ code BayesTS shares; a BayesTS build linked against an optimised BLAS draws faster.

BayesTS draws with the seed in object$model$seed, which add_initial_values sets and add_seed replaces. A model without one is given a seed drawn from R's random number generator, since BayesTS would otherwise start every model from the same state of its own generator. The seed used is kept in the returned model.

The executable runs single threaded unless OMP_NUM_THREADS says otherwise, which suits simulating one model per worker of a cluster.

Value

A function of one argument, a model of class 'bvarmodel' or 'bvecmodel', that returns that model with its posterior draws.

See Also

Other posterior simulation: add_forecast_input.bvarmodel(), add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecas
add_posterior_forecasts.bvecmodel(), add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel
add_seed(), bayests_files(), bvar(), bvec(), chain_diagnostics(), predict.bvecmodel()

bvar *Bayesian Vector Autoregression Objects*

Description

bvar collects the posterior draws of a vector autoregressive model in an object of class 'bvarmodel'.

Usage

```
bvar(
  data = NULL,
  exogen = NULL,
  y,
  x = NULL,
  z = NULL,
  A0 = NULL,
  A = NULL,
  B = NULL,
  C = NULL,
  Sigma = NULL,
  error = NULL,
  varsel = NULL,
  iterations = NULL,
  burnin = 0
)
```

Arguments

data	the original time-series object of endogenous variables. If NULL (default), the object provided in argument y is used.
exogen	the original time-series object of unmodelled variables. If NULL (default), it is reconstructed from the current values of those variables in argument x.
y	a time-series object of endogenous variables with T observations, usually, a result of a call to create_bvarmodel.
x	a time-series object of $(pK + (1 + s)M + N)$ regressor variables, usually, a result of a call to create_bvarmodel.
z	a $TK \times K(pK+(1+s)M+N)$ data matrix, with $K(K-1)/2$ further columns for the contemporaneous endogenous variables of a structural model, usually, a result of a call to create_bvarmodel. If NULL (default), it is generated from argument x.
A0	either a $K^2 \times S$ matrix of MCMC coefficient draws of structural parameters or a named list, where element coeffs contains a $K^2 \times S$ matrix of MCMC coefficient draws of structural parameters and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed. For time varying parameter models the coefficient matrix must

be $TK^2 \times S$. Draws of the error covariance matrix of the state equation can be provided as a $K^2 \times S$ matrix in an additional list element.

`A` either a $pK^2 \times S$ matrix of MCMC coefficient draws of lagged endogenous variables or a named list, where element `coeffs` contains a $pK^2 \times S$ matrix of MCMC coefficient draws of lagged endogenous variables and element `lambda` contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed. For time varying parameter models the coefficient matrix must be $pTK^2 \times S$. Draws of the error covariance matrix of the state equation can be provided as a $pK^2 \times S$ matrix in an additional list element.

`B` either a $((1 + s)MK) \times S$ matrix of MCMC coefficient draws of unmodelled, non-deterministic variables or a named list, where element `coeffs` contains a $((1 + s)MK) \times S$ matrix of MCMC coefficient draws of unmodelled, non-deterministic variables and element `lambda` contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed. For time varying parameter models the coefficient matrix must be $(1+s)TMK \times S$. Draws of the error covariance matrix of the state equation can be provided as a $(1 + s)MK \times S$ matrix in an additional list element.

`C` either a $KN \times S$ matrix of MCMC coefficient draws of deterministic terms or a named list, where element `coeffs` contains a $KN \times S$ matrix of MCMC coefficient draws of deterministic terms and element `lambda` contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed. For time varying parameter models the coefficient matrix must be $TKN \times S$. Draws of the error covariance matrix of the state equation can be provided as a $KN \times S$ matrix in an additional list element.

`Sigma` a $K^2 \times S$ matrix of MCMC draws for the error variance-covariance matrix or a named list, where element `coeffs` contains a $K^2 \times S$ matrix of MCMC draws for the error variance-covariance matrix and element `lambda` contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed to the covariances. For models with stochastic volatility the matrix must be $TK^2 \times S$.

`error` a character specifying the model that was used for the estimation of the covariance matrix of the error term. If `NULL` (default), it is inferred from the draws in argument `Sigma`, which can only distinguish a constant covariance matrix from a time varying one. See `create_bvarmodel`.

`varsel` a character specifying the variable selection algorithm that was employed. If `NULL` (default), it is inferred from the presence of draws of inclusion parameters. See `create_bvarmodel`.

`iterations` an integer of the number of MCMC draws in the provided coefficient matrices. If `NULL` (default), it is obtained from those matrices.

`burnin` an integer of the number of MCMC draws that were used to initialise the sampler. Defaults to zero, since the draws provided to the function are assumed to exclude them already.

Details

For the VARX model

$$A_0 y_t = \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=0}^{s} B_i x_{t-i} + C d_t + u_t$$

the function collects the S draws of a Gibbs sampler (after the burn-in phase) in a standardised object, where y_t is a K-dimensional vector of endogenous variables, A_0 is a $K \times K$ matrix of structural coefficients. A_i is a $K \times K$ coefficient matrix of lagged endogenous variabels. x_t is an M-dimensional vector of unmodelled, non-deterministic variables and B_i its corresponding coefficient matrix. d_t is an N-dimensional vector of deterministic terms and C its corresponding coefficient matrix. u_t is an error term with $u_t \sim N(0, \Sigma_u)$.

For time varying parameter and stochastic volatility models the respective coefficients and error covariance matrix of the above model are assumed to be time varying, respectively.

The draws of the different coefficient matrices provided in `A0`, `A`, `B`, `C` and `Sigma` have to correspond to the same MCMC iterations.

The result is the same kind of object as the output of `create_bvarmodel` in combination with `add_posterior_coefficients`, so it can be used with the methods of class 'bvarmodel' such as `irf.bvarmodel`, `predict.bvarmodel` or `summary.bvarmodel`. Accordingly, the draws are stored in the parameterisation those methods expect: the coefficients of the conditional mean are collected in a single matrix `a` in the order in which their regressors enter the data matrix and the draws of the error term are stored as draws of the inverse of Σ_u. Draws of A_0 are reduced to the free elements below its diagonal, which are the elements that are estimated.

Since a model that is constructed from posterior draws does not contain the specification of its priors, the elements `priors` and `initial` of the resulting object are empty. Use `add_priors` and `add_initial_values` if the model should be re-estimated.

Value

An object of class `"bvarmodel"` containing the following components:

`model`	a list containing information on the model specification.
`data`	a list of data objects. Element `original` contains the original time-series objects of endogenous, unmodelled and deterministic variables. Element `train` contains the time-series objects `y` and `x` of dependent variables and regressors as well as the matrix `z` of regressors in SUR form.
`posterior`	a list of posterior draws. Element `a` contains an $S \times (K(pK + (1+s)M + N) + K(K-1)/2)$ "mcmc" object of the draws of the coefficients of the conditional mean and, if provided, the corresponding draws of inclusion parameters in element `lambda` and of the error covariance matrix of the state equation in element `sigma`. Element `u_sigma_inv` contains an $S \times K^2$ "mcmc" object of the draws of the inverse of the error variance-covariance matrix. For time varying parameter and stochastic volatility models the draws of all periods are appended to each other.

See Also

Other posterior simulation: add_forecast_input.bvarmodel(), add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecast
add_posterior_forecasts.bvecmodel(), add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel
add_seed(), bayests_files(), bayests_posterior(), bvec(), chain_diagnostics(), predict.bvecmodel()

Examples

```
# Get data
data("e1")
e1 <- diff(log(e1))
e1 <- window(e1, end = c(1978, 4))

# Generate model data
data <- create_bvarmodel(e1, p = 2, deterministic = "const")

# Add priors
model <- add_priors(data,
                    coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = 1, scale = .00001))

# Set RNG seed for reproducibility
set.seed(1234567)

iterations <- 400 # Number of iterations of the Gibbs sampler
# Chosen number of iterations and burnin should be much higher.
burnin <- 100 # Number of burn-in draws
draws <- iterations + burnin # Total number of MCMC draws

y <- t(model$data$train$y)
x <- t(model$data$train$x)
tt <- ncol(y) # Number of observations
k <- nrow(y) # Number of endogenous variables
m <- k * nrow(x) # Number of estimated coefficients

# Priors
a_mu_prior <- model$priors$a$mu # Vector of prior parameter means
a_v_i_prior <- model$priors$a$v_inv # Inverse of the prior covariance matrix

u_sigma_df_prior <- model$priors$u_sigma$df # Prior degrees of freedom
u_sigma_scale_prior <- model$priors$u_sigma$scale # Prior covariance matrix
u_sigma_df_post <- tt + u_sigma_df_prior # Posterior degrees of freedom

# Initial values
u_sigma_i <- diag(1 / .00001, k)

# Data containers for posterior draws
draws_a <- matrix(NA, m, iterations)
draws_sigma <- matrix(NA, k^2, iterations)

# Start Gibbs sampler
for (draw in 1:draws) {
```

```
 # Draw conditional mean parameters
 a <- post_normal(y, x, u_sigma_i, a_mu_prior, a_v_i_prior)

 # Draw variance-covariance matrix
 u <- y - matrix(a, k) %*% x # Obtain residuals
 u_sigma_scale_post <- solve(u_sigma_scale_prior + tcrossprod(u))
 u_sigma_i <- matrix(rWishart(1, u_sigma_df_post, u_sigma_scale_post)[,, 1], k)

 # Store draws
 if (draw > burnin) {
  draws_a[, draw - burnin] <- a
  draws_sigma[, draw - burnin] <- solve(u_sigma_i)
 }
}

# Generate bvarmodel object
bvar_est <- bvar(data = e1,
                 y = model$data$train$y, x = model$data$train$x,
                 A = draws_a[1:18,], C = draws_a[19:21, ],
                 Sigma = draws_sigma)
```

`bvartools_model` *The Structure of a Model Object*

Description

What the object every step of an analysis passes along is made of, where each part comes from and how the numbers in it are laid out.

Details

`create_bvarmodel` and `create_bvecmodel` return a list of class `'bvarmodel'` or `'bvecmodel'`, and every later step returns that list with something added. Each call therefore has to be assigned back: `model <- add_priors(model, ...)`.

The object is an ordinary list, so anything in it can be reached with `$` or `[[`, and anything the package does not do can be done to it directly. The layout below is the contract that makes that safe.

The elements

`model` The specification. `k` endogenous variables, `p` lags of them, `m` unmodelled variables with `s` lags, `n` deterministic terms, and the names in `endogen`, `exogen` and `deterministic`. `error` names the error distribution, `varsel` the variable selection scheme, `tvp` whether the coefficients drift and `structural` whether the model is identified. `iterations` and `burnin` size the chain, `thin` is present only above one, and `algorithm` names the sampler that will draw it. A VEC model adds `rank`, `k_beta` and the deterministic terms restricted to the cointegration space; a quantile VAR adds `quantile`, or `quantiles` for a grid of them, and `add_posterior_forecasts` a grid's `forecast_quantile`. `add_dummy_variables` adds `dummy_variables`, the rule each of its dummies is continued by in a forecast. `add_seed` adds `seed`, `add_forecast_input` adds `h` and `forecast_states`.

`data$original` The series as given, before any lag was taken: `endogen`, and `exogen` and `deterministic` where there are any. What the model is rebuilt from when a window of it is taken.

`data$train` The estimation sample. `y` is one row per period and one column per variable; `x` is the regressors in the same compact layout, one row per period; `z` is the same regressors in SUR form, `kron(x, I_k)`, which is what the samplers of a VAR read. A VEC model adds `w`, the lagged levels the cointegration term is formed from. A VAR estimated from a panel not observed whole adds `constraints`, what was observed of it: `value` and `group` with one element per observation, and `row`, `period`, `variable` and `weight` with one per entry of the linear combination each observation is, counted from one. See `create_bvarmodel`.

`data$forecast` Added by `add_forecast_input`: the regressors of the forecast periods, `x` in the compact layout, one row per horizon. A VEC model's are in the layout of its VAR representation, which is the form it is forecast in.

`data$test$y` Added by `add_forecast_errors`: what the horizon realised, one row per period and one column per variable, the periods of the horizon and nothing else. The only thing under `data` that no sampler reads – it is what a forecast is scored against, not anything it is estimated from – and what lets a model that was written to a file be scored again from the file alone.

`priors` Added by `add_priors`: the prior hyperparameters, one block per parameter block of the posterior. Prior variances are given as precisions. Which elements are needed depends on `error`; the method's own help page, `add_priors.bvarmodel` or `add_priors.bvecmodel`, lists them. `a$shrinkage`, `mu` and `constraints` are added by `add_prior_options`.

`initial` Added by `add_initial_values`: the starting values the sampler begins from.

`posterior` Added by `add_posterior_coefficients` and the later `add_posterior_*` functions. The draws; see below.

The draws

`posterior` holds one named block per group of parameters, and each block is a list whose `coeffs` carries the draws:

`a` The coefficients. `coeffs` has M columns, or TM with `tvp = TRUE`, the whole path of a draw stacked by period. `lambda` holds the inclusion indicators where variable selection is on and `sigma` the variances of the random walk where the coefficients drift.

`u_sigma_inv` The error precision matrix, K^2 columns per period, vectorised. `sigma` holds the variance of the log-volatility innovations under stochastic volatility, which is what a forecast steps the volatility forward by.

`u_omega_inv` The diagonal of that precision, K columns per period, for every error but `"wishart"`.

`psi` The triangular matrix of the covariance block, K^2 columns – the whole matrix, not only its free elements – with `lambda` and `sigma` as above.

`u_scale` The scale of the asymmetric Laplace, K columns, for a quantile VAR, and K per level for a grid of quantiles, whose `a` is stacked the same way and which keeps no `u_sigma_inv` or `u_omega_inv`.

`y` The completed panel of a VAR estimated from one not observed whole, TK columns stacked by period, and `constraints_inv` the error precisions of its soft constraints, one column per group.

`mu` The unconditional means under a steady-state prior, K columns. Under an adaptive prior `a` adds `shrinkage`, the scales of its groups, and for the horseshoe and the normal-gamma prior `local`, one scale per coefficient, and `theta` where the normal-gamma prior draws it.

`beta` The cointegration vectors of a VEC model, $k_\beta r$ columns, and `rho` where a time-varying cointegration space put a prior on its state autoregression.

`q` The rotations of a sign-restricted identification, added by `add_sign_restrictions`.

`loglik` The pointwise log-likelihood of the estimation sample, one column per period, added by `add_posterior_loglik`. A matrix of draws rather than a block with `coeffs`, since there is only one thing to hold.

`forecast` Everything the forecast periods produce: `forecasts`, the simulated paths from `add_posterior_forecasts`, and `errors`, what `add_forecast_errors` took against `data$test$y`. Both have Kh columns, stacked by period. The members are named after what they hold rather than one of them being `draws`, because all of them are draws.

Two conventions worth knowing

Draws are rows. Every matrix of draws is a `mcmc` object with one row per draw and one column per parameter, carrying the `start`, `end` and `thin` of the chain. A posterior mean is `colMeans()` and a credible interval a column quantile; transposing to "fix" the shape gives a summary of the wrong thing. `bvar` and `bvec`, which collect the draws of a sampler written by hand, are the exception and take the transpose.

A period is a block of columns. Where a quantity moves with time, the path of one draw is stacked by period along that draw's row, so period t of a K^2 wide block is columns $(t-1)K^2+1$ to tK^2. The same holds of a forecast: the K variables of the first horizon come first, then those of the second.

Lists of models

Passing a vector to an argument such as `p`, `r` or `quantile` gives one model per specification in a list of class `'modellist'`, and `use_expanding_window` a list of class `'expandingwindow'`, one model per window. Every step of the workflow has a method for both and applies to each model in turn, so the structure above is what is inside either of them. `open_models` is the same idea for a folder too large to hold: a handle to the files rather than the models.

The same layout on disk

`write_to_hdf5` writes the object to an HDF5 file with the paths the elements above are named by – `/data/train/y`, `/data/test/y`, `/posterior/a/coeffs`, `/posterior/forecast/forecasts` – and `read_model_from_hdf5` reads it back. It is the format the BayesTS command line runs on, which is what makes a model estimated here and a model estimated there the same object.

In the file every dataset is one row per quantity and one column per draw. R's readers reverse the dimension order, so a session sees the transpose of that, which is the draws-in-rows layout above.

See Also

`bvartools` for the workflow the steps are taken in, and `summary.bvarmodel` for the summaries the package takes of the draws itself.

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100

model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = 1, scale = .0001))
model <- add_posterior_coefficients(add_initial_values(model))

# The specification and the sample
model[["model"]][["k"]]
dim(model[["data"]][["train"]][["y"]])

# The draws, one row per draw and one column per parameter
draws <- model[["posterior"]][["a"]][["coeffs"]]
dim(draws)
colMeans(draws)
```

---

bvec *Bayesian Vector Error Correction Objects*

Description

bvec collects the posterior draws of a vector error correction model in an object of class 'bvec-model'.

Usage

```
bvec(
  y,
  alpha = NULL,
  beta = NULL,
  beta_x = NULL,
  beta_d = NULL,
  r = NULL,
  w = NULL,
  w_x = NULL,
  w_d = NULL,
  Gamma = NULL,
  Upsilon = NULL,
  C = NULL,
  x = NULL,
  x_x = NULL,
```

```
  x_d = NULL,
  A0 = NULL,
  Sigma = NULL,
  data = NULL,
  exogen = NULL,
  z = NULL,
  error = NULL,
  varsel = NULL,
  iterations = NULL,
  burnin = 0
)
```

Arguments

y	a time-series object of differenced endogenous variables, usually, a result of a call to create_bvecmodel.
alpha	a $Kr \times S$ matrix of MCMC coefficient draws of the loading matrix α.
beta	a $Kr \times S$ matrix of MCMC coefficient draws of cointegration matrix β corresponding to the endogenous variables of the model.
beta_x	a $Mr \times S$ matrix of MCMC coefficient draws of cointegration matrix β corresponding to unmodelled, non-deterministic variables.
beta_d	a $N^R r \times S$ matrix of MCMC coefficient draws of cointegration matrix β corresponding to restricted deterministic terms.
r	an integer of the rank of the cointegration matrix. If NULL (default), it is obtained from argument alpha.
w	a time-series object of lagged endogenous variables in levels, which enter the cointegration term, usually, a result of a call to create_bvecmodel.
w_x	a time-series object of lagged unmodelled, non-deterministic variables in levels, which enter the cointegration term, usually, a result of a call to create_bvecmodel.
w_d	a time-series object of deterministic terms, which enter the cointegration term, usually, a result of a call to create_bvecmodel.
Gamma	a $(p-1)K^2 \times S$ matrix of MCMC coefficient draws of differenced lagged endogenous variables or a named list, where element coeffs contains a $(p-1)K^2 \times S$ matrix of MCMC coefficient draws of lagged differenced endogenous variables and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed.
Upsilon	an $sMK \times S$ matrix of MCMC coefficient draws of differenced unmodelled, non-deterministic variables or a named list, where element coeffs contains a $sMK \times S$ matrix of MCMC coefficient draws of unmodelled, non-deterministic variables and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed.
C	an $KN^{UR} \times S$ matrix of MCMC coefficient draws of unrestricted deterministic terms or a named list, where element coeffs contains a $KN^{UR} \times S$ matrix of MCMC coefficient draws of deterministic terms and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed.

x	a time-series object of $K(p-1)$ differenced endogenous variables.
x_x	a time-series object of Ms differenced unmodelled regressors.
x_d	a time-series object of N^{UR} deterministic terms that do not enter the cointegration term.
A0	either a $K^2 \times S$ matrix of MCMC coefficient draws of structural parameters or a named list, where element coeffs contains a $K^2 \times S$ matrix of MCMC coefficient draws of structural parameters and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed.
Sigma	a $K^2 \times S$ matrix of MCMC draws for the error variance-covariance matrix or a named list, where element coeffs contains a $K^2 \times S$ matrix of MCMC draws for the error variance-covariance matrix and element lambda contains the corresponding draws of inclusion parameters in case variable selection algorithms were employed to the covariances. For models with stochastic volatility the matrix must be $TK^2 \times S$.
data	the original time-series object of endogenous variables in levels. If NULL (default), it is reconstructed from arguments y and w.
exogen	the original time-series object of unmodelled variables in levels. If NULL (default), it is reconstructed from arguments w_x and x_x.
z	a $TK \times K(r + (p-1)K + sM + N)$ data matrix, with $K(K-1)/2$ further columns for the contemporaneous endogenous variables of a structural model, usually, a result of a call to create_bvecmodel. If NULL (default), it is generated from the provided data.
error	a character specifying the model that was used for the estimation of the covariance matrix of the error term. If NULL (default), it is inferred from the draws in argument Sigma, which can only distinguish a constant covariance matrix from a time varying one. See create_bvecmodel.
varsel	a character specifying the variable selection algorithm that was employed. If NULL (default), it is inferred from the presence of draws of inclusion parameters. See create_bvecmodel.
iterations	an integer of the number of MCMC draws in the provided coefficient matrices. If NULL (default), it is obtained from those matrices.
burnin	an integer of the number of MCMC draws that were used to initialise the sampler. Defaults to zero, since the draws provided to the function are assumed to exclude them already.

Details

For the vector error correction model with unmodelled exogenous variables (VECX)

$$A_0 \Delta y_t = \alpha \beta' \begin{pmatrix} y_{t-1} \\ x_{t-1} \\ d^R_{t-1} \end{pmatrix} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \sum_{i=0}^{s-1} \Upsilon_i \Delta x_{t-i} + C^{UR} d^{UR}_t + u_t$$

the function collects the S draws of a Gibbs sampler in a standardised object, where Δy_t is a K-dimensional vector of differenced endogenous variables and A_0 is a $K \times K$ matrix of structural

coefficients. α is the $K \times r$ loading matrix and β the $(K + M + N^R) \times r$ cointegration matrix of the error correction term, where y_{t-1}, x_{t-1} and d^R_{t-1} are the first lags of endogenous, exogenous variables in levels and restricted deterministic terms, respectively. Γ_i is a coefficient matrix of lagged differenced endogenous variabels. Δx_t is an M-dimensional vector of unmodelled, non-deterministic variables and Υ_i its corresponding coefficient matrix. d_t is an N^{UR}-dimensional vector of unrestricted deterministics and C^{UR} the corresponding coefficient matrix. u_t is an error term with $u_t \sim N(0, \Sigma_u)$.

For time varying parameter and stochastic volatility models the respective coefficients and error covariance matrix of the above model are assumed to be time varying, respectively.

The draws of the different coefficient matrices provided in `alpha`, `beta`, `beta_x`, `beta_d`, `A0`, `Gamma`, `Upsilon`, `C` and `Sigma` have to correspond to the same MCMC iteration.

The result is the same kind of object as the output of `create_bvecmodel` in combination with `add_posterior_coefficients`, so it can be used with the methods of class 'bvecmodel' such as `vec_to_var.bvecmodel` or `summary.bvecmodel`. Accordingly, the draws are stored in the parameterisation those methods expect. The error correction term is described by α and β and not by $\Pi = \alpha\beta'$, since the latter cannot be decomposed into the former. The draws of α are collected in the same matrix as the coefficients of the remaining regressors and the draws of β of all cointegration relations are collected in a matrix of their own. Draws of the error term are stored as draws of the inverse of Σ_u and draws of A_0 are reduced to the free elements below its diagonal, which are the elements that are estimated.

Since a model that is constructed from posterior draws does not contain the specification of its priors, the elements `priors` and `initial` of the resulting object are empty. Use `add_priors` and `add_initial_values` if the model should be re-estimated.

Value

An object of class `"bvecmodel"` containing the following components:

`model`	a list containing information on the model specification.
`data`	a list of data objects. Element `original` contains the original time-series objects of endogenous and unmodelled variables in levels. Element `train` contains the time-series objects `y` of differenced endogenous variables, `w` of the regressors of the error correction term and `x` of the remaining regressors as well as the matrix `z` of regressors in SUR form.
`posterior`	a list of posterior draws. Element `a` contains an $S \times (Kr + K((p-1)K + sM + N^{UR}) + K(K-1)/2)$ "mcmc" object of the draws of the loading matrix and the coefficients of the remaining regressors and, if provided, the corresponding draws of inclusion parameters in element `lambda` and of the error covariance matrix of the state equation in element `sigma`. Element `beta` contains an $S \times ((K + M + N^R)r)$ "mcmc" object of the draws of the cointegration matrix. Element `u_sigma_inv` contains an $S \times K^2$ "mcmc" object of the draws of the inverse of the error variance-covariance matrix.

See Also

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecas`

add_posterior_forecasts.bvecmodel(), add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel
add_seed(), bayests_files(), bayests_posterior(), bvar(), chain_diagnostics(), predict.bvecmodel()

Examples

```
# Load data
data("e6")
# Generate model
model <- create_bvecmodel(e6, p = 4, r = 1, const = "unrestricted", seasonal = "unrestricted")
# Obtain data matrices
y <- t(model$data$train$y)
w <- t(model$data$train$w)
x <- t(model$data$train$x)

# Reset random number generator for reproducibility
set.seed(1234567)

iterations <- 400 # Number of iterations of the Gibbs sampler
# Chosen number of iterations should be much higher, e.g. 30000.

burnin <- 100 # Number of burn-in draws
draws <- iterations + burnin

r <- 1 # Set rank

tt <- ncol(y) # Number of observations
k <- nrow(y) # Number of endogenous variables
k_w <- nrow(w) # Number of regressors in error correction term
k_x <- nrow(x) # Number of differenced regressors and unrestrictec deterministic terms

k_alpha <- k * r # Number of elements in alpha
k_beta <- k_w * r # Number of elements in beta
k_gamma <- k * k_x

# Set uninformative priors
a_mu_prior <- matrix(0, k_alpha + k_gamma) # Vector of prior parameter means
a_v_i_prior <- diag(0, k_alpha + k_gamma) # Inverse of the prior covariance matrix

v_i <- 0
p_tau_i <- diag(1, k_w)

u_sigma_df_prior <- r # Prior degrees of freedom
u_sigma_scale_prior <- diag(0.01, k) # Prior covariance matrix
u_sigma_df_post <- tt + u_sigma_df_prior # Posterior degrees of freedom

# Initial values
beta <- matrix(c(1, -4), k_w, r)
u_sigma_i <- diag(1 / .0001, k)
g_i <- u_sigma_i

# Data containers
draws_alpha <- matrix(NA, k_alpha, iterations)
draws_beta <- matrix(NA, k_beta, iterations)
```

```
draws_gamma <- matrix(NA, k_gamma, iterations)
draws_sigma <- matrix(NA, k^2, iterations)

# Start Gibbs sampler
for (draw in 1:draws) {
  # Draw conditional mean parameters
  temp <- post_coint_kls(y = y, beta = beta, w = w, sigma_i = u_sigma_i,
                         v_i = v_i, p_tau_i = p_tau_i, g_i = g_i,
                         x = x,
                         gamma_mu_prior = a_mu_prior,
                         gamma_v_i_prior = a_v_i_prior)
  alpha <- temp$alpha
  beta <- temp$beta
  Pi <- temp$Pi
  gamma <- temp$Gamma

  # Draw variance-covariance matrix
  u <- y - Pi %*% w - matrix(gamma, k) %*% x
  u_sigma_scale_post <- solve(tcrossprod(u) +
     v_i * alpha %*% tcrossprod(crossprod(beta, p_tau_i) %*% beta, alpha))
  u_sigma_i <- matrix(rWishart(1, u_sigma_df_post, u_sigma_scale_post)[,, 1], k)
  u_sigma <- solve(u_sigma_i)

  # Update g_i
  g_i <- u_sigma_i

  # Store draws
  if (draw > burnin) {
    draws_alpha[, draw - burnin] <- alpha
    draws_beta[, draw - burnin] <- beta
    draws_gamma[, draw - burnin] <- gamma
    draws_sigma[, draw - burnin] <- u_sigma
  }
}

# Number of non-deterministic coefficients
k_nondet <- (k_x - 4) * k

# Generate bvecmodel object
bvec_est <- bvec(y = model$data$train$y, w = model$data$train$w,
                 x = model$data$train$x[, 1:6],
                 x_d = model$data$train$x[, 7:10],
                 r = r,
                 alpha = draws_alpha,
                 beta = draws_beta,
                 Gamma = draws_gamma[1:k_nondet,],
                 C = draws_gamma[(k_nondet + 1):nrow(draws_gamma),],
                 Sigma = draws_sigma)
```

---

chain_diagnostics *Convergence Diagnostics of Several Chains*

Description

Compares the chains of a model estimated with `chains` above one in `add_posterior_coefficients` and reports, for every parameter, the potential scale reduction factor and two effective sample sizes.

Usage

```
chain_diagnostics(object, ...)
```

Arguments

`object`	an object of class 'bvarmodel' or 'bvecmodel', whose posterior was simulated with `chains` of at least two.
`...`	further arguments passed to or from other methods.

Details

A single chain can look converged and not be: its autocorrelations and its effective sample size describe how well it explores the region it is in, not whether that region is the posterior. Chains started from the same values but drawing different random numbers settle in different places if the posterior has several modes or the sampler has not left the neighbourhood of its start, and comparing them is the check a single chain cannot provide.

The statistic is the rank-normalised split $\hat{R}$ of Vehtari et al. (2021). Every chain is cut into two halves, the pooled draws are replaced by their ranks and put back on a normal scale, and the variance between the means of the halves is compared with the variance within them,

$$\hat{R} = \sqrt{\frac{\frac{n-1}{n}W + \frac{1}{n}B}{W}},$$

where n is the length of a half, W the average variance within the halves and B n times the variance of their means. Splitting the chains catches a chain that is still drifting.

The rank normalisation is what makes the number mean the same thing whatever scale the parameter is on. The plain version of this statistic is built on variances, so it is undefined for a posterior heavy-tailed enough to have none and it moves when the same draws are transformed monotonically – which is awkward for a package whose parameters are freely rescaled and whose variance blocks are reported both as `omega` and as its square. What is reported is the larger of the rank-normalised $\hat{R}$ and the one computed on the draws folded around their median, so that chains agreeing about the centre but not about the spread are caught as well.

Values close to one indicate that the chains describe the same distribution; above 1.01, Vehtari et al. (2021) recommend running the chains longer or reconsidering the model. A parameter that does not vary in any chain, such as a coefficient that variable selection excluded throughout or an equation that `iid` left without coefficients, has no $\hat{R}$ and is reported as `NA`.

The signed standard deviations `omega` of a block estimated under the non-centred prior `omega_v` are symmetric around zero by construction – the sampler switches their sign at random – so their $\hat{R}$ is close to one whether or not the chains agree. Their squares, `sigma`, which are in the posterior beside them, are the ones to read.

Two effective sample sizes are reported, and for this package the second is usually the one that matters. `ess_bulk` is computed on the rank-normalised draws and says how much independent

information the sample carries about the centre of the posterior. `ess_tail` is the smaller of the effective sample sizes at the 5th and 95th percentiles and says the same about the extremes. Almost nothing this package reports is a point: `irf`, `fevd` and the forecasts all come back as quantiles, and it is `ess_tail` that says whether those quantiles have settled. A sample can carry a comfortable `ess_bulk` and still have a band that moves from one run to the next.

Neither is a convergence diagnostic: chains stuck in different modes can each have a large one.

The chains start from the same initial values, those of `add_initial_values`, and differ in their random numbers. Dispersed starting values would make the comparison more demanding.

Value

A data frame with one row per parameter of every block of posterior draws, other than the forecasts and the log-likelihood, and the columns `block`, `parameter`, `rhat`, `ess_bulk` and `ess_tail`.

References

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis* (3rd ed.). Boca Raton: CRC Press.

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2021). Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC. *Bayesian Analysis, 16*(2), 667–718. doi:10.1214/20BA1221

See Also

Other posterior simulation: `add_forecast_input.bvarmodel()`, `add_forecast_input.bvecmodel()`, `add_posterior_coefficients.bvarmodel()`, `add_posterior_coefficients.bvecmodel()`, `add_posterior_forecast`
`add_posterior_forecasts.bvecmodel()`, `add_posterior_loglik.bvarmodel()`, `add_posterior_loglik.bvecmodel`
`add_seed()`, `bayests_files()`, `bayests_posterior()`, `bvar()`, `bvec()`, `predict.bvecmodel()`

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100

model <- create_bvarmodel(e1, p = 1, deterministic = "const",
                          iterations = 200, burnin = 100)
model <- add_priors(model, coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = "k", scale = 1))
model <- add_initial_values(model)
model <- add_posterior_coefficients(model, chains = 2)

diag <- chain_diagnostics(model)
max(diag$rhat, na.rm = TRUE)
```

choose_best_model *Choose Best Model*

Description

A generic function that applies model selection criteria to find the best model.

Usage

```
choose_best_model(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of object, as described on the pages of the methods.

See Also

Methods: choose_best_model.selcritlist.

choose_best_model.selcritlist *Choose Best Model*

Description

Chooses the best model according the selection criteria in an object of class 'selcritlist'.

Usage

```
## S3 method for class 'selcritlist'
choose_best_model(object, criterion = "WAIC", ...)
```

Arguments

object	object of class 'selcritlist', usually, a result of a call to selection_criteria.
criterion	the selection criterion that should be used. Available choices are "LL", "AIC", "BIC", "HQ", "WAIC" (default), "LOOIC", "LPL" and, for the discounted models, "LML".
...	further arguments passed to or from other methods.

Details

If argument `criterion` is "LL", "LPL" or "LML", the model with the maximum value is chosen, otherwise, the model with the minimum value.

Which criterion to use depends on the models that are compared. See `selection_criteria`, whose details set out what each of them penalises and when a count of parameters ceases to describe a model. The default is `"WAIC"`, because it is the criterion that stays defined across the models this package estimates: it penalises by the flexibility the fit used rather than by a count of parameters, which describes neither a model whose coefficients or variances follow a state equation nor one whose prior shrinks them. `"AIC"`, `"BIC"` and `"HQ"` remain the criteria for the lag order of a model with constant coefficients and a weak prior, which is the case they are derived for. `"LPL"`, the log predictive likelihood of an expanding window exercise with `add_predictive_loglik`, is the criterion for the rank of a VEC model whose coefficients or variances follow a state equation.

`"LML"` is the one criterion of the discounted models and the only one they carry. It is the exact log marginal likelihood of the sample, read off the file rather than estimated, so a grid over the rank, the cointegration matrix, the lag order or the two discounts is compared by it directly. It is not comparable with `"LL"`, which conditions on the parameters where this integrates them out.

Value

An integer giving the position of the best model in the list provided in argument `object`. Where several models attain the best value exactly, the positions of all of them, in increasing order.

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `create_external_forecast`, `folder_steps`, `map_draws()`, `map_models()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 1:3, deterministic = "const",
                          iterations = 50, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)
```

```
# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Obtain posterior log-likelihoods
model <- add_posterior_loglik(model)

# Calculate selection criteria
sel <- selection_criteria(model)

# View results
sel

# Choose best model according to WAIC, the default
choose_best_model(sel)

# Choose best model according to AIC
choose_best_model(sel, criterion = "AIC")
```

`coint_kls2010_reparameterise_two` *Cointegration Reparameterisation*

Description

Performs the second transformation of the loading and cointegration matrix as proposed in Koop et al. (2010).

Usage

```
coint_kls2010_reparameterise_two(alpha, beta)
```

Arguments

`alpha`	a $K \times r$ matrix.
`beta`	an $M \times r$ matrix.

Details

The function performs two transformations:

- $A = \alpha(\beta'\beta)^{1/2}$
- $B = \beta(\beta'\beta)^{-1/2}$

Value

A list of two matrices:

`alpha`	Loading matrix A.
`beta`	Cointegration matrix B (semiorthogonal).

References

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224-242. doi:10.1080/07474930903382208

Examples

```
# Generate input data
alpha <- matrix(c(-0.07, 0.17), 2)
beta <- matrix(c(1, -4), 2)

# Reparameterise
coint_kls2010_reparameterise_two(alpha, beta)
```

`coint_prepare_sur_data` *Posterior Data Preparation*

Description

Generates the input matrix for the simulation of the cointegration matrix β.

Usage

```
coint_prepare_sur_data(w, alpha, k, r, reparameterise, tvp)
```

Arguments

`w`	a $T \times M$ matrix.
`alpha`	a $K \times r$ matrix of the loading matrix α.
`k`	integer of the number of endogenous variables in the model.
`r`	integer with the rank of the cointegration matrix Π.
`reparameterise`	logical indicating if the reparameterisation described in Koop et al. (2010) should be performed.
`tvp`	logical indicating if the SUR matrix with the values of regressors should be prepared for the estimation of constant or time varying parameters.

Value

A sparse block matrix.

Examples

```
data("e6") # Load data

# Generate model input
mod <- create_bvecmodel(e6, p = 4, r = 1,
                        const = "unrestricted", seasonal = "unrestricted")

# Obtain input data
alpha <- matrix(c(-0.1, 0.16, -0.04, -0.02), 2)
w <- mod$data$train$w

# Constant coefficients
coint_prepare_sur_data(w, alpha, 2, 1, TRUE, FALSE)

# Time varying coefficients
coint_prepare_sur_data(w, alpha, 2, 1, FALSE, TRUE)
```

---

cointspace_prior *Build the Prior on the Cointegration Space*

Description

Checks the specification of the prior on the cointegration space of a VEC model and builds it, in the form in which add_priors stores it as element priors$beta.

Usage

```
cointspace_prior(object, coint)
```

Arguments

object	a list of class 'bvecmodel', or any model with the same layout: elements model$rank and model$tvp, and the regressors data$train$y, data$train$w and data$train$x.
coint	a named list of prior specifications for the coefficients determining the cointegration space. It has no default. See section 'Prior on the cointegration space'.

Details

add_priors calls this function for VEC models, so it rarely needs to be called directly. It is exported for packages that build VEC models of their own layout, such as the sub-models of a global VEC model, whose error correction term also holds weakly exogenous and global variables. Calling it gives them the same checks and the same prior.

All dimensions are taken from the regressors, not from the model specification: the prior has one row and column for each series in the error correction term data$train$w, and one block for each of the model$rank cointegration vectors.

Value

NULL for a model without cointegration, model$rank = 0, whose coint is checked all the same. Otherwise a list with type = "cointspace" and, for constant cointegration parameters, v_inv and p_tau_inv, and g_inv if g_i was given, or, for time varying ones, rho, mu and v_inv of the state equation, together with rho_min and rho_max for a uniform prior on ρ and the transition p_tau added by p_tau_i = "ml".

Prior on the cointegration space

Argument coint can contain the following elements. Which of them are required depends on whether the cointegration vectors are constant or time varying, since the two are given different kinds of prior: a matric-variate prior on the cointegration space in the first case, and a state equation in the second. Any other element raises an error.

v_i non-negative numeric specifying the shrinkage of the cointegration space prior, or "ml". See below. Required for models with constant cointegration parameters and not used otherwise.

p_tau_i the inverse of the matrix P_τ, which determines the central location of the cointegration space $sp(\beta)$. Either a numeric of its diagonal elements, a full symmetric matrix, or "ml", with one row and column per series in the error correction term data$train$w. See below. Required for models with constant cointegration parameters. For models with time varying cointegration parameters only "ml" is used.

weight positive numeric specifying the weight of a prior centred on the maximum likelihood estimate in units of the information of the sample. Default is 1. Only used if p_tau_i = "ml".

rho a numeric specifying the autocorrelation coefficient of the state equation of β. It must be smaller than 1. Required for models with time varying cointegration parameters and not used otherwise. If rho_min and rho_max are given as well, this is the value the chain starts ρ at rather than the value it keeps, and it must lie between them.

rho_min, rho_max numerics specifying the support of a uniform prior on ρ, which makes it a drawn parameter rather than a fixed hyperparameter. Optional, and either both or neither; they must satisfy $0 <$rho_min$<$rho_max≤ 1. Koop et al. (2011) use $(0.999, 1)$. Only used for models with time varying cointegration parameters.

g_i the inverse of the matrix G that scales the prior of the loadings, for models with constant cointegration parameters and stochastic volatility, error = "sv" or "sv+covar", and refused for every other model. Either a numeric of its diagonal elements, a full symmetric positive definite matrix with one row and column per endogenous variable, or "ml" for the inverse of the maximum likelihood estimate of the error covariance. Optional. See below.

For a model with constant cointegration parameters the prior is that of Koop et al. (2010). The sampler uses v_i and p_tau_i only through their product, so with v_i = 0 the prior on the cointegration space is uniform whatever p_tau_i is. An informative prior on the space therefore needs a positive v_i, which also shrinks the loadings: for β close to the centre of the space they have prior $N(0, \Sigma/v)$.

In Koop et al. (2010) the loadings' prior is scaled by a matrix G, which may be the error covariance Σ or any fixed, known matrix. The models with a constant error covariance take $G = \Sigma$. With stochastic volatility the covariance differs from period to period, so G is fixed for the whole run instead: G^{-1} is g_i if it is given, and otherwise the error precision implied by the starting values of the log-volatilities, averaged over the sample once before the first draw. That fallback depends on

`add_initial_values`, so giving `g_i` makes the prior independent of how the chain is started. `g_i = "ml"` uses Johansen's (1995) estimate of the error covariance, the same one `v_i = "ml"` is based on.

With `p_tau_i = "ml"` the prior is centred on the space spanned by Johansen's (1995) maximum likelihood estimate $\hat{\beta}$, computed from the error correction term as it is stored in the model – so call `scale_error_correction` first if the series should be scaled; scaling is refused afterwards. Let H be an orthonormal basis of that space, $H_\perp$ one of its orthogonal complement and $\beta = H + H_\perp\delta$. The matrix

$$P_\tau^{-1} = HH' + H_\perp T^{-1} H_\perp', \quad T = \frac{v}{w}(H_\perp' S_{11} H_\perp)^{-1},$$

where S_{11} is the cross product of the residuals of a regression of the error correction term on the short-run regressors and w is `weight`, makes the prior of δ given α $N(0, (\alpha'\Sigma^{-1}\alpha)^{-1} \otimes w^{-1}(H_\perp' S_{11} H_\perp)^{-1})$. This is the asymptotic distribution of the maximum likelihood estimator with its precision multiplied by w: `weight = 1` adds as much information about the space as the sample itself holds. Since the centre is estimated from the same sample, the posterior then overstates the precision of $sp(\beta)$. Eigenvalues of T are capped at one, which is the uniform prior.

With `v_i = "ml"` the shrinkage is chosen so that the prior mean of $tr(\alpha'\Sigma^{-1}\alpha)$ is 100 times its maximum likelihood estimate. It can be combined with a numeric `p_tau_i`.

For a model with time varying cointegration parameters the state equation is $\beta_t = \rho\beta_{t-1} + \eta_t$ with $\eta_t \sim N(0, I)$, and the prior on the state before the sample is that equation's own stationary distribution, $N(0, I/(1-\rho^2))$. This is what makes the prior proper, so a ρ close to one is intended and one further from it draws a warning. `add_priors` gives the loadings the compensating scale: only the product $\alpha\beta'$ is identified, so their prior variance is shrunk by $1-\rho^2$, leaving the product on the scale `coef$v_i` asks for.

When ρ is drawn, those last two are computed from `coint$rho` once and do not follow the draw: the state before the sample keeps the normal prior built here, and the loadings keep the shrinkage of `add_priors`. This is a deliberate departure from Koop et al. (2011), in whom the state before the sample is the stationary distribution of whatever ρ currently is. It is also what makes the draw an exact Gibbs block rather than their Metropolis-within-Gibbs step: with the state before the sample free of ρ, the conditional posterior of ρ is a normal truncated to the prior's support. Draws of ρ are returned in `object$posterior$beta$rho`.

How far the space moves is not set by the prior on any variance. The steps η_t have unit variance whatever `coef$rate` and `coef$rate_alpha` are, and how far a step moves $\Pi_t = \alpha_t\beta_t'$ depends on the scale of β_t. Only the product is identified, so the posterior can pair small loadings with large cointegration vectors, which a unit step barely turns, or large loadings with small ones, which it turns a lot. Two things decide which. One is the prior precision of the loadings, `coef$v_i` $/(1-\rho^2)$, which restrains the drift only if it is tight relative to the scale of Π the data call for, and that scale depends on the units of the data. The other is the series in the error correction term: a step moves $\beta_t' w_t$ by $\eta_t' w_t$, which grows with the levels of the series and not only with their variation. For log levels far from zero the shift is of the order of the levels themselves, and the sampler either lets it act as a random walk intercept, which no prior on the deterministic terms controls and which can absorb the residuals of an equation, or draws loadings close to zero, which switches the error correction term off.

Both have been seen. In the US sub-model of the **bgvars** data set `gvar2023`, with the oil price among its endogenous variables and a rank of one, a prior that pinned the state variances of all coefficients, the loadings and the constant included, at 1e-14 still left the residual standard deviation of one equation as low as 0.1 to 0.5 of the maximum likelihood one in some chains, with which

equation and how low depending on the chain, and a saved chain reported a posterior mean log-likelihood several hundred above the maximum of the constant coefficient model. The chains do not move between these outcomes within a few thousand draws, so a single chain can report any of them. The Austrian sub-model under the same prior lost fit that the constant coefficient model has, and the chains that were inspected drew loadings close to zero. On data simulated from a constant coefficient VEC model, whose posterior under such a prior should reproduce the maximum likelihood fit, single chains did so only with the series centred and `coef$v_i = 10`, or around zero and `coef$v_i = 100`. Around 12, where log levels usually are, a chain drew loadings close to zero, and around zero with `coef$v_i = 1` one fitted part of the residuals.

The same mechanism makes `scale_error_correction` risky in a model whose coefficients vary over time. It divides the series by the standard deviation of their differences, which makes them, and with them both a drift in a loading and a step of $\beta_t' w_t$, larger by the same factor. The coefficient paths can then absorb the residuals of an equation almost entirely: its residual variance – or, under stochastic volatility, its volatility path – is reported far below that of a least squares fit of the same regressors and nearly constant over the sample. In the model of the vignette on time varying parameters and stochastic volatility in error correction models, `vignette("tvp-sv-vec", package = "bvartools")`, a scaled error correction term did that to one equation in each of four chains with different seeds, and the unscaled one with a small `coef$rate` in none.

It is safer to leave the series unscaled, to set `coef$v_i` with the scale of Π in mind as well as choosing small rates, and to compare the residual variances with those of a least squares fit of the same regressors, in chains with different seeds, for a model that is to be used. Small rates alone do not guarantee that a model passes that check, and passing it is what says that neither the coefficients nor the cointegration vectors fit the residuals. `scale_error_correction(object, scale = FALSE, centre = TRUE)` centres the series before posterior simulation, which removes the part of a step of $\beta_t' w_t$ that comes from the levels of the series rather than from their variation, and `rescale_error_correction` writes the draws back in terms of the series as they are, so that `vec_to_var` and the forecasts can use them. It does not reach the other channel, the scale of the loadings, and it is not a full remedy: in the US sub-model above, centring alone roughly halved how far the error correction term and the constant moved over the sample and still left the oil price equation at about 0.8 of the maximum likelihood residual standard deviation in two chains. Whether a centred model is right is again what the comparison with least squares shows.

For a model with time varying cointegration parameters `p_tau_i = "ml"` centres the marginal prior of the cointegration space on Johansen's estimate as well, using the informative marginal prior of Koop et al. (2011, working paper version). The state equation becomes $\beta_t = \rho(I_r \otimes P_\tau)\beta_{t-1} + \eta_t$ with $P_\tau = HH' + H_\perp T H_\perp'$: the part of β_t along $sp(H)$ keeps ρ, the part off it decays faster, and the mode of the marginal distribution of $sp(\beta_t)$ is $sp(H)$ in every period. T is chosen so that the prior spread of the tilt of β_t away from $sp(H)$ in a period, approximately $T^* = (1-\rho^2)(I-\rho^2 T^2)^{-1}$, is the sampling variance of Johansen's estimator with its precision multiplied by `weight`. The state before the sample is given the stationary distribution the transition implies. The transition is stored as element `p_tau`.

ρ limits how informative this prior can be: even $T = 0$ leaves the tilt a spread of $1 - \rho^2$ per period, and a `weight` asking for more is floored there with a warning.

A larger `weight` does not always give a tighter posterior. It narrows the prior spread of the tilt in each period by lowering T, but T is also how much of the tilt carries over from one period to the next. As T approaches zero the tilt of each period becomes independent of the last, the data of neighbouring periods stop informing it, and the posterior spread of $sp(\beta_t)$ can widen again. On data set E6 with $\rho = 0.999$, for example, `weight = 1`, which gives $T \approx 0.44$, produced a tighter

posterior than `weight = 100`, which reaches $T = 0$. The useful range of `weight` is the one that keeps T clearly above zero, and a ρ closer to one widens it.

A `weight` so small that T would be the identity in every direction leaves the noninformative prior unchanged. Both P_τ and the prior on the state before the sample are computed at `coint$rho`, and do not follow the draw if ρ is drawn. `coint$v_i` has no counterpart for these models.

References

Johansen, S. (1995). *Likelihood-based inference in cointegrated vector autoregressive models*. Oxford: Oxford University Press.

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224–242. doi:10.1080/07474930903382208

Koop, G., León-González, R., & Strachan R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 2, r = 1,
                          const = "unrestricted",
                          iterations = 10, burnin = 10)

# A prior centred on the maximum likelihood estimate of the space
prior <- cointspace_prior(model, coint = list(v_i = 0.01, p_tau_i = "ml"))
prior$p_tau_inv
```

---

combine_models *Combine Models*

Description

Convenience function that combines multiple models into one object.

Usage

```
combine_models(...)
```

Arguments

`...`	One or multiple lists that contain either model objects or posterior objects.

Value

A list of class 'modellist' that holds the models of all arguments in one flat list, in the order in which they were given.

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvarmodel(), add_initial_values.bvecmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), aggregation_weights(), create_bvarmodel(), create_bvecmodel(), transform_variables(), use_expanding_window.bvarmodel(), use_expanding_window.bvecmodel()

Examples

```
# Load data
data(e6)

# Create multiple VAR models
model_1 <- create_bvarmodel(diff(e6), p = 0:2, deterministic = "const")

# Create multiple VEC models
model_2 <- create_bvecmodel(e6, p = 1, r = 0:1, const = "unrestricted")

# Combine the models in one object
model <- combine_models(model_1, model_2)
```

---

covar_prepare_data *Covariance: Data Preparation*

Description

Convenience function, which generates the input data for posterior simulation of covariance parameters.

Usage

```
covar_prepare_data(y, omega_i, k, tt, tvp)
```

Arguments

y	a $KT \times 1$ vector of input data.
omega_i	a $K \times K$ or $KT \times KT$ matrix of error variances. The matrix must be sparse.
k	an integer of the number of endogenous variables.
tt	an integer of the number of observations.
tvp	logical indicating if the SUR matrix with the values of regressors should be prepared for the estimation of constant or time varying parameters.

Details

For the model

$$y_t = Z_t a_t + u_t$$

with $u_t \sim N(0, \Psi\Omega_t\Psi')$ and Ω_t as a diagonal matrix of error variances, the function produces the input data for the posterior simulation of the lower triangular covariance coefficients of Ψ as presented in Primiceri (2005).

Value

A list with three elements:

y	The prepared vector of endogenous variables.
z	The prepared matrix of regressors.
omega_i	The prepared diagonal matrix of measurement error variances.

All matrices are returned as sparse matrices.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Primiceri, G. E. (2005). Time varying structural vector autoregressions and monetary policy. *The Review of Economic Studies 72*(3), 821–852. doi:10.1111/j.1467937X.2005.00353.x

Examples

```
k <- 3
tt <- 4
u <- matrix(1:(k * tt))

# Generate simple variance matrix
omega_i <- Matrix::Matrix(diag(1:k, k))
# Generate block diagonal variance matrix
tv_omega_i <- Matrix::Matrix(0, k * tt, k * tt)
for (i in 1:tt) {
  tv_omega_i[(i - 1) * k + 1:k, (i - 1) * k + 1:k] <- omega_i
}

# Constant error variances

# Constant coefficients
covar_prepare_data(u, omega_i, k, tt, FALSE)
# Time varying coefficients
covar_prepare_data(u, omega_i, k, tt, TRUE)


# Time varying error variances

# Constant coefficients
covar_prepare_data(u, tv_omega_i, k, tt, FALSE)
```

```
# Time varying coefficients
covar_prepare_data(u, tv_omega_i, k, tt, TRUE)
```

`covar_vector_to_matrix` *Covariance: Vector to Matrix*

Description

Convenience function, which takes the vector of draws of lower triangular covariance coefficients and transforms it into a matrix with ones on the main diagonal. In case of time varying parameters the resulting matrix will be block diagonal.

Usage

```
covar_vector_to_matrix(psi, k, tt)
```

Arguments

`psi`	a $K(K-1)/2 \times 1$ or $TK(K-1)/2 \times 1$ vector of input data.
`k`	the number K of endogenous variables.
`tt`	the number T of observations.

Value

A sparse, block diagonal matrix.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Primiceri, G. E. (2005). Time varying structural vector autoregressions and monetary policy. *The Review of Economic Studies 72*(3), 821–852. doi:10.1111/j.1467937X.2005.00353.x

Examples

```
# Create artificial data
k <- 5
tt <- 4
n_covar <- (k - 1) * k / 2

# Constant parameters
psi <- matrix(1:(n_covar))
covar_vector_to_matrix(psi, k, tt)

# Time varying parameters
psi <- matrix(1:(n_covar * tt))
covar_vector_to_matrix(psi, k, tt)
```

`create_bvarmodel` *Create a Vector Autoregressive Model*

Description

Produces the input for the estimation of a vector autoregressive (VAR) model.

Usage

```
create_bvarmodel(
  data,
  p = 2,
  exogen = NULL,
  s = 2,
  deterministic = "const",
  seasonal = FALSE,
  iid = NULL,
  structural = FALSE,
  error = "wishart",
  quantile = 0.5,
  tvp = FALSE,
  varsel = "none",
  algorithm = NULL,
  delta_beta = 1,
  delta_sigma = 1,
  iterations = 20000,
  burnin = 2000,
  thin = 1,
  missing = "omit",
  aggregate = NULL,
  soft = NULL,
  quantile_grid = FALSE
)
```

Arguments

`data`	a time-series object of endogenous variables.
`p`	an integer vector of the lag order (default is `p = 2`).
`exogen`	an optional time-series object of external regressors.
`s`	an integer vector of the lag order of the external regressors (default is `s = 2`). Ignored if `exogen` is `NULL`.
`deterministic`	a character specifying which deterministic terms should be included. Available values are `"none"`, `"const"` (default) for an intercept, `"trend"` for a linear trend, and `"both"` for an intercept with a linear trend.

`seasonal` logical. If `TRUE`, seasonal dummy variables are generated as additional deterministic terms. The amount of dummies depends on the frequency of the time-series object provided in `data`. Defaults to `FALSE`.

`iid` an optional character vector naming endogenous variables whose equations carry no coefficients at all – no lags, no deterministic terms, nothing. They are white noise, and reach the rest of the model only through the error covariance. They must be the first columns of `data`. See 'Details'.

`structural` logical indicating whether data should be prepared for the estimation of a structural VAR model. Defaults to `FALSE`.

`error` character specifying the model that should be used for the estimation of the covariance matrix of the error term. Default is `"wishart"`. See 'Details'.

`quantile` a numeric vector of quantiles in the interval $(0, 1)$ that should be estimated. Only used, if `error = "ald"`. Defaults to `0.5`, the median. One model is created per quantile, so a vector produces a list of models, unless `quantile_grid` is `TRUE`. See 'Details'.

`tvp` logical indicating whether the model parameters are time varying. Defaults to `FALSE`.

`varsel` character specifying the type of variable selection algorithm that should be employed. Default is `"none"`. See 'Details'.

`algorithm` algorithm that should be used for posterior simulation. If `NULL` (default), the algorithm is named by `tvp` and `error`. The one non-standard option is `"discount"`. See 'Details'.

`delta_beta`, `delta_sigma`
numeric discount factors in $(0, 1]$ of the discounted model, the first governing the coefficients and the second the error covariance. Both default to 1, at which the quantity they govern does not move. Ignored unless `algorithm = "discount"`, and a vector in either produces one model per value. See 'Details'.

`iterations` an integer of MCMC draws excluding burn-in draws (defaults to 20000).

`burnin` an integer of MCMC draws used to initialize the sampler (defaults to 2000). These draws do not enter the computation of posterior moments, forecasts etc.

`thin` an integer thinning interval of the sampler (defaults to 1). After the burn-in the sampler keeps the last of every `thin` draws, so it runs `burnin + iterations * thin` draws and still keeps `iterations`. Unlike `thin`, which thins draws already made, the draws that are not kept are never held in memory.

`missing` a character, what to do with periods in which a series of `data` is `NA`. `"omit"` (default) drops them, as `na.omit` does. `"estimate"` keeps them and estimates what was not observed together with the model. See 'Details'.

`aggregate` an optional named list, one element per series of `data` that is observed at a lower frequency than the model, holding the weights with which it aggregates the periods of the model, oldest first, as `aggregation_weights` returns them. The series is `NA` in every period in which it is not observed, and its observations stand in the last period they aggregate. Implies `missing = "estimate"`.

`soft` an optional character vector of series of `data` whose observations hold up to a normal measurement error instead of exactly, each series with an error variance of its own. See 'Details'.

`quantile_grid` logical. If `TRUE`, the quantiles in `quantile` are estimated as one structural quantile VAR rather than as one model each. Needs `error = "ald"` and constant coefficients. Defaults to `FALSE`. See 'Details'.

Details

The function produces the data matrices for vector autoregressive (VAR) models, which can also include unmodelled, non-deterministic variables:

$$A_0 y_t = \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=0}^{s} B_i x_{t-i} + C d_t + u_t,$$

where y_t is a K-dimensional vector of endogenous variables, A_0 is a $K \times K$ coefficient matrix of contemporaneous endogenous variables, A_i is a $K \times K$ coefficient matrix of endogenous variables, x_t is an M-dimensional vector of exogenous regressors and B_i its corresponding $K \times M$ coefficient matrix. d_t is an N-dimensional vector of deterministic terms and C its corresponding $K \times N$ coefficient matrix. p is the lag order of endogenous variables, s is the lag order of exogenous variables, and u_t is an error term.

The model can be rewritten as

$$A_0 y_t = Z_t a + u_t,$$

where Z_t is a $KT \times K * (Kp + M(s+1) + N)$ data matrix and a the corresponding coefficient vector. Unless structural models are be estimated, A_0 is assumed to be an identity matrix.

If a vector is provided as argument `p` or `s`, the function will produce a distinct model for all possible combinations of those specifications.

If `structural = TRUE`, the data matrix Z_t is augmented by negative contemporaneous observations of endogenous variables, which correspond to the coefficients in the lower triangular of A_0.

If `tvp` is `TRUE`, the respective coefficients of the above model are assumed to be time varying. If `error` is `"sv"` or `"sv+covar"`, the error covariance matrix is assumed to be time varying.

Argument `error` specifies the structure of the covariance matrix of the error term and how it is estimated. Possible specifications are:

- `"wishart"`: The covariance is estimated using a Wishart prior.
- `"gamma"`: Only the diagonal elements of the covariance matrix are estimated using a gamma prior. Off-diagonal elements are not estimated and set to zero.
- `"gamma+covar"`: The diagonal elements of the covariance matrix are estimated using a gamma prior. Covariances are estimated based on a triangular decomposition.
- `"sv"`: Only the diagonal elements of the covariance matrix are estimated using a stochastic volatility algorithm. Off-diagonal elements are not estimated and set to zero.
- `"sv+covar"`: Only the diagonal elements of the covariance matrix are estimated using a stochastic volatility algorithm. Covariances are estimated based on a triangular decomposition.
- `"ald"`: The errors are assumed to follow an asymmetric Laplace distribution, which turns the model into a Bayesian quantile regression: the coefficients describe the conditional quantile specified in argument `quantile` instead of the conditional mean. Off-diagonal elements of the covariance matrix are not estimated and set to zero. See 'Details'.

Models with `error = "ald"` estimate a conditional quantile after Kozumi and Kobayashi (2011). Minimising the quantile loss at q corresponds to maximising the likelihood of an asymmetric Laplace distribution, which is a scale mixture of normal distributions. Conditional on the latent scales of that mixture every equation is a weighted normal regression, which is what makes the model a Gibbs sampler like the others.

Three properties of these models differ from the rest of the package. Covariances are not estimated, since rotating the equations into each other leaves a residual whose quantile is not the one that was asked for. A single quantile does not forecast, since the h step ahead quantile is not the quantile of the iterated one step ahead quantiles. And the asymmetric Laplace is a working likelihood rather than a claim about the data, so the posterior locates the quantile, but the spread of the draws is not a calibrated credible interval without the adjustment of Yang et al. (2016), which is not applied. Variable selection is available as `"bvs"`, not as `"ssvs"`.

With `quantile_grid = TRUE` the quantiles in `quantile`, at least two, form one model rather than one model each: the structural quantile VAR of Chavleishvili and Manganelli (2019). Every level is estimated – each is the chain its single-quantile model would have drawn – and the draws of `posterior$a` and `posterior$u_scale` are stacked level by level, the block of the first quantile first. With `structural = TRUE`, or a single variable, the grid describes the whole distribution of each variable given the ones ordered before it: the estimated quantiles are sorted (Chernozhukov et al., 2010), interpolated linearly between the levels and continued by exponential tails. That distribution is what `add_posterior_forecasts` simulates from, variable by variable, so a grid does forecast. A scenario given there pins variables in forecast periods, and a `forecast_quantile` takes every draw at one level, which gives the quantile paths the impulse responses of the model are differences of. `add_posterior_loglik` scores the density the grid describes. Only models with constant coefficients take a grid. Summaries, plots and impulse responses work on the single levels, which `split_quantile_grid` returns.

Available specifications for argument `varsel` are:

- `"none"`: No variable selection algorithm is used.
- `"bvs"`: Bayesian variable selection as proposed in Korobilis (2013).
- `"ssvs"`: Stochastic search variable selection as proposed in George et al. (2008).

The one specification for argument `algorithm` is `"discount"`, the discounted time varying parameter model of West & Harrison (1997, ch. 16) with the discounted Wishart of Uhlig (1997), estimated by `VarTvpDiscount`. It is not a sampler: its posterior is closed form – one pass over the sample, no chain – so `burnin` must be 0 and `thin` 1, and `iterations` says only how many i.i.d. draws a forecast takes from the answer. Its error covariance is the inverse Wishart whole, so `error` must be `"wishart"` and neither variable selection nor a structural model is available. What it buys is speed and an exact marginal likelihood: the sum of `/posterior/loglik` is the log marginal likelihood of the sample given the two discounts, so a grid over them can be compared without a chain being run for any of it.

`add_posterior_coefficients` estimates it like any other algorithm, the filter being part of the vendored BayesTS core. It consumes no random numbers, so two runs agree to the bit and a model estimated here and the same model estimated by the `bayests` command line over a file written with `write_to_hdf5` give the same numbers rather than merely the same distribution. What comes back is a posterior rather than a chain: one row per period under `posterior$a$mean`, `posterior$a$cov`, `posterior$u_sigma$scale` and `posterior$df`, and no `coeffs` anywhere, because joining one draw per period would look like a sampled path and is not one.

Argument `iid` restricts the equations of the variables it names to carry no coefficients at all. Such a variable is white noise: nothing forecasts it, it forecasts nothing, and what the model is estimated for is its contemporaneous correlation with the errors of the equations that do have dynamics. That is how a high-frequency surprise becomes a variable of a monthly VAR rather than an instrument beside one, following Jarocinski and Karadi (2020).

The restriction is exact rather than a tight prior on those coefficients: the sampler never draws them, so they are zero in every draw. Everything else treats the model as an ordinary VAR – the restricted variables are still regressors in the other equations, the error covariance still covers them, and `irf` and `fevd` read the draws unchanged.

The variables named in `iid` have to be the first columns of `data`, in any order among themselves. The restriction is carried by a variable's position rather than by a list of positions, which is what lets the sampler apply it without being told again which coefficients it owns. A data set in another order is refused with a message naming the columns that are there instead. It is available for models with constant coefficients and cannot be combined with `structural` or `varsel`.

With `missing = "estimate"` the panel does not have to be observed whole. Every observation becomes one linear constraint on the panel, stored in `data$train$constraints`, and every sweep of the sampler draws the periods that were not observed from their distribution given the constraints, the coefficients and the error covariance before it draws those (Chan, Poon and Zhu 2023). A series observed at a lower frequency is a constraint on several periods, the weights of which `aggregate` gives, so that a quarterly series enters a monthly model as the average, sum or growth rate of three months (Schorfheide and Song 2015). Which periods a constraint reaches is decided by the dates of `data`, and one that reaches before the estimation sample is left out with a message. The periods before the sample that the first lags reach are taken as observed, with gaps there filled by interpolation. What `data$train$y` holds where nothing was observed is a starting value: a linear interpolation of what was. The draws of the completed panel come back as `posterior$y`.

A series named in `soft` holds its constraints up to a normal error whose precision is estimated with a gamma prior, which `add_prior_options` sets. It suits an aggregate whose weights are an approximation, such as the growth rate of an average.

Estimating what was not observed is available for `error = "wishart"`, `"gamma"`, `"gamma+covar"`, `"sv"` and `"sv+covar"`, with constant or time varying coefficients. It cannot be combined with `structural` or the discounted model. It can be combined with `iid`, for models with constant coefficients: a variable named there that was not observed in a period – a high-frequency surprise whose series starts after the sample does, as in Jarocinski and Karadi (2020) – is drawn from its white noise given the errors of the other equations, and its gaps start at zero rather than at an interpolation. Forecasts start from each draw's completed panel. Forecasts are scored and conditioned on a scenario in `data$forecast$constraints` only by models with constant coefficients and `error = "wishart"` or `"gamma"`.

Value

An object of class 'bvarmodel' or, if a vector is given in `p`, `s` or `quantile`, a list of class 'modellist' with one such object per specification. A 'bvarmodel' is a list with the elements

`data` a list with element `original`, which holds the time-series objects `endogen`, `exogen` and `deterministic`, and element `train`, which holds the estimation sample: `y`, a $T \times K$ time-series object of the endogenous variables, `x`, a time-series object of the regressors, and `z`, the corresponding TK row matrix of regressors in SUR form, which is absent for the discounted model.

`model` a list of the specification, including `type` (`"VAR"`), `algorithm`, the name of the posterior simulation algorithm, `k`, `p`, `m`, `s` and `n`, the numbers of endogenous variables, lags, exogenous variables, their lags and deterministic terms, `endogen`, the names of the endogenous variables, and `deterministic`, `structural`, `error`, `varsel`, `tvp`, `iterations`, `burnin`, `thin` if it is above 1, and, for `error = "ald"`, `quantile` as specified.

The later steps of the workflow add the elements `priors`, `initial` and `posterior`.

References

Chan, J. C. C., Poon, A., & Zhu, D. (2023). High-dimensional conditionally Gaussian state space models with missing data. *Journal of Econometrics, 236*(1), 105468. doi:10.1016/j.jeconom.2023.05.005

Chan, J., Koop, G., Poirier, D. J., & Tobias, J. L. (2019). *Bayesian Econometric Methods* (2nd ed.). Cambridge: University Press.

Chavleishvili, S., & Manganelli, S. (2019). Forecasting and stress testing with quantile vector autoregression. *ECB Working Paper*, 2330.

Chernozhukov, V., Fernandez-Val, I., & Galichon, A. (2010). Quantile and probability curves without crossing. *Econometrica, 78*(3), 1093–1125.

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

Kozumi, H., & Kobayashi, G. (2011). Gibbs sampling methods for Bayesian quantile regression. *Journal of Statistical Computation and Simulation, 81*(11), 1565–1578. doi:10.1080/00949655.2010.496117

Schorfheide, F., & Song, D. (2015). Real-time forecasting with a mixed-frequency VAR. *Journal of Business & Economic Statistics, 33*(3), 366–380. doi:10.1080/07350015.2014.954707

Lütkepohl, H. (2006). *New Introduction to Multiple Time Series Analysis* (2nd ed.). Berlin: Springer.

Uhlig, H. (1997). Bayesian vector autoregressions with stochastic volatility. *Econometrica, 65*(1), 59–73. doi:10.2307/2171813

West, M., & Harrison, J. (1997). *Bayesian forecasting and dynamic models* (2nd ed.). New York: Springer.

See Also

`bvartools_model` describes the object this returns, element by element.

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvarmodel()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bvarmodel()`, `use_expanding_window.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
```

```
# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 50, burnin = 10)
# Number of iterations and burnin should be much higher.
```

`create_bvecmodel` *Create Vector Error Correction Models*

Description

Produces the input for the estimation of a vector error correction (VEC) model.

Usage

```
create_bvecmodel(
  data,
  p,
  exogen = NULL,
  s = NULL,
  r = NULL,
  const = NULL,
  trend = NULL,
  seasonal = NULL,
  structural = FALSE,
  error = "wishart",
  tvp = FALSE,
  varsel = "none",
  algorithm = NULL,
  delta_beta = 1,
  delta_sigma = 1,
  iterations = 20000,
  burnin = 2000,
  thin = 1
)
```

Arguments

`data`	a time-series object of endogenous variables.
`p`	an integer vector of the lag order of the endogenous variables in the corresponding VAR in levels, which must be at least 1. Following common convention, the resulting VEC model contains $p - 1$ lags of the differenced endogenous variables. There is no default. See 'Details'.
`exogen`	an optional time-series object of external regressors.
`s`	an integer vector of the lag order of the exogenous variables in the corresponding VAR in levels, which must be at least 1. The resulting VEC model contains the contemporaneous difference of the exogenous variables and $s - 1$ lags of it. Must be specified if `exogen` is given and is ignored otherwise. See 'Details'.

`r` an integer vector of the cointegration rank. See 'Details'.

`const` a character specifying whether a constant term enters the error correction term (`"restricted"`) or the non-cointegration term as an `"unrestricted"` variable. If `NULL` (default) no constant term will be added.

`trend` a character specifying whether a trend term enters the error correction term (`"restricted"`) or the non-cointegration term as an `"unrestricted"` variable. If `NULL` (default) no constant term will be added.

`seasonal` a character specifying whether seasonal dummies should be included in the error correction term (`"restricted"`) or in the non-cointegreation term as `"unrestricted"` variables. If `NULL` (default) no seasonal terms will be added. The amount of dummy variables will be automatically detected and depends on the frequency of the time-series object provided in `data`.

`structural` logical indicating whether data should be prepared for the estimation of a structural VEC model, whose lower triangular matrix A_0 multiplies the differences of the endogenous variables.

`error` character specifying the model that should be used for the estimation of the covariance matrix of the error term. Default is `"wishart"`. See 'Details'.

`tvp` logical indicating whether the model parameters are time varying.

`varsel` character specifying the type of variable selection algorithm that should be employed. Default is `"none"`. See 'Details'.

`algorithm` algorithm that should be used for posterior simulation. If `NULL` (default), standard algorithms will be used. See 'Details' for available non-standard options.

`delta_beta`, `delta_sigma`
numeric discount factors in $(0, 1]$ of the discounted model, the first governing the coefficients and the second the error covariance. Both default to 1, at which the quantity they govern does not move. Ignored unless `algorithm = "discount"`, and a vector in either produces one model per value. See 'Details'.

`iterations` an integer of MCMC draws excluding burn-in draws (defaults to 20000).

`burnin` an integer of MCMC draws used to initialize the sampler (defaults to 2000). These draws do not enter the computation of posterior moments, forecasts etc.

`thin` an integer thinning interval of the sampler (defaults to 1). After the burn-in the sampler keeps the last of every `thin` draws, so it runs `burnin + iterations * thin` draws and still keeps `iterations`. Unlike `thin`, which thins draws already made, the draws that are not kept are never held in memory.

Details

The function produces the variable matrices of vector error correction (VEC) models, which can also include exogenous variables:

$$\Delta y_t = \Pi w_t + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \sum_{i=0}^{s-1} \Upsilon_i \Delta x_{t-i} + C^{UR} d_t^{UR} + u_t,$$

where Δy_t is a $K \times 1$ vector of differenced endogenous variables, w_t is a $(K + M + N^R) \times 1$ vector of cointegration variables, Π is a $K \times (K + M + N^R)$ matrix of cointegration parameters,

Γ_i is a $K \times K$ coefficient matrix of endogenous variables, Δx_t is a $M \times 1$ vector of differenced exogenous regressors, Υ_i is a $K \times M$ coefficient matrix of exogenous regressors, d_t^{UR} is a $N^{UR} \times 1$ vector of deterministic terms, and C^{UR} is a $K \times N^{UR}$ coefficient matrix of deterministic terms that do not enter the cointegration term. p is the lag order of endogenous variables and s is the lag order of exogenous variables of the corresponding VAR model. u_t is a $K \times 1$ error term.

If an integer vector is provided as argument `p`, `s` or `r`, the function will produce a distinct model for all possible combinations of those specifications.

Argument `error` specifies the structure of the covariance matrix of the error term and how it is estimated. Possible specifications are:

- `"wishart"`: The covariance is estimated using a Wishart prior.
- `"gamma"`: Only the diagonal elements of the covariance matrix are estimated using a gamma prior. Off-diagonal elements are not estimated and set to zero.
- `"gamma+covar"`: The diagonal elements of the covariance matrix are estimated using a gamma prior. Covariances are estimated based on a triangular decomposition.
- `"sv"`: Only the diagonal elements of the covariance matrix are estimated using a stochastic volatility algorithm. Off-diagonal elements are not estimated and set to zero.
- `"sv+covar"`: Only the diagonal elements of the covariance matrix are estimated using a stochastic volatility algorithm. Covariances are estimated based on a triangular decomposition.

Available specifications for argument `varsel` are:

- `"none"`: No variable selection algorithm is used.
- `"bvs"`: Bayesian variable selection as proposed in Korobilis (2013).
- `"ssvs"`: Stochastic search variable selection as proposed in George et al. (2008).

Available specifications for argument `algorithm` are:

- `"KLGS2010"`: Algorithm proposed in Koop, León-González & Strachan (2010).
- `"discount"`: The discounted time varying parameter model of West & Harrison (1997, ch. 16) with the discounted Wishart of Uhlig (1997), estimated by `VecTvpDiscount`.

The discounted model is not a sampler. Its posterior is closed form – one pass over the sample, no chain – so `burnin` must be 0 and `thin` 1, and `iterations` says only how many i.i.d. draws a forecast takes from the answer. Its error covariance is the inverse Wishart whole, so `error` must be `"wishart"` and neither variable selection nor a structural model is available. What it buys is speed and an exact marginal likelihood: the sum of `/posterior/loglik` is the log marginal likelihood of the sample given the rank, the cointegration matrix and the two discounts, so a grid over any of them can be compared without a chain being run for any of it.

The one assumption that separates it from the sampling VEC models is that the cointegration space is given rather than estimated. `add_initial_values` puts Johansen's maximum likelihood estimate at `/initial/beta` and the model conditions on it, so what drifts is the adjustment to the long-run relations and not the relations themselves. That is a different question from the one `"KLGS2010"` answers, not a cheaper way of answering the same one.

`add_posterior_coefficients` estimates it like any other algorithm, the filter being part of the vendored BayesTS core. It consumes no random numbers, so two runs agree to the bit and a model

estimated here and the same model estimated by the `bayests` command line over a file written with `write_to_hdf5` give the same numbers rather than merely the same distribution. What comes back is a posterior rather than a chain: one row per period under `posterior$a$mean`, `posterior$a$cov`, `posterior$u_sigma$scale` and `posterior$df`, and no `coeffs` anywhere, because joining one draw per period would look like a sampled path and is not one.

Value

An object of class 'bvecmodel' or, if a vector is given in `p`, `s` or `r`, a list of class 'modellist' with one such object per specification. A 'bvecmodel' is a list with the elements

`data` a list with element `original`, which holds the time-series objects `endogen`, `exogen` and `deterministic` in levels, and element `train`, which holds the estimation sample: `y`, a $T \times K$ time-series object of the differenced endogenous variables, `w`, the lagged levels that enter the cointegration term, `x`, the remaining regressors, and `z`, the corresponding TK row matrix of regressors in SUR form, which is absent for the discounted model.

`model` a list of the specification with the same elements as that of a 'bvarmodel', where `type` is `"VEC"` and `p` is the lag order of the VAR in levels, and additionally `rank`, the cointegration rank, `k_beta`, the number of variables in the cointegration term, and `n_restricted`, the number of deterministic terms restricted to it.

The later steps of the workflow add the elements `priors`, `initial` and `posterior`.

References

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224–242. doi:10.1080/07474930903382208

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Uhlig, H. (1997). Bayesian vector autoregressions with stochastic volatility. *Econometrica, 65*(1), 59–73. doi:10.2307/2171813

West, M., & Harrison, J. (1997). *Bayesian forecasting and dynamic models* (2nd ed.). New York: Springer.

See Also

`bvartools_model` describes the object this returns, element by element.

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvarmodel()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `transform_variables()`, `use_expanding_window.bvarmodel()`, `use_expanding_window.bvecmodel()`

Examples

```
# Load data
data("e6")

# Create model
model <- create_bvecmodel(e6, p = 4, r = 1,
                          const = "unrestricted", seasonal = "unrestricted",
                          iterations = 10, burnin = 10)
# Number of iterations and burn-in should be much higher.
```

---

create_external_forecast

Objects for Externally Produced Forecasts

Description

Turns externally produced point forecasts into an object, which can be evaluated and compared with the forecasts of Bayesian VAR and VEC models.

Usage

```
create_external_forecast(
  forecasts,
  object,
  n_ahead = NULL,
  period = "period",
  origin = "origin",
  variable = "variable",
  value = "value",
  by = NULL,
  data_lag = 1,
  select = "last"
)

## S3 method for class 'externalforecast'
add_priors(object, ...)

## S3 method for class 'externalforecast'
add_initial_values(object, ...)

## S3 method for class 'externalforecast'
add_seed(object, seed, ...)

## S3 method for class 'externalforecast'
add_posterior_coefficients(object, ...)
```

```
## S3 method for class 'externalforecast'
add_posterior_loglik(object, ...)

## S3 method for class 'externalforecast'
add_forecast_input(object, ...)

## S3 method for class 'externalforecast'
add_posterior_forecasts(object, ...)

## S3 method for class 'externalforecast'
minnesota_prior(object, ...)

## S3 method for class 'externalforecast'
ssvs_prior(object, ...)

## S3 method for class 'externalforecast'
inclusion_prior(object, ...)

## S3 method for class 'externalforecast'
thin(x, thin = 10, ...)

## S3 method for class 'externalforecast'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

forecasts	a data frame in long format, which contains the external point forecasts. See 'Details' for the required columns.
object	for create_external_forecast a model object of class 'bvarmodel', 'bvecmodel', 'expandingwindow' or 'modellist', which is used as the reference of the comparison. It provides the endogenous variables, the frequency of the data and the ends of the training samples, to which the external forecasts are matched. For the methods of the estimation workflow an object of class 'externalforecast'.
n_ahead	the maximum forecast horizon that is considered. If NULL (default), the forecast horizon of the models in object is used, which requires that add_forecast_input was already applied to them. For models whose forecasts were aggregated with aggregate_forecasts it is a number of years.
period	name of the column of forecasts, which contains the period, for which a forecast was made.
origin	name of the column of forecasts, which contains the period, in which a forecast was published.
variable	name of the column of forecasts, which contains the names of the forecasted variables. They must correspond to the names of the endogenous variables of the models in object.
value	name of the column of forecasts, which contains the values of the forecasts.
by	name of an optional column of forecasts, which contains the names of the forecasters. If specified, one object is produced per forecaster and the result is a

	list of class 'modellist'.
data_lag	an integer specifying the number of periods, by which the publication of the data of the endogenous variables lags behind. See 'Details'.
select	either "last" (default) or "first" specifying which forecast is used, if multiple publications are matched to the same training sample. See 'Details'.
...	arguments passed forward to method.
seed	not used, since external forecasts are not simulated.
x	an object of class 'externalforecast'.
thin	an integer specifying the thinning interval between successive draws.
digits	the number of significant digits.

Details

Argument forecasts must be a data frame in long format, where each row contains a single point forecast. The names of the required columns can be specified in the arguments period, origin, variable and value. Periods can be provided either as objects of class 'Date' or as numerics, which follow the convention of time, i.e. 2007.25 for the second quarter of 2007. The periods, for which a forecast was made, are rounded to the frequency of the data of the models in object.

Rounding places a forecast in the period of the data it falls into; it does not convert between frequencies. Forecasts made at a frequency other than that of the data are therefore refused: annual forecasts of a quarterly model, whose periods such as 2020 and 2021 would otherwise be read as the first quarters of those years and an annual growth rate scored as a quarterly one, or monthly forecasts of a quarterly model, of which three would fall into the same quarter. The frequency of the forecasts is read off the spacing of the periods within a publication and, where every publication forecasts a single period, off the position of the periods within the year.

Annual forecasts are compared with models estimated on quarterly or monthly data by aggregating the forecasts of the models to annual figures with aggregate_forecasts first and passing the result as argument object. The periods of the forecasts are then read as years, while publications are still matched to the training samples at the frequency of the data, and data_lag still counts periods of the data. Horizon 1 is the year of the first period after the training sample, so that a forecast for the year of its publication is a forecast of horizon 1 and one for the year after it of horizon 2, whichever quarter it was published in. The realised values, against which the forecasts are scored, are the annual figures of the data of the models, which the aggregated forecasts of the models are scored against, too, so that add_forecast_errors does not need argument test_sample.

In contrast to a model, an external forecast does not have a training sample. Therefore, each publication is matched to the training sample, which ends closest before the publication of the forecast, so that a model and an external forecaster are evaluated on comparable information sets. Since the data of the endogenous variables are usually published with a delay, argument data_lag can be used to specify the number of periods, by which the last available observation lags behind the publication of a forecast. Thus, a publication is matched to the last training sample, which does not end after origin - data_lag periods. With the default of one period an annual forecast, which was published in the course of 2021, is matched to a training sample that ends in 2020, so that the forecast for 2021 is a one-step ahead forecast.

If multiple publications are matched to the same training sample, only one of them is used, because otherwise a forecaster would enter the comparison multiple times for the same period. Argument select controls whether the latest ("last") or the earliest ("first") of those publications is used.

The resulting object mimics the structure of a model, which was estimated with the expanding window approach of use_expanding_window, where each publication corresponds to one estimation window. It can be added to a list of models with combine_models and the functions, which are applied to obtain posterior draws, are without effect for it. Since external forecasts are point forecasts, their forecast errors consist of a single draw. Accordingly, the credible bands of the out-of-sample statistics in selection_criteria are degenerate and in-sample criteria are not available.

External forecasts do not have to be estimated, so the functions, which add priors, initial values or posterior draws to a model are without effect for objects of class 'externalforecast'. This allows to combine them with model objects in a list of class 'modellist' and to apply the usual workflow to all elements of that list.

Value

A list of class 'externalforecast' or, if argument by is specified, a list of class 'modellist', which contains one object of class 'externalforecast' per forecaster.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("us_macrodata")

# Create model
model <- create_bvarmodel(data = us_macrodata, p = 1, deterministic = "none",
                          error = "gamma", iterations = 10, burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.

model <- use_expanding_window(model, start = 2007)

# Artificial external forecasts of two forecasters
fcst <- expand.grid(origin = c(2007, 2007.25),
                    h = 1:2,
                    variable = c("Dp", "r"),
                    forecaster = c("A", "B"),
                    stringsAsFactors = FALSE)
fcst[["period"]] <- fcst[["origin"]] + fcst[["h"]] / 4
fcst[["value"]] <- 0

# Create objects of the external forecasts, where the data of the endogenous
# variables are assumed to be published with a delay of one quarter
ext <- create_external_forecast(fcst, model, n_ahead = 4,
                                by = "forecaster", data_lag = 1)

# Calculate forecast errors
```

```
ext <- add_forecast_errors(ext, test_sample = us_macrodata)

# Compare the forecast performance
selection_criteria(ext)
```

`discount_models` *The Discounted Models*

Description

Whether a model is one of the two discounted models, and what a discounted model cannot be. Both are exported for the packages that build a model of their own on this one and have to make the same distinction.

Usage

```
is_discount_model(x)

check_discount_specification(
  k,
  error = "wishart",
  varsel = "none",
  structural = FALSE,
  burnin = 0,
  thin = 1,
  delta_beta = 1,
  delta_sigma = 1
)
```

Arguments

`x`	a model, a specification list – the `model` element of one – or the name of an algorithm.
`k`	the number of endogenous variables.
`error`, `varsel`, `structural`, `burnin`, `thin`	the corresponding arguments of `create_bvarmodel` or `create_bvecmodel`.
`delta_beta`, `delta_sigma`	the two discount factors.

Details

`VarTvpDiscount` and `VecTvpDiscount` are the matrix normal dynamic linear model of West & Harrison (1997, ch. 16) with the discounted Wishart of Uhlig (1997). Their posterior is closed form – one pass over the sample, no chain, no random numbers consumed – which is what every refusal below follows from. See `create_bvecmodel` for what they are and how they are set up.

`check_discount_specification` raises an error for a specification a discounted model cannot carry and returns nothing otherwise. Each refusal is one BayesTS would raise as well, against a file that had already been written; raising it here means one error for a grid of models rather than one per file.

Value

`is_discount_model` returns a single logical. `check_discount_specification` returns `NULL` invisibly.

References

Uhlig, H. (1997). Bayesian vector autoregressions with stochastic volatility. *Econometrica, 65*(1), 59–73. doi:10.2307/2171813

West, M., & Harrison, J. (1997). *Bayesian forecasting and dynamic models* (2nd ed.). New York: Springer.

Examples

```
data("e6")
model <- create_bvecmodel(e6 * 100, p = 2, r = 1, const = "unrestricted",
                          algorithm = "discount", delta_beta = 0.98,
                          iterations = 10, burnin = 0, thin = 1)
is_discount_model(model)
is_discount_model("VecNormalWishart")
```

---

e1 *West German economic time series data*

Description

The data set contains quarterly, seasonally adjusted time series for West German fixed investment, disposable income, and consumption expenditures in billions of DM from 1960Q1 to 1982Q4. It was produced from file E1 of the data sets associated with Lütkepohl (2006). Raw data are available at http://www.jmulti.de/download/datasets/e1.dat and were originally obtained from Deutsche Bundesbank.

Usage

```
data("e1")
```

Format

A named time-series object with 92 rows and 3 variables:

invest fixed investment.

income disposable income.

cons consumption expenditures.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

`e6` *German interest and inflation rate data*

Description

The data set contains quarterly, seasonally unadjusted time series for German long-term interest and inflation rates from 1972Q2 to 1998Q4. It was produced from file E6 of the data sets associated with Lütkepohl (2006). Raw data are available at http://www.jmulti.de/download/datasets/e6.dat and were originally obtained from Deutsche Bundesbank and Deutsches Institut für Wirtschaftsforschung.

Usage

```
data("e6")
```

Format

A named time-series object with 107 rows and 2 variables:

R nominal long-term interest rate (Umlaufsrendite).

Dp Δ log of GDP deflator.

Details

The data cover West Germany until 1990Q2 and all of Germany aferwards. The values refer to the last month of a quarter.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

`expected_model_size` *Expected Size of a Model*

Description

A generic function that calculates how much memory a model will take once its posterior draws are complete, and so roughly how much disk space it will take when written to a file.

Usage

```
expected_model_size(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Details

The functions that draw or write a model warn before they start when the result would exceed the limit set by `options(bvartools.size_warning)`, in bytes, which is 1e9, one gigabyte, unless set otherwise, and `Inf` turns the warnings off. `add_posterior_coefficients` checks the size `expected_model_size()` predicts for the draws it is about to simulate, and `write_to_hdf5` checks the size of the object it is about to write. A package that adds a model class gets both warnings by adding a method to this generic.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `expected_model_size.bvarmodel`, `expected_model_size.bvecmodel`, `expected_model_size.expandingwi`
`expected_model_size.modellist`.

`expected_model_size.bvarmodel` *Expected Size of a Model*

Description

Calculates how much memory a model will take once its posterior draws are complete, block by block, from its specification alone, so that it can be called before a long estimation is started.

Usage

```
## S3 method for class 'bvarmodel'
expected_model_size(object, chains = NULL, ...)

## S3 method for class 'bvecmodel'
expected_model_size(object, chains = NULL, ...)

## S3 method for class 'expandingwindow'
expected_model_size(object, ...)

## S3 method for class 'modellist'
expected_model_size(object, ...)
```

Arguments

`object`	an object of class 'bvarmodel' or 'bvecmodel', usually a result of a call to `create_bvarmodel` or `create_bvecmodel`, or a list of such models of class 'modellist' or 'expandingwindow'.
`chains`	the number of chains that will be simulated. If `NULL` (default), the value in `object$model$chains` is used, and one chain when there is none. See `add_posterior_coefficients`.
`...`	further arguments passed to or from other methods.

Details

Every block of posterior draws is a matrix of numbers with one row per kept draw, and each number takes eight bytes. How many draws are kept is set by `iterations`, how many columns each block has by the specification: the number of coefficients, whether they vary over time, the error distribution, variable selection and the rank of a VEC model. **Time-varying parameters and stochastic volatility multiply the size of a block by the number of periods in the sample**, which is how a model that is small as a constant coefficient model can need gigabytes.

A few blocks depend on the priors: the inclusion indicators of the covariance block, the draws of a non-centred prior (`omega_v`) and a drawn autocorrelation of the cointegration space (`rho_min`). They are counted once `add_priors` has added the priors. Forecast draws are counted once `add_forecast_input` has set a horizon.

The numbers are what the object will hold in memory. R may briefly hold a second copy while a step modifies it, so the memory a session needs can be up to twice as large. `write_to_hdf5` compresses what it writes, but posterior draws compress poorly, so the size of the file is usually a little below the size in memory.

Value

A data frame of class 'modelsize' with one row per block of the object and the columns

`model` for a list of models, the name or position of the model.

`element` where the block will be in the object, such as `posterior$a$coeffs`. The first row of each model is everything that is there already, the data, the priors and the initial values.

`step` the function that adds the block, empty for the first row. The warning of `add_posterior_coefficients` counts the first row and the blocks that function adds itself.

`draws`, `columns` the dimensions of the block.

`bytes` the size of the block in bytes.

`sum(result$bytes)` is the expected size of the whole object, and the print method shows it by step.

See Also

`expected_model_size` for the warnings that are based on it, and `bvartools_model` for the blocks of the object.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# A model with constant coefficients and one with time-varying parameters
# and stochastic volatility
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 5000, burnin = 1000)
expected_model_size(model)

model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          tvp = TRUE, error = "sv",
                          iterations = 5000, burnin = 1000)
expected_model_size(model)

# The total in bytes
sum(expected_model_size(model)$bytes)
```

`fevd` *Forecast Error Variance Decomposition*

Description

A generic function used to calculate forecast error variance decompositions.

Usage

```
fevd(x, ...)
```

Arguments

`x`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `x`, as described on the pages of the methods.

See Also

Methods: `fevd.bvarmodel`.

`fevd.bvarmodel` *Forecast Error Variance Decomposition*

Description

Produces the forecast error variance decomposition for an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
fevd(
  x,
  response = NULL,
  n_ahead = 5,
  type = "oir",
  normalise_gir = FALSE,
  period = NULL,
  max_groups = NULL,
  impact = NULL,
  statistic = "mean",
  ci = NULL,
  ...
)
```

Arguments

`x`	an object of class 'bvarmodel'.
`response`	name of the response variable.
`n_ahead`	number of steps ahead. Zero is allowed and returns the decomposition on impact alone.
`type`	type of the impulse responses used to calculate forecast error variable decompositions. Possible choices are orthogonalised `"oir"` (default), structural `"sir"`, generalised `"gir"`, structural generalised `"sgir"`, sign restricted `"sign"` and `"custom"` impulse responses. For a structural model only `"sir"` and `"sgir"` are available; the other four require a non-structural model. See 'Details'.
`normalise_gir`	logical. Should the GIR-based FEVD be normalised?
`period`	integer. Index of the period, for which the variance decomposition should be generated. Only used for TVP or SV models. Default is `NULL`, so that the posterior draws of the last time period are used. With `type = "sign"` the default is the period the restrictions were imposed in, and another period is refused, since the rotations do not identify it.
`max_groups`	integer. Maximum number of variables the decomposition should contain. The `max_groups - 1` variables with the largest contributions across the whole horizon are kept and the contributions of the remaining variables are added up in a further column named `"Other"`. This keeps the legend of the corresponding

	plot readable for models with many variables. Default is NULL, so that a column is returned for every variable.
impact	the impact matrix of a "custom" decomposition, either a single $K \times K$ matrix that identifies every posterior draw the same way, or a list of such matrices with one entry per draw. Ignored for every other value of type. See 'Details'.
statistic	the posterior summary of each share, "mean" (default) or "median".
ci	an optional probability, the coverage of a credible interval returned alongside, such as 0.68 for the 16th and 84th percentiles.
...	further arguments passed to or from other methods.

Details

The function produces forecast error variance decompositions (FEVD) for the VAR model

$$A_0 y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t,$$

with $u_t \sim N(0, \Sigma)$. For non-structural models matrix A_0 is set to the identiy matrix and can therefore be omitted, where not relevant.

For a structural model the posterior draws describe the structural form, so only "sir" and "sgir", which invert A_0, recover the reduced form the recursion for Φ_i needs. The other types are therefore not available for a structural model, and the two structural types not for any other.

If the FEVD is based on the orthogonalised impulse resonse (OIR), the FEVD will be calculated as

$$\omega_{jk,h}^{OIR} = \frac{\sum_{i=0}^{h}(e_j' \Phi_i P e_k)^2}{\sum_{i=0}^{h}(e_j' \Phi_i \Sigma \Phi_i' e_j)},$$

where Φ_i is the forecast error impulse response for the ith period, P is the lower triangular Choleski decomposition of the variance-covariance matrix Σ, e_j is a selection vector for the response variable and e_k a selection vector for the impulse variable.

If type = "sign", the decomposition uses PQ, with Q the rotation that add_sign_restrictions accepted for that draw, and is otherwise the orthogonalised case. A rotation of the Choleski factor still factorises Σ, so the shares add up as they do there. Draws that no admissible rotation was found for are left out.

If type = "custom", the decomposition uses the matrix P supplied in argument impact in place of the Choleski factor, while the denominator stays the one of the orthogonalised case: an impact matrix relabels the shocks but says nothing about the model, so the forecast error variance being decomposed is still that of Σ. The shares therefore add up to one across shocks exactly when $PP' = \Sigma$, which holds for a rotation of the Choleski factor and need not hold for an arbitrary matrix. This is not enforced: a decomposition that does not sum to one is the honest report of an impact matrix that does not factorise Σ.

If type = "sir", the structural FEVD will be calculated as

$$\omega_{jk,h}^{SIR} = \frac{\sum_{i=0}^{h}(e_j' \Phi_i A_0^{-1} P e_k)^2}{\sum_{i=0}^{h}(e_j' \Phi_i A_0^{-1} \Sigma A_0^{-1\prime} \Phi_i' e_j)},$$

where P is the lower triangular Choleski decomposition of Σ. Since Σ is the covariance matrix of the structural errors, the decomposition weighs each structural shock by its own variance.

If `type = "gir"`, the generalised FEVD of Pesaran and Shin (1998) will be calculated as

$$\omega_{jk,h}^{GIR} = \frac{\sigma_{kk}^{-1} \sum_{i=0}^{h} (e_j' \Phi_i \Sigma e_k)^2}{\sum_{i=0}^{h} (e_j' \Phi_i \Sigma \Phi_i' e_j)},$$

where σ_{kk} is the variance of the error of the *impulse* variable k, the diagonal element of Σ that belongs to the shock. This is the decomposition that `spillover` is built on.

If `type = "sgir"`, the structural generalised FEVD will be calculated as

$$\omega_{jk,h}^{SGIR} = \frac{\sigma_{kk}^{-1} \sum_{i=0}^{h} (e_j' \Phi_i A_0^{-1} \Sigma e_k)^2}{\sum_{i=0}^{h} (e_j' \Phi_i A_0^{-1} \Sigma A_0^{-1\prime} \Phi_i' e_j)},$$

where Σ is the covariance matrix of the structural errors and σ_{kk} the variance of the structural shock k.

Since GIR-based FEVDs do not add up to unity, they can be normalised by setting `normalise_gir = TRUE`, which divides each row by its sum after the decomposition has been averaged over the posterior draws. `spillover` normalises each draw before averaging, so its shares differ from these by the difference between a ratio of means and a mean of ratios.

The row of period h sums the impulse responses of periods 0 to h, so it is the decomposition of the $h + 1$ step forecast error variance. Period 0 is the decomposition on impact.

Value

A time-series object of class 'bvarfevd' running from period 0 to `n_ahead`, with one column per variable holding the share of the forecast error variance of `response` that is due to its shocks. For `type = "oir"` the rows sum to one; for `type = "gir"` they do so only with `normalise_gir = TRUE`. With `ci`, attributes `lower` and `upper` hold the bounds of the credible interval of each share in the same shape. With `statistic = "median"` or `ci`, a GIR-based decomposition is normalised draw by draw rather than after averaging, and the medians need not sum to one. `ci` cannot be combined with `max_groups`, since the bounds of a pooled column are not the pooled bounds.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Pesaran, H. H., & Shin, Y. (1998). Generalized impulse response analysis in linear multivariate models. *Economics Letters, 58*, 17-29.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Obtain FEVD
vd <- fevd(object, response = "cons")

# Plot FEVD
plot(vd)
```

---

fevd.bvecmodel *Forecast Error Variance Decomposition*

Description

Guard method for objects of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
fevd(x, ...)
```

Arguments

x	an object of class 'bvecmodel'.
...	additional arguments.

Value

Nothing. The method raises an error, since variance decompositions of a VEC model are obtained from its VAR representation: apply vec_to_var first and then fevd to the resulting 'bvarmodel'.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

---

folder_steps *Estimation Steps on a Folder of Models*

Description

The steps of an estimation, applied to the models of a folder one at a time rather than to a list of models in the session. Each reads a model, changes it, writes it back and drops it, so a step costs one model per worker.

Usage

```
## S3 method for class 'bvarfolder'
add_priors(object, ..., cores = 1)

## S3 method for class 'bvarfolder'
add_initial_values(object, ..., cores = 1)

## S3 method for class 'bvarfolder'
add_seed(object, seed, ..., cores = 1)

## S3 method for class 'bvarfolder'
add_posterior_coefficients(object, ..., cores = 1, export = NULL)

## S3 method for class 'bvarfolder'
add_posterior_loglik(object, ..., cores = 1)

## S3 method for class 'bvarfolder'
thin(x, ..., cores = 1)

## S3 method for class 'bvarfolder'
selection_criteria(object, ..., cores = 1)

## S3 method for class 'bvarfolder'
time_variation_test(object, ..., cores = 1)
```

Arguments

object	an object of class 'bvarfolder', from open_models.
...	further arguments passed to the method for a single model.

cores	the number of worker processes. Defaults to one.
seed	an integer, the seed of the first model. The models are numbered through in the order of the manifest, as the models of a list are.
export	character vector of names of objects the workers need, for instance the sampler that posterior_function refers to. See map_models.
x	an object of class 'bvarfolder'.

Details

The methods do what their counterparts for a 'bvarmodel' or a 'bvecmodel' do, model by model, and they take the same arguments.

Each model is drawn with a seed of its own, so the draws do not depend on how many workers a step runs on, nor on whether the models were estimated in one run or in several. The seeds are the ones a list of models gets from add_seed: seed, seed + 1, and so on in the order of the manifest.

A step takes models, which map_models understands: a character vector of the models it is applied to, the rest of the folder being left as it is. An estimation of many expensive models is run in parts that way, and one that was interrupted is picked up where it stopped.

add_posterior_coefficients is the step to think twice about. It writes the largest thing a model has, and a cluster of R workers carries every draw into the session and out again to do it. Where the sampler is the BayesTS executable, bayests_files runs it on the files instead:

```
run <- bayests_files(executable = "/opt/bayests/bin/bayests")
run(model_files(stored), jobs = 6)
```

which gives the same draws, since a model is drawn with the seed in its file.

selection_criteria and time_variation_test are the methods that read rather than write: they hand back a result per model and leave the folder as it is. The first is what choose_best_model compares, so a grid of specifications too large to hold is still chosen from; the second gives the Bayes factors for time variation of every model of the folder, which is how a grid of time varying models is asked whether its coefficients move at all.

Value

The object in object, with its manifest brought up to date.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("e1")
train <- diff(log(e1)) * 100

models <- create_bvarmodel(data = train, p = 1:2, deterministic = "const",
                           iterations = 20, burnin = 10)

folder <- file.path(tempdir(), "bvartools-example-steps")
unlink(folder, recursive = TRUE)
dir.create(folder, recursive = TRUE)
write_to_hdf5(models, folder = folder)

stored <- open_models(folder)
stored <- add_priors(stored, coef = list(v_i = 1),
                     sigma = list(df = 3, scale = 1))
stored <- add_initial_values(stored)
stored <- add_seed(stored, 20260919)
stored <- add_posterior_coefficients(stored)
stored <- add_posterior_loglik(stored)
```

---

from_bayests_tree *Turn the Tree of a Model File into a Model*

Description

The step of `read_model_from_hdf5` that turns what `read_bayests_tree` reads into the object of a model package.

Usage

```
from_bayests_tree(tree, ...)

## Default S3 method:
from_bayests_tree(tree, ...)
```

Arguments

`tree`	the tree of a model file, as `read_bayests_tree` returns it, with the class recorded in the file's `/model/rclass`.
`...`	further arguments passed to or from other methods.

Details

`read_model_from_hdf5` reads the VAR and VEC models of this package itself. For a file whose `rclass` names any other class it reads the tree, gives it that class and calls this generic, so a package that writes its models with `write_bayests_tree` reads them back by registering a method for its

class. The method undoes the translation its writer made: names, orderings and shapes that BayesTS wants and the package's object does not.

The package named in the file's `/model/rpackage` is loaded first, if it is installed, so that its methods are registered whether or not it is attached. A file of the BayesTS command line carries no such attribute; one of a factor model is read with **dfmtools** loaded.

Without a method the default refuses, and names the class it found no method for.

A method keeps the attributes of `/model` it does not know in element `model` of what it returns. `write_to_hdf5` records there where a model stood in the 'modellist' it was written from, and `read_models_from_folder` rebuilds the list from it.

Value

An object of the class of `tree`, as its package defines it.

`generate_artificial_var`

Generate Artificial VAR Data

Description

Generates an artificial data set from a vector autoregressive model with constant or time varying parameters and constant or stochastic volatility, for testing the algorithms of the package.

Usage

```
generate_artificial_var(
  nobs = 100,
  k = 3,
  p = 2,
  deterministic = "none",
  structural = FALSE,
  tvp = FALSE,
  sv = FALSE,
  range_a = c(-0.5, 0.5),
  a_zeros = 0.5,
  range_a0 = c(-0.5, 0.5),
  range_const = c(-0.5, 0.5),
  range_trend = c(-0.01, 0.01),
  range_variance = c(1, 1),
  range_psi = c(0, 0),
  range_variance_state = c(1e-04, 1e-04),
  range_variance_sv = c(0.01, 0.01),
  stable = TRUE,
  presample = 100,
  level = 0
)
```

Arguments

nobs	number of generated observations. Defaults to 100.
k	number of endogenous variables. Defaults to 3.
p	number of lags of the VAR model. Can be zero. Defaults to 2.
deterministic	a character specifying which deterministic terms the model contains: "none" (default), "const" for an intercept, "trend" for a linear trend, and "both" for an intercept with a linear trend. The terms are the same as in create_bvarmodel.
structural	logical specifying whether a structural VAR model with a lower triangular matrix A_0 of contemporaneous coefficients is generated. Defaults to FALSE. See 'Details'.
tvp	logical specifying whether the coefficients, A_0 and the elements of Ψ follow random walks. Defaults to FALSE. See 'Details'.
sv	logical specifying whether the error variances follow a stochastic volatility process. Defaults to FALSE. See 'Details'.
range_a	numeric vector with two elements containing the minimum and maximum value of the coefficients of lagged endogenous variables. Defaults to c(-0.5, 0.5).
a_zeros	numeric between 0 and 1 indicating the share of coefficients of lagged endogenous variables, which should be set to zero. Default is 0.5.
range_a0	numeric vector with two elements containing the minimum and maximum value of the free elements of A_0. Only used if structural = TRUE. Defaults to c(-0.5, 0.5).
range_const	numeric vector with two elements containing the minimum and maximum value of the intercept terms. Defaults to c(-0.5, 0.5).
range_trend	numeric vector with two elements containing the minimum and maximum value of the coefficients of the linear trend. Defaults to c(-0.01, 0.01).
range_variance	numeric vector with two elements containing the minimum and maximum value of the error variances. For sv = TRUE these are the variances in the first period. Defaults to c(1, 1).
range_psi	numeric vector with two elements containing the minimum and maximum value of the free elements of Ψ. Defaults to c(0, 0), which gives uncorrelated errors. Must be c(0, 0) for structural models.
range_variance_state	numeric vector with two elements containing the minimum and maximum value of the variances of the state equations of the coefficients, of A_0 and of Ψ. Only used if tvp = TRUE. Defaults to c(0.0001, 0.0001).
range_variance_sv	numeric vector with two elements containing the minimum and maximum value of the variances of the state equations of the log-volatilities. Only used if sv = TRUE. Defaults to c(0.01, 0.01).
stable	logical specifying whether the coefficients of lagged endogenous variables are restricted to a stable VAR process. Defaults to TRUE. See 'Details'.
presample	numeric specifying the number of observations, which are generated before the first returned observation and then discarded, so that the series do not depend on their initial values. Defaults to 100.

`level` numeric vector with one or k elements, which are added to the generated series to shift their levels. Defaults to 0. A non-zero level requires an intercept. See 'Details'.

Details

The function produces artificial observations for a vector autoregressive (VAR) model:

$$A_{0t} y_t = \sum_{i=1}^{p} A_{it} y_{t-i} + C_t d_t + u_t,$$

where y_t is a K-dimensional vector of endogenous variables, A_{0t} is a $K \times K$ matrix of contemporaneous coefficients, A_{it} is a $K \times K$ coefficient matrix of endogenous variables, d_t is a vector of deterministic terms and C_t its coefficient matrix. p is the lag order of endogenous variables.

As in Primiceri (2005) u_t is an error term with $u_t \sim N(0, \Sigma_t)$ and $\Psi_t \Sigma_t \Psi_t' = \Omega_t$, where Ψ_t is a lower triangular matrix with ones on the main diagonal and Ω_t is a diagonal matrix with error variances $\omega_{1t}, \ldots, \omega_{Kt}$.

Unless `structural = TRUE`, A_{0t} is an identity matrix. Otherwise it is a lower triangular matrix with ones on its main diagonal, whose free elements are drawn from `range_a0`, and the errors of the structural equations are uncorrelated, so that Ψ_t is an identity matrix and $\Sigma_t = \Omega_t$. This is the model that `create_bvarmodel` produces with `structural = TRUE`. The coefficients A_{it} and C_t are those of the structural equations, and the reduced form of the model has the coefficients $A_{0t}^{-1} A_{it}$ and the error covariance matrix $A_{0t}^{-1} \Omega_t A_{0t}^{-1\prime}$.

The linear trend takes the value 1 in period $p + 1$ of the returned series, which is the first period that `create_bvarmodel` uses for estimation with lag order p. The true coefficients of the trend are therefore those that model estimates with the same lag order refer to.

If `tvp = FALSE` and `sv = FALSE`, all parameters are constant over time. If `tvp = TRUE`, the coefficients $a_t = vec(A_{1t}, \ldots, A_{pt}, C_t)$, the free elements of A_{0t} and the free elements ψ_t of Ψ_t follow random walks

$$a_t = a_{t-1} + v_t, \quad v_t \sim N(0, Q_a), \qquad \psi_t = \psi_{t-1} + w_t, \quad w_t \sim N(0, Q_\psi),$$

with diagonal Q_a and Q_ψ, whose elements are drawn from `range_variance_state`, and likewise for A_{0t}. Coefficients that are zero in the first period, such as those set to zero by `a_zeros`, have a state variance of zero and stay zero. With the default `range_psi = c(0, 0)` the errors therefore remain uncorrelated. If `sv = TRUE`, the log-volatilities follow random walks

$$\ln \omega_{it} = \ln \omega_{i,t-1} + \eta_{it}, \quad \eta_{it} \sim N(0, \sigma_i^2),$$

with σ_i^2 drawn from `range_variance_sv`. These are the state equations of the models that `create_bvarmodel` produces with `tvp = TRUE` and `error = "sv"` or `"sv+covar"`.

The parameters of the first period are drawn from the ranges above. During the presample they are held at these values, so that the returned paths start at them and the presample only removes the influence of the initial values of the series.

If `stable = TRUE`, the coefficients of lagged endogenous variables are drawn until the companion matrix of the reduced form of the process has no eigenvalue on or outside the unit circle. For `tvp = TRUE` this holds in every period: an innovation of the coefficients, which would make the process unstable, is drawn again, and if no stable innovation is found in 100 attempts, the coefficients keep

the values of the previous period. This restricts the random walks to the stable region as in Cogley and Sargent (2005). Set `stable = FALSE` to generate integrated or explosive series, usually together with `presample = 0`. Cointegrated series are generated by `generate_artificial_vec`.

Argument `level` shifts the generated series by a vector m, which gives them high levels without changing their dynamics. The model becomes $A_{0t}(y_t - m) = \sum_{i=1}^{p} A_{it}(y_{t-i} - m) + C_t d_t + u_t$, so that the intercept absorbs the shift: the returned intercept is $c_t + (A_{0t} - \sum_{i=1}^{p} A_{it})m$, where c_t is drawn from `range_const`. Without an intercept this term would not be part of the model, so that a non-zero level requires `deterministic = "const"` or `"both"`.

Value

A list with the elements

`data` a $T \times K$ time-series object of the artificial series, named `var1`, `var2` etc.

`params` a list of the true parameters with the elements

- `a_coef` the $K \times M$ coefficient matrix $(A_1, \ldots, A_p, C)$, with the columns in the order of the regressors of `create_bvarmodel`, or `NULL` if the model has no regressors. For `tvp = TRUE` a $K \times M \times T$ array.
- `a0_coef` for `structural = TRUE`, the $K \times K$ matrix A_0, or for `tvp = TRUE` a $K \times K \times T$ array.
- `psi_coef` the $K \times K$ matrix Ψ, or for `tvp = TRUE` a $K \times K \times T$ array. `NULL` if $K = 1$ or `structural = TRUE`.
- `u_omega` the $K \times K$ diagonal matrix of error variances Ω, or for `sv = TRUE` a $K \times K \times T$ array.
- `u_sigma` the $K \times K$ covariance matrix Σ of u_t, or for `tvp = TRUE` or `sv = TRUE` a $K \times K \times T$ array.
- `a_state_variance` for `tvp = TRUE`, the $K \times M$ matrix of the variances of the state equations of the coefficients.
- `a0_state_variance` for `tvp = TRUE` and `structural = TRUE`, the $K \times K$ matrix of the variances of the state equations of A_0.
- `psi_state_variance` for `tvp = TRUE`, $K > 1$ and `structural = FALSE`, the $K \times K$ matrix of the variances of the state equations of Ψ.
- `u_state_variance` for `sv = TRUE`, the K variances of the state equations of the log-volatilities.

For time varying parameters, `as.vector()` of an array gives the parameters period by period, which is the order of the columns of the posterior draws of TVP models.

References

Cogley, T., & Sargent, T. J. (2005). Drifts and volatilities: Monetary policies and outcomes in the post WWII US. *Review of Economic Dynamics, 8*(2), 262–302. doi:10.1016/j.red.2004.10.009

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Primiceri, G. E. (2005). Time varying structural vector autoregressions and monetary policy. *The Review of Economic Studies, 72*(3), 821–852. doi:10.1111/j.1467937X.2005.00353.x

See Also

Other artificial data: `generate_artificial_vec()`

Examples

```
# Set seed of RNG
set.seed(1)

# Time series without intercept terms
dt <- generate_artificial_var(nobs = 200, k = 3)

# Time series with all intercept terms equal to 5
dt <- generate_artificial_var(nobs = 200, k = 3, deterministic = "const",
                              range_const = c(5, 5))

# Time series with high levels, shifted by 100, 50 and 20
dt <- generate_artificial_var(nobs = 200, k = 3, deterministic = "const",
                              level = c(100, 50, 20))
dt[["params"]][["a_coef"]][, "const"]

# Time varying parameters and stochastic volatility with correlated errors
dt <- generate_artificial_var(nobs = 200, k = 2, p = 1, deterministic = "const",
                              tvp = TRUE, sv = TRUE, range_psi = c(-0.5, 0.5))

# Path of the error variance of the first variable
plot(dt[["params"]][["u_omega"]][1, 1, ], type = "l")

# Structural model
dt <- generate_artificial_var(nobs = 200, k = 3, p = 1, structural = TRUE)
dt[["params"]][["a0_coef"]]
```

`generate_artificial_vec` *Generate Artificial VEC Data*

Description

Generates an artificial data set from a vector error correction model with constant or time varying parameters and constant or stochastic volatility, for testing the algorithms of the package.

Usage

```
generate_artificial_vec(
  nobs = 100,
  k = 3,
  p = 2,
  r = 1,
  const = NULL,
  trend = NULL,
  structural = FALSE,
  tvp = FALSE,
```

```
  sv = FALSE,
  range_alpha = c(-0.5, 0.5),
  range_beta = c(-1, 1),
  range_gamma = c(-0.5, 0.5),
  gamma_zeros = 0.5,
  range_a0 = c(-0.5, 0.5),
  range_const = c(-0.5, 0.5),
  range_trend = c(-0.01, 0.01),
  range_variance = c(1, 1),
  range_psi = c(0, 0),
  range_variance_state = c(1e-04, 1e-04),
  range_variance_sv = c(0.01, 0.01),
  stable = TRUE,
  presample = 100,
  level = 0
)
```

Arguments

nobs	number of generated observations in levels. Defaults to 100.
k	number of endogenous variables. Defaults to 3.
p	lag order of the series in the (levels) VAR, so that the VEC model has $p-1$ lags of differences. Must be at least 1. Defaults to 2.
r	cointegration rank between 0 and k. Defaults to 1.
const	a character specifying whether a constant term enters the error correction term ("restricted") or the non-cointegration term as an "unrestricted" variable. If NULL (default) no constant term is added.
trend	a character specifying whether a linear trend enters the error correction term ("restricted") or the non-cointegration term as an "unrestricted" variable. If NULL (default) no trend is added.
structural	logical specifying whether a structural VEC model with a lower triangular matrix A_0 of contemporaneous coefficients is generated. Defaults to FALSE. See 'Details'.
tvp	logical specifying whether the coefficients, A_0 and the elements of Ψ follow random walks. Defaults to FALSE. See 'Details'.
sv	logical specifying whether the error variances follow a stochastic volatility process. Defaults to FALSE. See 'Details'.
range_alpha	numeric vector with two elements containing the minimum and maximum value of the loadings α. Defaults to c(-0.5, 0.5).
range_beta	numeric vector with two elements containing the minimum and maximum value of the free elements of the cointegration matrix β, which belong to endogenous variables. Defaults to c(-1, 1).
range_gamma	numeric vector with two elements containing the minimum and maximum value of the coefficients of lagged differences. Defaults to c(-0.5, 0.5).
gamma_zeros	numeric between 0 and 1 indicating the share of coefficients of lagged differences, which should be set to zero. Default is 0.5.

`range_a0` numeric vector with two elements containing the minimum and maximum value of the free elements of A_0. Only used if `structural = TRUE`. Defaults to `c(-0.5, 0.5)`.

`range_const` numeric vector with two elements containing the minimum and maximum value of the coefficients of the constant term. Defaults to `c(-0.5, 0.5)`.

`range_trend` numeric vector with two elements containing the minimum and maximum value of the coefficients of the linear trend. Defaults to `c(-0.01, 0.01)`.

`range_variance` numeric vector with two elements containing the minimum and maximum value of the error variances. For `sv = TRUE` these are the variances in the first period. Defaults to `c(1, 1)`.

`range_psi` numeric vector with two elements containing the minimum and maximum value of the free elements of Ψ. Defaults to `c(0, 0)`, which gives uncorrelated errors. Must be `c(0, 0)` for structural models.

`range_variance_state` numeric vector with two elements containing the minimum and maximum value of the variances of the state equations of the coefficients, of A_0 and of Ψ. Only used if `tvp = TRUE`. Defaults to `c(0.0001, 0.0001)`.

`range_variance_sv` numeric vector with two elements containing the minimum and maximum value of the variances of the state equations of the log-volatilities. Only used if `sv = TRUE`. Defaults to `c(0.01, 0.01)`.

`stable` logical specifying whether the coefficients are restricted to a process that is integrated of order one with cointegration rank `r`. Defaults to `TRUE`. See 'Details'.

`presample` numeric specifying the number of observations, which are generated before the first returned observation and then discarded. Defaults to 100.

`level` numeric vector with one or `k` elements, which are added to the generated series to shift their levels. Defaults to 0. A non-zero level requires a constant term if `r > 0`. See 'Details'.

Details

The function produces artificial observations for a vector error correction (VEC) model:

$$A_{0t}\Delta y_t = \alpha_t\beta_t' w_t + \sum_{i=1}^{p-1}\Gamma_{it}\Delta y_{t-i} + C_t d_t + u_t,$$

where Δy_t is a K-dimensional vector of differenced endogenous variables, w_t is the vector of the endogenous variables in levels y_{t-1} and the restricted deterministic terms, α_t is the $K \times r$ matrix of loadings, β_t is the cointegration matrix with $K+N^R$ rows and r columns, $\Pi_t = \alpha_t\beta_t'$, Γ_{it} is a $K \times K$ coefficient matrix of lagged differences, d_t is the vector of unrestricted deterministic terms and C_t its coefficient matrix. The error term u_t and A_{0t} are specified as in `generate_artificial_var`: $u_t \sim N(0, \Sigma_t)$ with $\Psi_t\Sigma_t\Psi_t' = \Omega_t$, and A_{0t} is an identity matrix unless `structural = TRUE`, where it is lower triangular with ones on its main diagonal and the structural errors are uncorrelated. This is the model that `create_bvecmodel` produces with the same arguments `p`, `r`, `const`, `trend` and `structural`.

The cointegration matrix is normalised so that its first r rows are an identity matrix. Its other rows for endogenous variables are drawn from `range_beta`, and its rows for a restricted constant

or trend from range_const and range_trend. Since add_posterior_coefficients may use another normalisation, estimates are best compared with the true parameters through Π_t, which does not depend on it.

The linear trend takes the value 1 in period $p+1$ of the returned series, which is the first period that create_bvecmodel uses for estimation with lag order p.

If tvp = TRUE, the loadings, the free elements of β_t, the coefficients of lagged differences and unrestricted deterministic terms, the free elements of A_{0t} and the free elements of Ψ_t follow random walks, whose innovation variances are drawn from range_variance_state. Coefficients that are zero in the first period stay zero. The random walk of the normalised β_t differs from the state equation of the unnormalised cointegration matrix in Koop et al. (2011), which create_bvecmodel uses for estimation. If sv = TRUE, the log-volatilities follow random walks as in generate_artificial_var. During the presample the parameters are held at their values of the first period.

If stable = TRUE, the coefficients are drawn until the VAR representation of the reduced form of the process has exactly $K - r$ unit roots and all other roots inside the unit circle, so that the series are integrated of order one and $\beta_t' w_t$ is stationary. With r = 0 the series are integrated but not cointegrated, with r = k they are stationary. For tvp = TRUE this holds in every period: an innovation, which violates it, is drawn again, and if no admissible innovation is found in 100 attempts, the coefficients keep the values of the previous period.

The series start at zero before the presample. Argument level shifts them by a vector m, which gives series with high levels that follow a stochastic trend. The dynamics of the differences are unchanged, but the error correction term becomes $\alpha_t \beta_t' (y_{t-1} - m)$, so that the constant term absorbs $-\beta_t' m$: the returned row of a restricted constant in β_t is $\beta_t^c - \beta_t' m$ and the returned unrestricted constant is $c_t - \alpha_t \beta_t' m$, where β_t^c and c_t are drawn from range_const and β_t here denotes the rows of the endogenous variables. Without a constant this term would not be part of the model, so that a non-zero level with r > 0 requires const = "restricted" or "unrestricted".

Value

A list with the elements

data a $T \times K$ time-series object of the artificial series in levels, named var1, var2 etc.

params a list of the true parameters with the elements

- alpha the $K \times r$ matrix of loadings α.
- beta the $(K + N^R) \times r$ cointegration matrix β.
- pi the $K \times (K + N^R)$ matrix $\Pi = \alpha\beta'$, with the columns in the order of the cointegration variables of create_bvecmodel.
- gamma the $K \times K(p-1)$ matrix $(\Gamma_1, \ldots, \Gamma_{p-1})$.
- c the $K \times N^{UR}$ matrix C of unrestricted deterministic terms.
- a0_coef for structural = TRUE, the $K \times K$ matrix A_0.
- psi_coef, u_omega, u_sigma the parameters of the error term as in generate_artificial_var.
- alpha_state_variance, beta_state_variance, gamma_state_variance, c_state_variance, a0_state_varianc for tvp = TRUE, the variances of the state equations of the respective parameters.
- u_state_variance for sv = TRUE, the K variances of the state equations of the log-volatilities.

Parameters that are not part of the model are NULL. Time varying parameters are $\cdots \times T$ arrays with one matrix per period.

References

Johansen, S. (1995). *Likelihood-based inference in cointegrated vector autoregressive models*. Oxford: Oxford University Press.

Koop, G., León-González, R., & Strachan R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

See Also

Other artificial data: `generate_artificial_var()`

Examples

```
# Set seed of RNG
set.seed(1)

# Three series with one cointegration relationship
dt <- generate_artificial_vec(nobs = 200, k = 3, p = 2, r = 1)
dt[["params"]][["pi"]]

# Restricted constant and time varying cointegration with stochastic volatility
dt <- generate_artificial_vec(nobs = 200, k = 2, p = 1, r = 1, const = "restricted",
                              tvp = TRUE, sv = TRUE, range_variance_state = c(0.001, 0.001))

# Path of the cointegration coefficient of the second variable
plot(dt[["params"]][["beta"]][2, 1, ], type = "l")

# Cointegrated series with high levels around 100, 50 and 20
dt <- generate_artificial_vec(nobs = 200, k = 3, p = 2, r = 1, const = "restricted",
                                level = c(100, 50, 20))
dt[["params"]][["beta"]]

# Structural model with an unrestricted constant
dt <- generate_artificial_vec(nobs = 200, k = 3, p = 2, r = 1, const = "unrestricted",
                                structural = TRUE)
```

---

generate_lower_block_diagonal

Posterior Data Preparation

Description

Generates a lower triangular block matrix with ones on the main diagonal, where the off-diagonal elements are blocks.

Usage

```
generate_lower_block_diagonal(a, k, tt)
```

Arguments

a	K^2p-dimensional vector of coefficients. See 'Details'.
k	integer of the number of columns per block.
tt	integer specifying

Details

For the $K \times Kp$ matrix A, where $a = vec(A)$, with $A = [A_1, A_2, ..., A_p]$ the function constructs the following sparse $KT \times KT$ diagonal block matrix:

$$\begin{bmatrix} I_K & 0 & 0 & 0 & \dots & 0 \\ -A_1 & I_K & 0 & 0 & \dots & 0 \\ -A_2 & -A_1 & I_K & 0 & \dots & 0 \\ 0 & -A_2 & -A_1 & I_K & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -A_2 & -A_1 & I_K \end{bmatrix}.$$

Value

A sparse matrix.

Examples

```
a <- matrix(1:8)
generate_lower_block_diagonal(a, 2, 5)
```

---

get_forecast_errors *Get Forecast Errors*

Description

A generic function used to obtain the forecast errors from a model.

Usage

```
get_forecast_errors(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

`get_model_specifications` *Get Model Specifications*

Description

A generic function used to obtain the model specifications of a model.

Usage

```
get_model_specifications(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

A one-row data frame with the specifications of the model. Which columns are returned depends on the class of argument `object`:

type the type of the model.

k the number of endogenous variables.

p the lag order of the endogenous variables.

m the number of unmodelled, non-deterministic variables.

s the lag order of the unmodelled, non-deterministic variables.

n the number of deterministic terms. For objects of class 'bvecmodel' the columns `n_unrestricted` and `n_restricted` are returned instead.

rank the rank of the cointegration matrix. Only for objects of class 'bvecmodel'.

T the number of observations used for estimation. Not for objects of class 'expandingwindow', where it differs across the estimation windows, and not for objects of class 'selcrit', which do not contain the input data.

varsel the used variable selection algorithm.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 10, burnin = 10)

# Obtain model specifications
get_model_specifications(model)
```

historical_decomposition *Historical Decomposition*

Description

Decomposes the path of a variable over the estimation sample into the contributions of the identified structural shocks.

Usage

```
historical_decomposition(x, ...)
```

Arguments

x	an object of class 'bvarmodel' with posterior draws.
...	further arguments passed to or from other methods.

Value

See the method, historical_decomposition.bvarmodel.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

historical_decomposition.bvarmodel *Historical Decomposition of a Vector Autoregressive Model*

Description

Decomposes the path of a variable over the estimation sample into the contributions of the identified structural shocks and a baseline.

Usage

```
## S3 method for class 'bvarmodel'
historical_decomposition(
  x,
  response = NULL,
  type = "oir",
  impact = NULL,
  statistic = "mean",
  ci = NULL,
  ...
)
```

Arguments

x	an object of class 'bvarmodel' with posterior draws of the coefficients and the error covariance.
response	name of the variable whose path is decomposed.
type	the identification: "oir" (default) for orthogonalised shocks from the Choleski factor of each draw's covariance, "sign" for the rotations add_sign_restrictions or add_sign_zero_restrictions stored in the model, and "custom" for impact matrices given in impact.
impact	for type = "custom", the impact matrix: one matrix, a list of one matrix per draw, or a function of a draw, as in irf.bvarmodel. Column j is the impact response to shock j; its column names, where it has any, name the shocks.
statistic	the posterior summary of each contribution, "mean" (default) or "median". Only the means add up to the data exactly.
ci	an optional probability, the coverage of the credible interval returned alongside, such as 0.68.
...	further arguments passed to or from other methods.

Details

With $u_t = P\epsilon_t$ the reduced form errors and ϵ_t the structural shocks, the model's moving average form splits each period's value into

$$y_t = b_t + \sum_j \sum_{s=0}^{t-1} \Phi_s P_{\cdot j} \epsilon_{j,t-s},$$

the contribution of each shock j over the sample so far plus a baseline b_t: what the lags before the sample, the deterministic terms and the exogenous variables would have produced without any shock. For every draw the shocks are recovered from that draw's residuals and its impact matrix, and the contributions are propagated by the draw's lag coefficients, so that the decomposition carries the posterior uncertainty of all three.

Where the panel was not observed whole (see create_bvarmodel), each draw is decomposed over its own completed panel, so that the periods nobody observed are decomposed as the model filled them.

Available for models with constant coefficients, a lag order of at least one and a covariance that does not change over the sample; that excludes time-varying parameters and stochastic volatility, whose impact matrix would differ from period to period, and structural models.

Value

A time series of class "bvarhd", one row per period of the estimation sample and one column per shock plus a last column baseline. With ci, attributes lower and upper hold the bounds of the credible interval in the same shape. The response itself is attribute data; for a completed panel it is the sum of the columns, the posterior mean of the completed series.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100
model <- create_bvarmodel(e1, p = 2, iterations = 100, burnin = 50)
model <- add_priors(model, coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
model <- add_posterior_coefficients(add_initial_values(model))
hd <- historical_decomposition(model, response = "cons")
plot(hd)
```

---

inclusion_prior *Prior Inclusion Probabilities*

Description

A generic function used to generate prior inclusion probabilities as required for stochastic search variable selection (SSVS) à la George et al. (2008) and Bayesian variable selection (BVS) à la Korobilis (2013). The function invokes particular methods which depend on the class of the first argument.

Usage

```
inclusion_prior(object, ...)
```

Arguments

object	an object of a class, for which a method should be called.
...	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `inclusion_prior.bvarmodel`, `inclusion_prior.bvecmodel`, `inclusion_prior.externalforecast`.

`inclusion_prior.bvarmodel`
Prior Inclusion Probabilities

Description

Prior inclusion probabilities as required for stochastic search variable selection (SSVS) à la George et al. (2008) and Bayesian variable selection (BVS) à la Korobilis (2013).

Usage

```
## S3 method for class 'bvarmodel'
inclusion_prior(
  object,
  prob = 0.5,
  exclude_deterministics = TRUE,
  minnesota_like = FALSE,
  kappa1 = 0.8,
  kappa2 = 0.5,
  kappa3 = 0.5,
  kappa4 = 0.8,
  ...
)
```

Arguments

`object`	an object of class `"bvarmodel"`, usually, a result of a call to `create_bvarmodel`.
`prob`	a numeric specifying the prior inclusion probability of all model parameters.
`exclude_deterministics`	logical. If `TRUE` (default), the vector of the positions of included variables does not include the positions of deterministic terms.
`minnesota_like`	logical. If `TRUE`, the prior inclusion probabilities of the parameters are calculated in a similar way as the Minnesota prior. See 'Details'.
`kappa1`	a numeric specifying the prior inclusion probability of coefficients that correspond to own lags of endogenous variables. Only used if `minnesota_like = TRUE`. See 'Details'.
`kappa2`	a numeric specifying the size of the prior inclusion probabilities of endogenous variables, which do not correspond to own lags. Only used if `minnesota_like = TRUE`. See 'Details'.

kappa3	a numeric specifying the size of the prior inclusion probabilities of non-deterministic exogenous variables, between 0 and 1. Default is 0.5. Only used if minnesota_like = TRUE. See 'Details'.
kappa4	a numeric specifying the size of the prior inclusion probabilities of deterministic terms. Only used if minnesota_like = TRUE. See 'Details'.
...	further arguments passed to or from other methods.

Details

If minnesota_like = TRUE, prior inclusion probabilities $\underline{\pi}_1$ are calculated as

$\frac{\kappa_1}{r}$	for own lags of endogenous variables,
$\frac{\kappa_2}{r}$	for other endogenous variables,
$\frac{\kappa_3}{1+r}$	for unmodelled exogenous variables,
κ_2	for contemporaneous endogenous variables of a structural model,
κ_4	for deterministic variables.

Value

A list containing a matrix of prior inclusion probabilities and an integer vector specifying the positions of variables, which should be included in the variable selection algorithm.

Examples

```
# Prepare data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model input
object <- create_bvarmodel(e1)

# Obtain inclusion prior
incl <- inclusion_prior(object)
```

---

inclusion_prior.bvecmodel

Prior Inclusion Probabilities

Description

Prior inclusion probabilities as required for stochastic search variable selection (SSVS) à la George et al. (2008) and Bayesian variable selection (BVS) à la Korobilis (2013).

Usage

```
## S3 method for class 'bvecmodel'
inclusion_prior(
  object,
  prob = 0.5,
  exclude_deterministics = TRUE,
  minnesota_like = FALSE,
  kappa1 = 0.8,
  kappa2 = 0.5,
  kappa3 = 0.5,
  kappa4 = 0.8,
  ...
)
```

Arguments

`object`	an object of class 'bvecmodel', usually, a result of a call to `create_bvecmodel`.
`prob`	a numeric specifying the prior inclusion probability of all model parameters.
`exclude_deterministics`	logical. If `TRUE` (default), the vector of the positions of included variables does not include the positions of deterministic terms.
`minnesota_like`	logical. If `TRUE`, the prior inclusion probabilities of the parameters are calculated in a similar way as the Minnesota prior. See 'Details'.
`kappa1`	a numeric specifying the prior inclusion probability of coefficients that correspond to own lags of endogenous variables. Only used if `minnesota_like = TRUE`. See 'Details'.
`kappa2`	a numeric specifying the size of the prior inclusion probabilities of endogenous variables, which do not correspond to own lags. Only used if `minnesota_like = TRUE`. See 'Details'.
`kappa3`	a numeric specifying the size of the prior inclusion probabilities of non-deterministic exogenous variables, between 0 and 1. Default is 0.5. Only used if `minnesota_like = TRUE`. See 'Details'.
`kappa4`	a numeric specifying the size of the prior inclusion probabilities of deterministic terms. Only used if `minnesota_like = TRUE`. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

If `minnesota_like = TRUE`, prior inclusion probabilities $\underline{\pi}_1$ are calculated as

$\frac{\kappa_1}{r}$	for own lags of endogenous variables,
$\frac{\kappa_2}{r}$	for other endogenous variables,
$\frac{\kappa_3}{1+r}$	for unmodelled exogenous variables,
κ_2	for contemporaneous endogenous variables of a structural model,
κ_4	for deterministic variables.

Value

A list containing a matrix of prior inclusion probabilities and an integer vector specifying the positions of variables, which should be included in the variable selection algorithm.

Examples

```
# Prepare data
data("e6")

# Generate model input
object <- create_bvecmodel(e6, p = 2, r = 1, const = "unrestricted")

# Obtain inclusion prior
priors <- inclusion_prior(object)
```

---

irf *Impulse Response Function*

Description

A generic function used to calculate impulse response functions.

Usage

```
irf(x, ...)
```

Arguments

x	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of x, as described on the pages of the methods.

See Also

Methods: irf.bvarmodel.

`irf.bvarmodel` *Impulse Response Function*

Description

Computes the impulse response coefficients for an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
irf(
  x,
  impulse = NULL,
  response = NULL,
  n_ahead = 5,
  ci = 0.95,
  shock = 1,
  type = "feir",
  cumulative = FALSE,
  keep_draws = FALSE,
  period = NULL,
  impact = NULL,
  ...
)
```

Arguments

`x`	an object of class 'bvarmodel'.
`impulse`	name of the impulse variable.
`response`	name of the response variable.
`n_ahead`	number of steps ahead. Zero is allowed and returns the impact response alone.
`ci`	a numeric between 0 and 1 specifying the probability mass covered by the credible intervals. Defaults to 0.95.
`shock`	size of the shock. For `"oir"`, `"gir"` and `"sgir"` it is counted in standard deviations of the shock, so the default of 1 is a shock of one standard deviation. For `"feir"` and `"sir"` it is counted in units of the reduced form or structural error of the impulse variable, and for `"sign"` and `"custom"` it rescales the columns of the impact matrix. `"sd"` and `"nsd"` are a positive and a negative shock of one standard deviation and are not available for `"sign"` and `"custom"`.
`type`	type of the impulse response. Possible choices are forecast error `"feir"` (default), orthogonalised `"oir"`, structural `"sir"`, generalised `"gir"`, structural generalised `"sgir"`, sign restricted `"sign"` and `"custom"` impulse responses. For a structural model only `"sir"` and `"sgir"` are available; the other five require a non-structural model. See 'Details'.
`cumulative`	logical specifying whether a cumulative IRF should be calculated.

keep_draws	logical specifying whether the function should return all draws of the posterior impulse response function. Defaults to FALSE so that the median and the credible intervals of the posterior draws are returned.
period	integer. Index of the period, for which the IR should be generated. Only used for TVP or SV models. Default is NULL, so that the posterior draws of the last time period are used. With type = "sign" the default is the period the restrictions were imposed in, and another period is refused, since the rotations do not identify it.
impact	the impact matrix of a "custom" impulse response, either a single $K \times K$ matrix that identifies every posterior draw the same way, or a list of such matrices with one entry per draw. Ignored for every other value of type.
...	further arguments passed to or from other methods.

Details

The function produces different types of impulse responses for the VAR model

$$A_0 y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t,$$

with $u_t \sim N(0, \Sigma)$.

Forecast error impulse responses Φ_i are obtained by recursions

$$\Phi_i = \sum_{j=1}^{i} \Phi_{i-j} A_j, i = 1, 2, ..., h$$

with $\Phi_0 = I_K$.

Orthogonalised impulse responses Θ_i^o are calculated as $\Theta_i^o = \Phi_i P$, where P is the lower triangular Choleski decomposition of Σ, so that they are the responses to orthogonalised shocks of one standard deviation.

Structural impulse responses Θ_i^s are calculated as $\Theta_i^s = \Phi_i A_0^{-1}$.

For a structural model the posterior draws describe the structural form: the coefficients are the A_i of the equation above and Σ is the covariance of the structural errors. The recursion for Φ_i needs the reduced form, i.e. $A_0^{-1} A_i$ and $A_0^{-1} \Sigma A_0^{-1\prime}$, which only "sir" and "sgir" form. The other types are therefore not available for a structural model, and the two structural types not for any other.

Sign restricted impulse responses Θ_i^r are calculated as $\Theta_i^r = \Phi_i P Q$, where P is the lower triangular Choleski decomposition of Σ and Q the rotation that add_sign_restrictions accepted for that draw. Draws for which no admissible rotation was found are left out, so the responses cover fewer draws than the posterior holds and the credible interval is one over the set of models the restrictions admit rather than over a single identified model.

Custom impulse responses Θ_i^c are calculated as $\Theta_i^c = \Phi_i P$, where P is the matrix supplied in argument impact. This is the route by which an identification that the package does not derive itself reaches the recursion; shock rescales the result but nothing normalises the columns of P, so an impact matrix that means to deliver unit shocks has to arrive that way.

(Structural) Generalised impulse responses to a shock to variable j are calculated as $\Theta_i^g = \sigma_{jj}^{-1/2} \Phi_i A_0^{-1} \Sigma e_j$, where σ_{jj} is the jth diagonal element of Σ, the variance of the error of the impulse variable, and

e_j is a selection vector containing one in its jth element and zero otherwise. They are therefore the responses to a shock of one standard deviation (Pesaran and Shin, 1998). If the "bvarmodel" object does not contain draws of A_0, it is assumed to be an identity matrix.

Value

A time-series object of class 'bvarirf' running from period 0 to n_ahead, with the lower bound, the median and the upper bound of the credible band of the response in three columns named after their quantiles, e.g. "2.5%", "50%" and "97.5%" for ci = .95. If keep_draws = TRUE, a matrix of class 'bvarirf' with one row per draw and one column per period instead.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Pesaran, H. H., Shin, Y. (1998). Generalized impulse response analysis in linear multivariate models. *Economics Letters, 58*, 17-29.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Obtain IR
ir <- irf(model, impulse = "invest", response = "cons")
```

irf.bvecmodel *Impulse Response Function*

Description

Guard method for objects of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
irf(x, ...)
```

Arguments

x	an object of class 'bvecmodel'.
...	additional arguments.

Value

Nothing. The method raises an error, since impulse responses of a VEC model are obtained from its VAR representation: apply `vec_to_var` first and then `irf` to the resulting 'bvarmodel'.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

kalman_durbin_koopman_2002 *Durbin and Koopman Simulation Smoother*

Description

An implementation of the Kalman filter and backward smoothing algorithm proposed by Durbin and Koopman (2002).

Usage

```
kalman_durbin_koopman_2002(y, z, sigma_u, sigma_v, B, a_init, P_init)
```

Arguments

y	a $K \times T$ matrix of endogenous variables.
z	a $KT \times M$ matrix of explanatory variables.
sigma_u	the constant $K \times K$ error variance-covariance matrix. For time varying variance-covariance matrices a $KT \times K$ can be specified.
sigma_v	the constant $M \times M$ coefficient variance-covariance matrix. For time varying variance-covariance matrices a $MT \times M$ can be specified.
B	the constant $M \times M$ autocorrelation matrix of the transition equation. For a time varying transition an $MT \times M$ matrix can be specified.
a_init	an M-dimensional vector of initial states.
P_init	an $M \times M$ variance-covariance matrix of the initial states.

Details

The function uses algorithm 2 from Durbin and Koopman (2002) to produce a draw of the state vector a_t for $t = 1, ..., T$ for a state space model with measurement equation

$$y_t = Z_t a_t + u_t$$

and transition equation

$$a_{t+1} = B_t a_t + v_t,$$

where $u_t \sim N(0, \Sigma_{u,t})$ and $v_t \sim N(0, \Sigma_{v,t})$. y_t is a K-dimensional vector of endogenous variables and $Z_t = z_t' \otimes I_K$ is a $K \times M$ matrix of regressors with z_t as a vector of regressors.

The algorithm takes into account Jarociński (2015), where a possible missunderstanding in the implementation of the algorithm of Durbin and Koopman (2002) is pointed out. Following that note the function sets the mean of the initial state to zero in the first step of the algorithm.

This is the routine the time varying parameter samplers of the package use. It comes from the vendored BayesTS core, so there is one implementation of the smoother rather than one for R and one for the samplers.

The draw uses R's random number generator, so `set.seed` makes it reproducible.

Value

A $M \times T + 1$ matrix of state vector draws. Column i is the state the observation in column i of y loads on, for $i = 1, ..., T$, so a caller wanting one state per observation takes the first T columns. The first column is not a state preceding the sample: it is a_1, already conditioned on every observation. The last column is the transition applied once past the end of the sample and is informed by no observation; it is returned because the recursions build it on the way.

References

Durbin, J., & Koopman, S. J. (2002). A simple and efficient simulation smoother for state space time series analysis. *Biometrika, 89*(3), 603–615.

Jarociński, M. (2015). A note on implementing the Durbin and Koopman simulation smoother. *Computational Statistics and Data Analysis, 91*, 1–3. doi:10.1016/j.csda.2015.05.001

Examples

```
# Load data
data("e1")
data <- diff(log(e1))

# Generate model data
temp <- create_bvarmodel(data = data, p = 2, deterministic = "const",
                         iterations = 1, burnin = 0)
y <- t(temp$data$train$y)
z <- temp$data$train$z
k <- nrow(y)
tt <- ncol(y)
m <- ncol(z)

# Priors
a_mu_prior <- matrix(0, m)
a_v_i_prior <- diag(0.1, m)

a_Q <- diag(.0001, m)

# Initial value of Sigma
sigma <- tcrossprod(y) / tt
sigma_i <- solve(sigma)

# Initial values for Kalman filter
y_init <- y * 0
a_filter <- matrix(0, m, tt + 1)

# Initialise the Kalman filter
for (i in 1:tt) {
  y_init[, i] <- y[, i] - z[(i - 1) * k + 1:k,] %*% a_filter[, i]
}
a_init <- post_normal_sur(y = y_init, z = z, sigma_i = sigma_i,
                          a_prior = a_mu_prior, v_i_prior = a_v_i_prior)
y_filter <- matrix(y) - z %*% a_init
y_filter <- matrix(y_filter, k) # Reshape

# Kalman filter and backward smoother
a_filter <- kalman_durbin_koopman_2002(y = y_filter, z = z, sigma_u = sigma,
                                       sigma_v = a_Q, B = diag(1, m),
                                       a_init = matrix(0, m), P_init = a_Q)

a <- a_filter + matrix(a_init, m, tt + 1)
```

---

list_models_in_hdf5 *Models in an HDF5 File*

Description

Finds the groups of an HDF5 file that hold a model.

Usage

```
list_models_in_hdf5(filename, group = "")
```

Arguments

filename	path to an HDF5 file.
group	the group to search under. Defaults to "", the root of the file, which searches all of it.

Details

A model is a group with a model subgroup carrying an algorithm attribute. The search stops at a model rather than descending into its data, priors and posterior, which are its own subtree and not a place further models could be.

This is the same rule the BayesTS command line applies for its --all-groups flag, so both agree on which groups of a file are models.

Value

A character vector of group names, sorted, empty if the file holds no model. A model at the root of its file is reported as "", which is the value read_model_from_hdf5 takes for it.

Examples

```
# Load data
data("e1")
train <- diff(log(e1)) * 100

# Create and store two models in one file
path_to_model <- tempfile(fileext = ".h5")
for (p in 1:2) {
  model <- create_bvarmodel(data = train, p = p, deterministic = "const",
                            iterations = 10, burnin = 10)
  write_to_hdf5(model, filename = path_to_model,
                group = paste0("/models/", p))
}

list_models_in_hdf5(path_to_model)
```

---

loglik_normal *Log-Likelihood of a Multivariate Normal Distribution*

Description

Calculates the log-likelihood of a multivariate normal distribution.

Usage

```
loglik_normal(u, sigma)
```

Arguments

u	a $K \times T$ matrix of residuals.
sigma	a $K \times K$ or $KT \times K$ variance-covariance matrix.

Details

The log-likelihood is calculated for each vector in period t as

$$-\frac{K}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_t| - \frac{1}{2}u_t'\Sigma_t^{-1}u_t$$

, where $u_t = y_t - \mu_t$.

Value

A numeric vector of length T, the log-likelihood of each period. Sum it for the log-likelihood of the sample.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1))

# Generate VAR model
data <- create_bvarmodel(e1, p = 2, deterministic = "const")
y <- t(data$data$train$y)
x <- t(data$data$train$x)

# LS estimate
ols <- tcrossprod(y, x) %*% solve(tcrossprod(x))

# Residuals
u <- y - ols %*% x # Residuals

# Covariance matrix
sigma <- tcrossprod(u) / ncol(u)

# Log-likelihood
loglik_normal(u = u, sigma = sigma)
```

map_draws *Apply a Function to the Draws of a Stored Model*

Description

Reads the chain of a stored model in pieces and applies a function to the model holding each piece, which is how the analysis of a model too large to hold is done.

Usage

```
map_draws(x, f, ..., chunk = 100)
```

Arguments

x	an object of class 'bvarfile', from open_model.
f	a function taking a model and returning whatever the caller wants to keep of that piece of the chain.
...	further arguments passed to f.
chunk	how many draws are read at a time. Defaults to 100.

Details

The model f is given is the model in the file with the draws of one piece, so anything that works on a model works on it. What it returns is collected in a list, one element per piece, and combining those is the caller's business: the responses of an impulse response are stacked, the shares of a variance decomposition are averaged over the pieces they came from.

Value

A list with one element per piece of the chain.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
model <- add_priors(model,
```

```
                    coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = 1, scale = .0001))
model <- add_posterior_coefficients(add_initial_values(model))

file <- file.path(tempdir(), "bvartools-example-draws.h5")
unlink(file)
write_to_hdf5(model, filename = file)

stored <- open_model(file)

# The mean of every coefficient, without reading the chain at once
sums <- map_draws(stored, function(model) {
  colSums(unclass(model[["posterior"]][["a"]][["coeffs"]]))
}, chunk = 25)
means <- Reduce(`+`, sums) / stored[["draws"]]
```

`map_models` *Apply a Function to the Models of a Folder*

Description

Reads every model of a folder, applies a function to it, writes it back and drops it, so that a step over many models costs the memory of one model per worker rather than of all of them.

Usage

```
map_models(x, f, ..., models = NULL, write = TRUE, cores = 1, export = NULL)
```

Arguments

`x`	an object of class 'bvarfolder', from `open_models`.
`f`	a function taking a model – a 'bvarmodel', a 'bvecmodel', or the model of another package that `read_model_from_hdf5` reads – and returning one. If it has an argument named `index`, the position of the model in the manifest is passed to it, which is what numbers seeds and what lets a caller tell the models apart.
`...`	further arguments passed to `f`.
`models`	character vector of the models the step is applied to, as the manifest names them, or `NULL`, the default, for all of them. The others are left untouched, and so are their rows of the manifest.
`write`	whether what `f` returns is written back. `TRUE`, the default, is a step of an estimation: `f` takes a model and returns one, and the model in the file becomes what it returned. `FALSE` reads only: `f` may return anything, nothing is written, and the results are collected and returned.
`cores`	the number of worker processes the models are handled on. Defaults to one. With more than one the models are spread over a socket cluster; every model has its own file, so the workers share nothing.

`export` character vector of names of objects in the calling environment that the workers need, for instance the sampler that `posterior_function` refers to. Ignored with one core.

Details

The unit is one model, which is what a file of the folder holds. A model is read with `read_model_from_hdf5`, passed to `f`, and written back to its own file through a temporary file that replaces the old one once it is complete, so an interrupted step leaves either the model as it was or the model as `f` made it, never half of it. A file holding more than one model cannot be worked on this way, since writing one of them back would have to rewrite the others, and such a model is refused rather than risked.

The manifest is rebuilt from the specifications of the models that come back, so a step that changes what a model is – its rank, its lag orders, the chain it asks for – leaves the table describing the files. It records what a model is, not how many draws are left in it, so thinning the draws does not change it.

A step over a subset is a step over the whole folder applied to part of it: the position a model has in the whole manifest is what is passed as `index` and what numbers its seed, so a run taken model by model draws what a run over all of them draws. That is what makes a long estimation resumable, and the estimation steps for a folder pass `models` on.

With more than one core, `f` is sent to the workers as it stands, and the workers load the installed package. A function that calls something the package does not export, or something only a newer version of it has, fails there rather than in the session it was written in. Pass what such a function needs as an argument instead.

For the posterior itself, prefer `bayests_files` over a cluster: the draws are the largest thing a model has, and reading them into R and writing them out again costs more than the sampler. An idle R worker is not cheap either – on Windows a fresh one commits about 2 GB before it does anything and about 4 GB once it has loaded this package.

Value

With `write = TRUE`, the object in `x` with its manifest brought up to date, invisibly. With `write = FALSE`, a named list of what `f` returned, one element per model.

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `choose_best_model.selcri`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
data("e1")
train <- diff(log(e1)) * 100

models <- create_bvarmodel(data = train, p = 1:2, deterministic = "const",
```

```
                          iterations = 20, burnin = 10)

folder <- file.path(tempdir(), "bvartools-example-map")
unlink(folder, recursive = TRUE)
dir.create(folder, recursive = TRUE)
write_to_hdf5(models, folder = folder)

stored <- open_models(folder)

# What add_priors() for a folder does
stored <- map_models(stored, function(model) {
  add_priors(model, coef = list(v_i = 1), sigma = list(df = 3, scale = 1))
})

# Reading only: how many observations each model was built on
map_models(stored, function(model) nrow(model$data$train$y), write = FALSE)
```

`minnesota_prior` *Minnesota Prior*

Description

Calculates the Minnesota prior for a model.

Usage

```
minnesota_prior(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Details

A generic function used to generate data matrices based on the Minnesota prior. The function invokes particular methods which depend on the class of the first argument.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `minnesota_prior.bvarmodel`, `minnesota_prior.bvecmodel`, `minnesota_prior.externalforecast`, `minnesota_prior.modellist`.

`minnesota_prior.bvarmodel` *Minnesota Prior*

Description

Calculates the Minnesota prior for a VAR model.

Usage

```
## S3 method for class 'bvarmodel'
minnesota_prior(
  object,
  kappa1 = 2,
  kappa2 = 0.5,
  kappa3 = NULL,
  kappa4 = 5,
  max_var = NULL,
  coint_var = FALSE,
  sigma = "AR",
  ...
)
```

Arguments

`object`	an object of class 'bvarmodel', usually, a result of a call to `create_bvarmodel`.
`kappa1`	a numeric specifying the prior variance of coefficients that correspond to own lags of endogenous variables. See 'Details'.
`kappa2`	a numeric specifying the size of the prior variance of endogenous variables, which do not correspond to own lags. See 'Details'.
`kappa3`	a numeric specifying the size of the prior variance of non-deterministic exogenous variables. Default is `NULL`, which indicates that the formula for the calculation of the prior variance of deterministic terms is used for all exogenous variables. See 'Details'.
`kappa4`	a numeric specifying the size of the prior variance of deterministic terms relative to argument `kappa1`.
`max_var`	a positive numeric specifying the maximum prior variance that is allowed for coefficients of non-deterministic variables. If `NULL` (default), the prior variances are not limited.
`coint_var`	a logical specifying whether the model is a cointegrated VAR model, for which the prior means of first own lags should be set to one.
`sigma`	either `"AR"` (default) or `"VAR"` indicating that the variances of the endogenous variables σ^2 are calculated based on a univariate AR regression or a least squares estimate of the VAR form, respectively. In both cases all deterministic variables are used in the regressions, if they appear in the model.
`...`	further arguments passed to or from other methods.

Details

The function calculates the Minnesota prior of a VAR model. For the endogenous variable i the prior variance of the lth lag of regressor j is obtained as

$$\frac{\kappa_1}{l^2} \text{ for own lags of endogenous variables,}$$

$$\frac{\kappa_1 \kappa_2}{l^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for endogenous variables other than own lags,}$$

$$\frac{\kappa_1 \kappa_3}{(l+1)^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for unmodelled exogenous variables,}$$

$$\kappa_1 \kappa_4 \sigma_i^2 \text{ for deterministic terms,}$$

where σ_i is the residual standard deviation of variable i of an unrestricted LS estimate. For exogenous variables σ_i is the sample standard deviation. In case structural parameters are estimated, the formula $\kappa_1 \kappa_2 \frac{\sigma_i^2}{\sigma_j^2}$ is used. If $kappa_3$ is not provided, prior variances are calculated in the same way as for deterministic terms. If the model does not contain exogenous variables, argument `kappa3` will be ignored.

Value

A list containing a matrix of prior means and the precision matrix of the coefficients and the inverse variance-covariance matrix of the error term, which was obtained by an LS estimation.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias, J. L. (2019). *Bayesian Econometric Methods* (2nd ed.). Cambridge: University Press.

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model input
object <- create_bvarmodel(e6, p = 1,
                           deterministic = "both",
                           seasonal = TRUE)

# Obtain Minnesota prior
prior <- minnesota_prior(object)
```

`minnesota_prior.bvecmodel` *Minnesota Prior*

Description

Calculates the Minnesota prior for a VEC model.

Usage

```
## S3 method for class 'bvecmodel'
minnesota_prior(
  object,
  kappa1 = 2,
  kappa2 = 1,
  kappa3 = NULL,
  kappa4 = 10,
  max_var = NULL,
  sigma = "AR",
  ...
)
```

Arguments

`object`	an object of class 'bvecmodel', usually, a result of a call to `create_bvecmodel`.
`kappa1`	a numeric specifying the prior variance of coefficients that correspond to own lags of endogenous variables. See 'Details'.
`kappa2`	a numeric specifying the size of the prior variance of endogenous variables, which do not correspond to own lags. See 'Details'.
`kappa3`	a numeric specifying the size of the prior variance of non-deterministic exogenous variables. Default is `NULL`, which indicates that the formula for the calculation of the prior variance of deterministic terms is used for all exogenous variables. See 'Details'.
`kappa4`	a numeric specifying the size of the prior variance of deterministic terms. See 'Details'.
`max_var`	a positive numeric specifying the maximum prior variance that is allowed for coefficients of non-deterministic variables. If `NULL` (default), the prior variances are not limited.
`sigma`	either `"AR"` (default) or `"VAR"` indicating that the variances of the endogenous variables σ^2 are calculated based on a univariate AR regression or a least squares estimate of the VAR form, respectively. In both cases all deterministic variables are used in the regressions, if they appear in the model.
`...`	further arguments passed to or from other methods.

Details

The function calculates the Minnesota prior in the same manner as for a traditional VAR model. For the endogenous variable i the prior variance of the lth lag of regressor j is obtained as

$$\frac{\kappa_1}{l^2} \text{ for own lags of endogenous variables,}$$

$$\frac{\kappa_1 \kappa_2}{l^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for endogenous variables other than own lags,}$$

$$\frac{\kappa_1 \kappa_3}{(l+1)^2} \frac{\sigma_i^2}{\sigma_j^2} \text{ for unmodelled exogenous variables,}$$

$$\kappa_1 \kappa_4 \sigma_i^2 \text{ for deterministic terms,}$$

where σ_i is the residual standard deviation of variable i of an unrestricted LS estimate. For exogenous variables σ_i is the sample standard deviation. In case structural parameters are estimated, the formula $\kappa_1 \kappa_2 \frac{\sigma_i^2}{\sigma_j^2}$ is used. If `kappa3` is not provided, prior variances are calculated in the same way as for deterministic terms. If the model does not contain exogenous variables, argument `kappa3` will be ignored.

The function only provides priors for the non-cointegration part of the model. The residual standard errors σ_i are based on an unrestricted LS regression of the endogenous variables on the error correction term and the non-cointegration regressors.

Value

A list containing a matrix of prior means and the precision matrix of the coefficients and the inverse variance-covariance matrix of the error term, which was obtained by an LS estimation.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias, J. L. (2019). *Bayesian Econometric Methods* (2nd ed.). Cambridge: University Press.

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Examples

```
# Load data
data("e6")

# Generate model input
object <- create_bvecmodel(e6, r = 0:2, p = 1:4,
                           const = "unrestricted",
                           seasonal = "unrestricted")

# Obtain Minnesota prior
prior <- minnesota_prior(object)
```

minnesota_prior.modellist *Minnesota Prior*

Description

Forwards the elements of an object of class 'modellist' forward to its respective method.

Usage

```
## S3 method for class 'modellist'
minnesota_prior(object, ...)
```

Arguments

object	a list of class 'modellist'.
...	further arguments passed to or from other methods.

Value

A list of class 'modellist'.

multipliers *Dynamic Multipliers*

Description

A generic function used to calculate the dynamic multipliers of a model with weakly exogenous variables. The function invokes particular methods which depend on the class of the first argument.

Usage

```
multipliers(x, ...)
```

Arguments

x	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Details

A dynamic multiplier is the response of the endogenous variables of a model to a change in one of its weakly exogenous variables. It is what an impulse response is for a shock to an error term, for a variable the model does not explain: the exogenous variable is moved by hand and the endogenous ones are followed.

The quantity is central to models whose foreign block is weakly exogenous, such as the country models of a global VAR (Pesaran, Schuermann and Weiner, 2004), where the responses to a change in the foreign variables are what the country model has to say about the rest of the world without the rest of the world being solved.

Value

The value returned by the method for the class of `x`, as described on the pages of the methods.

References

Pesaran, M. H., Schuermann, T., Weiner, S. M. (2004). Modeling regional interdependencies using a global error-correcting macroeconometric model. *Journal of Business & Economic Statistics, 22*(2), 129-162.

See Also

Methods: `multipliers.bvarmodel`, `multipliers.bvecmodel`.

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

`multipliers.bvarmodel` *Dynamic Multipliers of a VAR Model with Exogenous Variables*

Description

Computes the response of an endogenous variable to a change in a weakly exogenous variable of an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
multipliers(
  x,
  impulse = NULL,
  response = NULL,
  n_ahead = 5,
  ci = 0.95,
```

```
  shock = 1,
  type = "permanent",
  cumulative = FALSE,
  keep_draws = FALSE,
  period = NULL,
  ...
)
```

Arguments

x	an object of class 'bvarmodel' with at least one exogenous variable.
impulse	name of the exogenous variable that is moved.
response	name of the endogenous variable whose response is returned.
n_ahead	number of steps ahead. Zero is allowed and returns the impact response alone.
ci	a numeric between 0 and 1 specifying the probability mass covered by the credible intervals. Defaults to 0.95.
shock	size of the change in the exogenous variable, in its own units. Defaults to one.
type	"permanent" (default) for a change that is held from period zero on, or "transitory" for a change in period zero alone. See 'Details'.
cumulative	logical specifying whether the responses should be cumulated.
keep_draws	logical specifying whether the function should return all draws of the posterior multipliers. Defaults to FALSE, so that the median and the credible intervals of the posterior draws are returned.
period	integer. Index of the period the coefficients are taken from. Only used for models with time varying parameters. Defaults to NULL, so that the draws of the last period are used.
...	further arguments passed to or from other methods.

Details

For the model

$$y_t = \sum_{l=1}^{p} A_l y_{t-l} + \sum_{j=0}^{s} B_j x_{t-j} + C d_t + u_t,$$

where x_t is weakly exogenous, the dynamic multipliers M_h are obtained from the recursion

$$M_h = \sum_{l=1}^{\min(h,p)} A_l M_{h-l} + \sum_{j=0}^{\min(h,s)} B_j e,$$

with $M_h = 0$ for $h < 0$ and e the unit vector that selects the exogenous variable in impulse. The sum over B_j is what makes the change permanent: the variable is one unit higher from period zero on, so every lag of it that has come into range contributes. With type = "transitory" the variable is one unit higher in period zero alone, the sum collapses to B_h, and the response returns to zero unless the endogenous block has a unit root.

The deterministic terms and the errors play no part: a multiplier is a difference between two paths of the same model that differ only in the exogenous variable, so everything both paths share cancels.

The responses are in the units of the endogenous variables per unit of the exogenous one, and they are computed draw by draw, so what is returned is a posterior distribution of multipliers rather than a point estimate of them.

Whether the multipliers settle down is a property of the endogenous block alone. If the companion matrix of the A_l has a root on or outside the unit circle – which is the normal case for a model in levels whose variables are integrated – a permanent change moves the endogenous variables permanently, and the multipliers converge to a level rather than to zero.

Value

A time-series object of class 'bvarirf', which is what `irf` returns, so that the same `plot` method applies.

References

Pesaran, M. H., Schuermann, T., Weiner, S. M. (2004). Modeling regional interdependencies using a global error-correcting macroeconometric model. *Journal of Business & Economic Statistics, 22*(2), 129-162.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

Examples

```
data("e1")
e1 <- diff(log(e1)) * 100

# Investment and income as endogenous, consumption as weakly exogenous
model <- create_bvarmodel(data = e1[, c("invest", "income")],
                          exogen = e1[, "cons", drop = FALSE],
                          p = 2, s = 1, deterministic = "const",
                          iterations = 100, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))
model <- add_posterior_coefficients(add_initial_values(model))

dm <- multipliers(model, impulse = "cons", response = "income", n_ahead = 8)
plot(dm)
```

`multipliers.bvecmodel` *Dynamic Multipliers of a VEC Model with Exogenous Variables*

Description

Computes the response of an endogenous variable to a change in a weakly exogenous variable of an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
multipliers(x, ...)
```

Arguments

`x`	an object of class 'bvecmodel' with at least one exogenous variable.
`...`	arguments of `multipliers.bvarmodel`, which does the work.

Details

A multiplier is a statement about the levels of the endogenous variables, so the model is put into its levels form with `vec_to_var` and the multipliers of that form are returned. The transformation is exact, and it is where the error correction term does its work: the long-run relations enter the levels coefficients, so a change in an exogenous variable that belongs to a cointegrating relation moves the endogenous variables permanently, while at rank zero it moves them only through the short-run terms.

The responses are therefore in the units of the levels of the endogenous variables, whatever the error correction form has on its left-hand side.

Value

A time-series object of class 'bvarirf', which is what `irf` returns, so that the same `plot` method applies.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`, `vec_to_var.bvecmodel()`

Examples

```
data("e6")
set.seed(1)
exogen <- ts(cbind(g = as.numeric(e6[, "R"]) * 0.3 + rnorm(nrow(e6), sd = 0.1)),
             start = start(e6), frequency = frequency(e6))
```

```
model <- create_bvecmodel(data = e6, exogen = exogen, p = 2, s = 1, r = 1,
                          const = "unrestricted",
                          iterations = 100, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0), coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = 3, scale = 0.0001))
model <- add_posterior_coefficients(add_initial_values(model))

dm <- multipliers(model, impulse = "g", response = "R", n_ahead = 8)
```

---

open_model *Open a Model Stored in an HDF5 File*

Description

Opens a model written with write_to_hdf5 and returns a handle to it, which the analysis functions work through a piece of the chain at a time rather than reading every draw into the session.

Usage

```
open_model(filename, group = "")

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

Arguments

filename	path to an HDF5 file holding a model.
group	the group the model's tree hangs under inside its file, as read_model_from_hdf5 takes it. Defaults to "", the root.
x	an object of class 'bvarfile'.
...	further arguments passed to or from other methods.

Details

The draws of a model are what it is large in. A model with time varying coefficients keeps a path per draw, and a global model solved from many sub-models keeps a square matrix per lag and draw, so a chain that a posterior summary is comfortable with is a chain an analysis cannot hold beside everything else it needs.

A handle carries what a model is – its specification, its data and its priors – and the length of its chain, but none of the draws. The methods for it read the draws in pieces and keep only what they are asked for: the responses of an impulse response, the shares of a variance decomposition. What they return is what the same call on the model in memory returns.

Value

A list of class 'bvarfile' with the elements `filename`, `group`, `model`, `data`, `priors` and `draws`, the number of draws in the file.

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `choose_best_model.selcri`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `map_models()`, `open_models()`, `plot_forecast_errors_by_period()`, `pool_forecasts()`, `selection_criteria.bvarmodel()`, `selection_criteria.bvecmodel()`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Number of iterations and burn-in should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    sigma = list(df = 1, scale = .0001))
model <- add_posterior_coefficients(add_initial_values(model))

file <- file.path(tempdir(), "bvartools-example-model.h5")
unlink(file)
write_to_hdf5(model, filename = file)

stored <- open_model(file)
stored

# The analysis reads the draws in pieces
ir <- irf(stored, impulse = "invest", response = "income", n_ahead = 5,
          chunk = 25)
```

`open_models` *Open a Folder of Stored Models*

Description

Opens a folder written with `write_to_hdf5` and returns a handle to the models in it, which the estimation steps work on one at a time rather than reading all of them into the session.

Usage

```
open_models(folder)

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

model_files(x, models = NULL)
```

Arguments

folder	path to a folder holding models in HDF5 files.
x	an object of class 'bvarfolder'.
...	further arguments passed to or from other methods.
models	character vector of the models to name, as the manifest names them, or NULL for all of them.

Details

read_models_from_folder reads a folder into a list, draws and all, which is what to do with models that fit. Many do not. A lag and rank grid is dozens of models, an expanding window is one per quarter, and a model with time varying coefficients keeps a coefficient path per draw, so a folder can be tens of gigabytes while each model in it is a few hundred megabytes.

A handle carries what the folder holds – a row per model, saying where it is and what it is – and none of the draws. map_models and the estimation steps for a handle then read one model, change it, write it back and drop it, so a step costs one model per worker rather than the whole folder. open_model is the same idea for one model whose chain is too long to hold: a handle to a file, read a piece at a time.

The manifest is built by reading the specification of every model, which is a set of attributes rather than any of its data, so opening a folder is cheap. A model is named by where its file sits below folder, without the extension, and by its group where a file holds more than one – the names read_models_from_folder gives.

model_files names the files the models lie in, which is what a program that works on the files is pointed at. With bayests_files a folder is estimated as

```
run <- bayests_files(executable = "/opt/bayests/bin/bayests")
run(model_files(stored), jobs = 6)
```

and the draws never pass through R. Files are named once however many models they hold, since BayesTS works through every model in a file it is given.

Value

A list of class 'bvarfolder' with the elements folder and manifest, a data frame with the columns model, file, group and the specification of each model.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("e1")
train <- diff(log(e1)) * 100

models <- create_bvarmodel(data = train, p = 1:3, deterministic = "const",
                           iterations = 20, burnin = 10)

folder <- file.path(tempdir(), "bvartools-example-folder")
unlink(folder, recursive = TRUE)
dir.create(folder, recursive = TRUE)
write_to_hdf5(models, folder = folder)

stored <- open_models(folder)
stored
stored[["manifest"]][, c("model", "p", "iterations")]
```

---

persistence_profiles *Persistence Profiles*

Description

A generic function used to calculate the persistence profiles of the cointegrating relations of a model.

Usage

```
persistence_profiles(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of object, as described on the pages of the methods.

References

Pesaran, M. H., & Shin, Y. (1996). Cointegration and speed of convergence to equilibrium. *Journal of Econometrics, 71*(1-2), 117–143. doi:10.1016/03044076(94)016976

See Also

Methods: `persistence_profiles.bvecmodel`.

`persistence_profiles.bvarmodel`

Persistence Profiles of a VEC Model

Description

Calculates the persistence profiles of the cointegrating relations of an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvarmodel'
persistence_profiles(object, ...)

## S3 method for class 'bvecmodel'
persistence_profiles(
  object,
  n_ahead = 20,
  ci = 0.95,
  keep_draws = FALSE,
  period = NULL,
  ...
)

## S3 method for class 'modellist'
persistence_profiles(object, ...)

## S3 method for class 'expandingwindow'
persistence_profiles(object, ...)
```

Arguments

`object`	an object of class 'bvecmodel', usually, the result of a call to `add_posterior_coefficients`.
`...`	further arguments passed to or from other methods.
`n_ahead`	the number of horizons the profile is followed over.
`ci`	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
`keep_draws`	logical. If `FALSE` (default) the draws are summarised by their median and the band; if `TRUE` they are returned as they are.

`period` the period of a model with time varying coefficients the profile is calculated at. Defaults to `NULL`, the last period. Ignored for a model whose coefficients are constant.

Details

The persistence profile of Pesaran and Shin (1996) is the response of a cointegrating relation to a system-wide shock, scaled to one on impact:

$$PP_j(h) = \frac{\beta_j' \Psi_h \Sigma \Psi_h' \beta_j}{\beta_j' \Sigma \beta_j},$$

where Ψ_h are the moving average coefficients of the level VAR the model implies, $\Psi_0 = I$, and Σ is the covariance of its errors. It starts at one by construction.

A profile that does not fall to zero says the relation is not cointegrating, whatever the rank says. That is what the statistic is for: a rank chosen by a test or a criterion is an assertion about how many stationary combinations exist, and the profile is the check on it. How quickly the profile falls is the speed of convergence to equilibrium.

Unlike `irf` and `fevd`, which a VEC model reaches through `vec_to_var`, this statistic cannot be taken from the level VAR alone: `vec_to_var` drops `beta`, and the profile is a statement about the cointegrating vectors themselves. The level form is used for the moving average coefficients and the model's own `beta` for the relations.

For a model with weakly exogenous variables the profile is partial. A cointegrating vector of a VECX model spans the domestic variables and the weakly exogenous ones, and only the domestic block responds to a shock to this model. The relation is therefore followed over the part of it this model governs, which is the whole relation only where the model has no `exogen`. Where the weakly exogenous variables are endogenous to a larger system – a global model assembled from sub-models – take the profile there instead, and the function warns to that effect.

Value

A list with one element per cointegrating relation, each a matrix with one row per horizon and, unless `keep_draws` is `TRUE`, the columns `median`, `lower` and `upper`.

References

Pesaran, M. H., & Shin, Y. (1996). Cointegration and speed of convergence to equilibrium. *Journal of Econometrics, 71*(1-2), 117–143. doi:10.1016/03044076(94)016976

See Also

`persistence_profiles` for the generic.

Examples

```
# Load data
data("e6")

# Create model
```

```
model <- create_bvecmodel(e6, p = 2, r = 1, const = "unrestricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- add_priors(model,
                    coef = list(v_i = 0, v_i_det = 0),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 0.0001))

model <- add_initial_values(model)
model <- add_posterior_coefficients(model)

profiles <- persistence_profiles(model, n_ahead = 12)
round(profiles[[1]][1:5, ], 3)
```

`plot.bvarfevd` *Plotting Forecast Error Variance Decompositions of Bayesian Vector Autoregression*

Description

A plot function for objects of class "bvarfevd".

Usage

```
## S3 method for class 'bvarfevd'
plot(x, max_groups = NULL, ...)
```

Arguments

`x`	an object of class "bvarfevd", usually, a result of a call to `fevd`.
`max_groups`	integer. Maximum number of variables shown in the plot. The `max_groups - 1` variables with the largest contributions across the whole horizon are kept and the contributions of the remaining variables are added up in a further bar segment named `"Other"`, so that the legend does not become too large. Default is `NULL`, so that a segment is shown for every variable of the decomposition.
`...`	further graphical parameters.

Value

`x`, invisibly. The function is called for its side effect, the plot.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Obtain FEVD
vd <- fevd(object, response = "cons")

# Plot
plot(vd)
```

---

plot.bvarhd *Plot a Historical Decomposition*

Description

Stacked bars of the contributions of the shocks in each period, and the part of the response they account for as a line.

Usage

```
## S3 method for class 'bvarhd'
plot(x, baseline = FALSE, ...)
```

Arguments

x	an object of class "bvarhd", the result of historical_decomposition.
baseline	logical: should the baseline be stacked with the shocks? Defaults to FALSE, which plots the response net of the baseline – the part the shocks explain.
...	further arguments passed to barplot.

Value

x, invisibly.

---

plot.bvarirf *Plotting Impulse Responses of Bayesian Vector Autoregression*

Description

A plot function for objects of class "bvarirf".

Usage

```
## S3 method for class 'bvarirf'
plot(x, ...)
```

Arguments

x	an object of class "bvarirf", usually, a result of a call to irf.
...	further graphical parameters.

Value

x, invisibly. The function is called for its side effect, the plot.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Calculate IR
ir <- irf(object, impulse = "invest", response = "cons")
```

```
# Plot IR
plot(ir)
```

`plot.bvarmodel` *Plotting Draws of a Bayesian VAR Model*

Description

A plot function for objects of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
plot(x, ci = 0.95, type = "hist", show_zero_y = TRUE, max_cols = 6, ...)
```

Arguments

`x`	an object of class 'bvarmodel'.
`ci`	interval used to calculate credible bands for time-varying parameters.
`type`	either `"hist"` (default) for histograms, `"trace"` for a trace plot or `"boxplot"` for a boxplot. Only used for parameter draws of constant coefficients.
`show_zero_y`	if `TRUE` (default), a horizontal line with y = 0 is added to the plot. Only used for time varying parameters.
`max_cols`	an integer of the maximum number of regressors per figure. A block with more regressors than this is drawn as several figures of nearly equal width. Defaults to 6.
`...`	further graphical parameters.

Details

The function draws one figure per block of coefficients – the lags of the endogenous variables, the exogenous variables, the deterministic terms, the contemporaneous endogenous variables of a structural model, and the covariance matrix of the error term – instead of one figure for the whole model. A model with many regressors would otherwise produce panels too small to read.

Value

A plot per block of coefficients.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Plot
plot(model, type = "hist")
plot(model, type = "trace")
plot(model, type = "boxplot")
```

`plot.bvarprd` *Plotting Forecasts of BVAR Models*

Description

A plot function for objects of class 'bvarprd'.

Usage

```
## S3 method for class 'bvarprd'
plot(x, n_pre = NULL, ci = 0.95, ...)
```

Arguments

`x`	an object of class 'bvarprd', usually, a result of a call to `predict.bvarmodel`.
`n_pre`	number of plotted observations that precede the forecasts. If `NULL` (default), all available observations will be plotted.
`ci`	interval used to calculate the credible bands of the forecasts.
`...`	further graphical parameters. Arguments `main`, `ylab`, `lty` and `plot.type` overwrite the defaults of the function.

Value

x, invisibly. The function is called for its side effect, the plot.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add the data the forecasts are produced from
model <- add_forecast_input(model, n_ahead = 10)

# Simulate the forecasts
model <- add_posterior_forecasts(model)

# Collect them
pred <- predict(model)

# Plot forecasts
plot(pred)
```

---

plot.bvarprdlist *Plotting Forecasts of a List of Bayesian VAR Models*

Description

A plot function for objects of class 'bvarprdlist'.

Usage

```
## S3 method for class 'bvarprdlist'
plot(x, n_pre = NULL, ci = 0.95, ...)
```

Arguments

x	an object of class 'bvarprdlist', usually, a result of a call to predict.modellist.
n_pre	number of plotted observations that precede the forecasts. If NULL (default), all available observations will be plotted.
ci	interval used to calculate the credible bands of the forecasts.
...	further graphical parameters, which are passed on to plot.ts.

Details

The forecasts of the models in x are plotted on a grid, where each column corresponds to a model and each row to an endogenous variable.

Value

x, invisibly. The function is called for its side effect, the plot of the forecasts.

---

plot.bvarspillover *Plot a Spillover Index*

Description

Plots the directional connectedness measures of Diebold and Yilmaz (2012).

Usage

```
## S3 method for class 'bvarspillover'
plot(x, which = "net", ...)
```

Arguments

x	an object of class 'bvarspillover', usually the result of a call to spillover.
which	which measure to plot. One of "net" (default), "to" or "from".
...	further arguments passed to barplot.

Details

One bar per variable, with the credible interval drawn on it where the object carries one. Net values above zero mark variables that transmit more forecast error variance than they receive.

Value

NULL, invisibly. Called for its side effect.

plot.bvarspilloverts *Plot a Rolling Spillover Index*

Description

Plot a Rolling Spillover Index

Usage

```
## S3 method for class 'bvarspilloverts'
plot(x, ...)
```

Arguments

x	an object of class 'bvarspilloverts'.
...	further arguments passed to plot.ts.

Value

NULL, invisibly. Called for its side effect.

plot.bvecmodel *Plotting Draws of a Bayesian VEC Model*

Description

A plot function for objects of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
plot(x, ci = 0.95, type = "hist", show_zero_y = TRUE, max_cols = 6, ...)
```

Arguments

x	an object of class 'bvecmodel'.
ci	interval used to calculate credible bands for time-varying parameters.
type	either "hist" (default) for histograms, "trace" for a trace plot or "boxplot" for a boxplot. Only used for parameter draws of constant coefficients.
show_zero_y	if TRUE (default), a horizontal line with y = 0 is added to the plot. Only used for time varying parameters.
max_cols	an integer of the maximum number of regressors per figure. A block with more regressors than this is drawn as several figures of nearly equal width. Defaults to 6.
...	further graphical parameters.

Details

The coefficients of the error correction term are displayed as draws of the cointegration matrix $\Pi = \alpha\beta'$ and not as draws of the loading matrix α and the cointegration matrix β separately. The latter two are only identified up to a rotation, so their individual draws are not informative, while their product is.

The function draws one figure per block of coefficients – the cointegration matrix, the lagged differenced endogenous variables, the differenced exogenous variables, the deterministic terms, the contemporaneous endogenous variables of a structural model, and the covariance matrix of the error term – instead of one figure for the whole model. A model with many regressors would otherwise produce panels too small to read.

Value

A plot per block of coefficients.

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Create model
model <- create_bvecmodel(e6, p = 2, r = 1, const = "restricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Plot
plot(model, type = "hist")
plot(model, type = "trace")
plot(model, type = "boxplot")
```

`plot.expandwindbvarprdlist`

Plotting Forecasts of a List of Bayesian VAR Models

Description

A plot function for objects of class 'expandwindbvarprdlist'.

Usage

```
## S3 method for class 'expandwindbvarprdlist'
plot(x, n_pre = NULL, ci = 0.95, ...)
```

Arguments

x	an object of class 'expandwindbvarprdlist', usually, a result of a call to predict.expandingwindow.
n_pre	number of plotted observations that precede the forecasts. If NULL (default), all available observations will be plotted.
ci	interval used to calculate the credible bands of the forecasts.
...	further graphical parameters, which are passed on to plot.ts.

Value

x, invisibly. The function is called for its side effect, the plot of the forecasts.

---

plot.modellist *Plotting Draws of a Bayesian Time Series Models*

Description

A plot function for objects of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
plot(x, ...)
```

Arguments

x	an object of class 'modellist'.
...	arguments passed forward to other methods.

Value

x, invisibly. The function is called for its side effect, one plot per model in the list, drawn by the plot method of that model.

`plot.selcritlist` *Plotting Selection Criteria*

Description

A plot function for objects of class 'selcritlist' and 'selcrit'.

Usage

```
## S3 method for class 'selcritlist'
plot(x, criterion = "WAIC", ...)

## S3 method for class 'selcrit'
plot(x, criterion = "WAIC", ...)
```

Arguments

`x`	an object of class 'selcritlist' or 'selcrit', usually, a result of a call to `selection_criteria`.
`criterion`	the selection criterion that should be plotted. Available choices are the in-sample criteria `"LL"`, `"AIC"`, `"BIC"`, `"HQ"`, `"WAIC"` (default), `"LOOIC"`, the log marginal likelihood `"LML"` of a discounted model, the log predictive likelihood `"LPL"` of expanding windows and the out-of-sample statistics `"FE"`, `"AFE"` and `"RSFE"`.
`...`	further graphical parameters.

Details

For in-sample criteria each model is represented by a horizontal error bar, which is marked by the value of the criterion. The log-likelihood has a posterior distribution, so its bar covers the credible band and is marked by the median as well as the mean of its draws. `"WAIC"`, `"LOOIC"` and `"LPL"` are point estimates whose bars cover a normal interval built from their standard error. The interval of `"LPL"` is the one of `selection_criteria`, around the sum of the log predictive densities of the evaluated periods; `plot_predictive_loglik` plots the terms of that sum, which is where a difference between two models can be seen to be steady or to rest on a few periods. `"AIC"`, `"BIC"`, `"HQ"` and `"LML"` are point estimates without a standard error, so they are drawn as a single point. The default is `"WAIC"`, for the reasons given in `selection_criteria`. The criterion is measured on the x-axis and the models are arranged along the y-axis, beginning with the first model of `x` at the top. This keeps the plot readable, if `x` contains many models. The position of a model in `x` is added at the upper end of its error bar.

The specifications of the models are used as labels of the y-axis, where only the specifications that differ across the models are used, as in `print.modellist`. The lag order is always used. The left margin is adjusted to the width of the labels.

No title is added by default. Space above the plot is only used, if argument `main` is specified.

For out-of-sample statistics each row of the plot corresponds to an endogenous variable and the x-axis to the forecast horizon. Within a forecast horizon each model is represented by an error bar, which covers the credible band of the respective statistic and which is marked by the median and the mean of its posterior draws. The position of a model in `x` is added above its error bar. Since forecast

errors can be positive and negative, a horizontal reference line is added at zero for `criterion = "FE"`.

For both types of criteria arguments `col`, `pch`, `cex`, `lwd`, `main` and `xlab` can be used to change the appearance of the plot, where `col` and `lwd` are recycled over the models in `x`.

A 'selcrit' – the criteria of one model, or those of the windows of an 'expandingwindow' pooled into one – is drawn as a 'selcritlist' of that one model.

Value

`x`, invisibly. The function is called for its side effect, the plot.

`plot_forecast_errors_by_period`

Plotting Forecast Errors per Period

Description

Plots the forecast errors of a list of Bayesian models as error bars for each period, for which a forecast was made.

Usage

```
plot_forecast_errors_by_period(
  x,
  criterion = "AFE",
  ci = 0.95,
  col = "black",
  pch = 20,
  cex = 1,
  lwd = 1,
  main = NULL,
  xlab = "Period",
  ...
)
```

Arguments

`x`	an object of class 'bvarmodel', 'bvecmodel', 'expandingwindow' or 'modellist', which contains forecast errors. Usually, a result of a call to `add_forecast_errors`.
`criterion`	the statistic that should be plotted. Available choices are `"FE"` for forecast errors, `"AFE"` (default) for absolute forecast errors and `"RSFE"` for root squared forecast errors. The mean of `"RSFE"` is the root mean squared error; its median and band are those of `"AFE"` up to the interpolation between neighbouring draws, because the square root does not reorder them.
`ci`	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.

col	a vector of colours, which is recycled over the models in x.
pch	the plotting symbol used for the mean of a model.
cex	the size of the plotting symbol used for the mean of a model.
lwd	a vector of line widths, which is recycled over the models in x.
main	the title of the plot. If NULL (default), no title is added.
xlab	the label of the x-axis. If NULL, no label is added.
...	further graphical parameters, which are passed on to plot.

Details

Each row of the plot corresponds to an endogenous variable, each column to a forecast horizon and the x-axis to the period, for which a forecast was made. Within a period each model is represented by an error bar, which covers the credible band of the respective statistic and which is marked by the median and the mean of the posterior draws of its forecast errors. The position of a model in x is added above its error bar. For criterion = "FE" a horizontal reference line is added at zero.

In contrast to plot.selcritlist, which plots statistics that are calculated across all periods of the evaluation sample, this function plots the statistics of each period separately. Thus, it is most useful for forecast errors, which were obtained for multiple periods as, for example, in the case of expanding window estimation. For objects of class 'expandingwindow' the forecast errors of all estimation windows of a model are combined, where the period of an error is obtained from the end of the training sample of the respective window.

Value

x, invisibly. The function is called for its side effect, the plot.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
data("us_macrodata")

# AR(1) models as benchmark
model <- create_bvarmodel(data = us_macrodata,
                          p = 1:2,
                          deterministic = "none",
                          error = "gamma",
                          iterations = 10,
                          burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.
```

```
# Obtain objects for expanding window estimation
model <- use_expanding_window(model, start = 2007)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(shape = 3, rate = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = us_macrodata)

# Plot absolute forecast errors of each period
plot_forecast_errors_by_period(model)
```

`plot_predictive_loglik` *Plotting the Log Predictive Likelihood per Period*

Description

Plots the log predictive density of each evaluated period for one or more Bayesian models, either period by period or as a running sum.

Usage

```
plot_predictive_loglik(
  x,
  baseline = NULL,
  cumulative = TRUE,
  col = "black",
  lwd = 1,
  lty = 1,
  legend = TRUE,
  legend_position = "topleft",
  main = NULL,
  xlab = "Period",
```

```
  ylab = NULL,
  ...
)
```

Arguments

x	an object of class 'selcrit' or 'selcritlist' whose entries contain the log predictive likelihood, usually a result of a call to selection_criteria on expanding windows that carry the draws of add_predictive_loglik.
baseline	the model every other model is compared with, given as its position in x or as its name. If NULL (default), nothing is subtracted and the log predictive densities themselves are plotted.
cumulative	logical. Should the running sum over the periods be plotted (TRUE, the default) rather than the value of each period?
col	a vector of colours, which is recycled over the models in x.
lwd	a vector of line widths, which is recycled over the models in x.
lty	a vector of line types, which is recycled over the models in x.
legend	logical. Should a legend of the models be added?
legend_position	the position of the legend, passed on to legend. The default is "topleft", which a cumulative plot of differences often leaves free; where the lines rise into it, "bottomleft" or "right" usually does not.
main	the title of the plot. If NULL (default), no title is added.
xlab	the label of the x-axis.
ylab	the label of the y-axis. If NULL (default), a label is chosen from baseline and cumulative.
...	further graphical parameters, which are passed on to plot.

Details

The log predictive likelihood of a model is the sum of the log predictive densities of the periods it was evaluated for, and selection_criteria reports that sum with a band. This function plots the terms of the sum, which answers a question the sum cannot: whether a difference between two models is a steady advantage or the consequence of a few periods. A single quarter can decide a comparison of this kind, and with cumulative = TRUE such a quarter is a step in an otherwise flat line rather than a number hidden in a total.

With a baseline **the plot shows differences**, the cumulative sum of $lpd_t - lpd_t^{baseline}$ over the evaluated periods, and a horizontal reference line is drawn at zero. A line that ends above zero belongs to a model that predicted better than the baseline over the whole evaluation sample, and the slope at any point is how much better it was predicting then. Differences are what the criterion is read in, so a baseline is usually what is wanted as soon as there is more than one model; without one the lines are dominated by the periods that were hard for every model alike.

The periods come from attribute "terms" of the LPL entry of each model, which also holds the numerical standard error of each period. Models are plotted over the periods they share with the baseline, and a model without an LPL entry is left out with a warning.

Value

x, invisibly. The function is called for its side effect, the plot.

See Also

selection_criteria for the sum and its band, add_predictive_loglik for the draws it is computed from, and plot_forecast_errors_by_period for the same question asked of forecast errors.

Examples

```
data("us_macrodata")

# Two specifications to compare
models <- create_bvarmodel(data = us_macrodata, p = 1:2,
                           deterministic = "const",
                           iterations = 100, burnin = 50)
models <- add_priors(models,
                     coef = list(v_i = 1, v_i_det = 1 / 10),
                     sigma = list(df = 3, scale = 1))

# One window per evaluated period
windows <- lapply(models, use_expanding_window, start = c(1972, 1))

criteria <- lapply(windows, function(w) {
  w <- add_initial_values(w)
  w <- add_posterior_coefficients(w)
  w <- add_predictive_loglik(w)
  selection_criteria(w)
})
class(criteria) <- append("selcritlist", class(criteria))

plot_predictive_loglik(criteria, baseline = 1)
```

---

pool_forecasts *Pool Forecasts*

Description

Combines the forecasts of several models into one equal-weight pool, which is compared with the models like any of them.

Usage

```
pool_forecasts(...)

## S3 method for class 'forecastpool'
add_priors(object, ...)
```

```
## S3 method for class 'forecastpool'
add_initial_values(object, ...)

## S3 method for class 'forecastpool'
add_seed(object, seed, ...)

## S3 method for class 'forecastpool'
add_posterior_coefficients(object, ...)

## S3 method for class 'forecastpool'
add_posterior_loglik(object, ...)

## S3 method for class 'forecastpool'
add_forecast_input(object, ...)

## S3 method for class 'forecastpool'
add_posterior_forecasts(object, ...)

## S3 method for class 'forecastpool'
add_predictive_loglik(object, ...)

## S3 method for class 'forecastpool'
thin(x, thin = 10, ...)

## S3 method for class 'poolwindow'
add_priors(object, ...)

## S3 method for class 'poolwindow'
add_initial_values(object, ...)

## S3 method for class 'poolwindow'
add_seed(object, seed, ...)

## S3 method for class 'poolwindow'
add_posterior_coefficients(object, ...)

## S3 method for class 'poolwindow'
add_posterior_loglik(object, ...)

## S3 method for class 'poolwindow'
add_forecast_input(object, ...)

## S3 method for class 'poolwindow'
add_posterior_forecasts(object, ...)

## S3 method for class 'poolwindow'
add_predictive_loglik(object, ...)
```

```
## S3 method for class 'poolwindow'
thin(x, thin = 10, ...)

## S3 method for class 'forecastpool'
write_to_hdf5(object, ...)

## S3 method for class 'poolwindow'
write_to_hdf5(object, ...)

## S3 method for class 'forecastpool'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'poolwindow'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`...`	two or more models with forecasts: objects of class `"expandingwindow"`, or single models of class `"bvarmodel"` or `"bvecmodel"`, or `"modellist"` objects of either. Arguments may be named; the names label the members of the pool.
`object`, `x`	an object of class `"forecastpool"` or `"poolwindow"`.
`seed`	not used, since a pool is not simulated.
`thin`	an integer specifying the thinning interval between successive draws.
`digits`	the number of significant digits.

Details

A pool of forecasts is a mixture of predictive distributions: the predictive distribution of the pool is the average of those of its members. Combining forecasts in this way is one of the most reliable ways of improving them, because the errors of different models are not perfectly correlated and no single specification is best in every period (Bates and Granger, 1969; Hall and Mitchell, 2007; Geweke and Amisano, 2011).

The pool is built from draws the members already hold. For every forecast, the same number of draws is taken at random from each member – as many as the member with the fewest draws has – and stacked, so that every member has the same weight. The draws are selected with R's random number generator, so `set.seed()` makes a pool reproducible.

The members must forecast the same thing. The pool contains the endogenous variables all members share, and the forecast horizons all of them reach. A variable forecast by only some of the members is left out, and a pool of models that share no variable is refused. A VEC model forecasts the levels of its variables, so it can be pooled with VAR models of the same levels, but not with VAR models of their growth rates. Forecasts aggregated to annual figures can only be pooled with forecasts aggregated in the same way.

The windows of expanding window exercises are matched by the end of their estimation samples, so that each pooled forecast combines forecasts made with the same information. Windows that not every member has are left out, with a message. Single models and expanding windows cannot be pooled with each other, and external forecasts, which are point forecasts rather than predictive

distributions, cannot be pooled at all: they are compared beside a pool, by combining both with `combine_models`.

The log predictive densities that `add_predictive_loglik` and the forecasts of BayesTS provide are pooled as the forecasts are, which gives the log of the average predictive density of the members – the score of the mixture, not of its draws. They are kept only if every member carries them and all members have the same endogenous variables, because a predictive density is a joint density of all the variables of a model.

The weights are equal. Weights that favour members which forecast better in the past must only use the periods before each forecast, or they flatter the pool with information it did not have.

A pool is formed after the members' forecasts are drawn, and the functions that estimate a model leave it unchanged. It can be passed to `add_forecast_errors` and `selection_criteria`, and included in a list of models with `combine_models`. It cannot be written to a file: write its members and form the pool again after reading them back.

A pool is formed from draws its members already hold, so the functions that add priors, initial values, posterior draws, forecasts or predictive log-likelihoods to a model return it unchanged. This allows to combine a pool with its members in a list of class `"modellist"` and to apply the usual workflow to all elements of that list.

Value

An object of class `"forecastpool"`, which behaves like an expanding window exercise, or, if single models were pooled, a single pooled forecast of class `"poolwindow"`. Each pooled forecast holds the draws of the pool in `posterior$forecast$forecasts` and, in `model`, the names of the members in `members` and the number of draws taken from each in `draws`.

References

Bates, J. M., & Granger, C. W. J. (1969). The combination of forecasts. *Journal of the Operational Research Society, 20*(4), 451–468. doi:10.1057/jors.1969.103

Geweke, J., & Amisano, G. (2011). Optimal prediction pools. *Journal of Econometrics, 164*(1), 130–141. doi:10.1016/j.jeconom.2011.02.017

Hall, S. G., & Mitchell, J. (2007). Combining density forecasts. *International Journal of Forecasting, 23*(1), 1–13. doi:10.1016/j.ijforecast.2006.08.001

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`, `aggregate_forecasts()`, `align_model_obs.modellist()`, `analysis_of_stored_models`, `choose_best_model.selcri`, `create_external_forecast()`, `folder_steps`, `map_draws()`, `map_models()`, `open_model()`, `open_models()`, `plot_forecast_errors_by_period()`, `selection_criteria.bvarmodel()`, `selection_criteria.bve`, `selection_criteria.default()`, `selection_criteria.modellist()`

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
```

```
train <- window(e1, end = c(1978, 4))

# Two specifications, each estimated over expanding windows
model <- create_bvarmodel(train, p = 1:2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burn-in should be much higher.
model <- add_priors(model,
                    coef = list(v_i = 1 / 10, v_i_det = 1 / 100),
                    sigma = list(df = "k", scale = 1))
model <- use_expanding_window(model, start = c(1978, 1))
model <- add_initial_values(model)
model <- add_posterior_coefficients(model)
model <- add_forecast_input(model, n_ahead = 2)
model <- add_posterior_forecasts(model)

# The equal-weight pool of both
set.seed(123)
pool <- pool_forecasts(p1 = model[[1]], p2 = model[[2]])

# Scored and compared like the models
all_models <- combine_models(model, pool)
all_models <- add_forecast_errors(all_models, test_sample = e1)
selection_criteria(all_models)
```

---

post_bvs *Bayesian Variable Selection*

Description

post_bvs employs Bayesian variable selection as proposed by Korobilis (2013) to produce a vector of inclusion parameters for the coefficient matrix of a VAR model.

Usage

```
post_bvs(y, z, a, k, m, lambda, sigma_i, prob_prior, include = NULL)
```

Arguments

y	a $KT \times 1$ vector of the endogenous variables.
z	a $KT \times M$ matrix of explanatory variables.
a	an M-dimensional vector of parameter draws. If time varying parameters are used, an $M \times T$ coefficient matrix can be provided.
k	integer of the number of endogenous variables.
m	integer of the number of M
lambda	an $M \times M$ inclusion matrix that should be updated.
sigma_i	a sparse $KT \times KT$ block diagonal matrix containing the inverse variance-covariance matrix.

prob_prior	an M-dimensional vector of prior inclusion probabilities.
include	an integer vector specifying the positions of variables, which should be included in the BVS algorithm. If NULL (default), BVS will be applied to all variables.

Details

The function employs Bayesian variable selection as proposed by Korobilis (2013) to produce a vector of inclusion parameters, which are the diagonal elements of the inclusion matrix Λ for the VAR model

$$y_t = Z_t \Lambda a_t + u_t,$$

where $u_t \sim N(0, \Sigma_t)$. y_t is a K-dimensional vector of endogenous variables and $Z_t = x_t' \otimes I_K$ is a $K \times M$ matrix of regressors with x_t as a vector of regressors.

Value

A matrix of inclusion parameters on its diagonal.

References

Korobilis, D. (2013). VAR forecasting using Bayesian variable selection. *Journal of Applied Econometrics, 28*(2), 204–230. doi:10.1002/jae.1271

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model input, which uses BVS as variable selection algorithm
object <- create_bvarmodel(data = e1, p = 2, deterministic = "const",
                           varsel = "bvs")

# Add prior specifications, including the prior inclusion probabilities
object <- add_priors(object,
                     coef = list(v_i = 1, v_i_det = 1 / 10),
                     sigma = list(df = "k", scale = 1),
                     varsel = list(inprior = .1))

# Add initial values
object <- add_initial_values(object)

# Obtain data, initial values and priors
y <- matrix(t(object[["data"]][["train"]][["y"]]))
z <- object[["data"]][["train"]][["z"]] # Argument 'z' is taken dense
k <- object[["model"]][["k"]]
tt <- nrow(object[["data"]][["train"]][["y"]])
m <- ncol(z)
a <- object[["initial"]][["a"]]
prob_prior <- object[["priors"]][["a"]][["inprior"]]

# Arguments 'lambda' and 'sigma_i' have to be sparse
```

```
lambda <- Matrix::Matrix(diag(1, m), sparse = TRUE)

# Initial value of the inverse error covariance matrix
u <- matrix(y - z %*% a, k)
sigma_i <- Matrix::Matrix(kronecker(diag(1, tt), solve(tcrossprod(u) / tt)),
                          sparse = TRUE)

# Draw inclusion parameters
post_bvs(y, z, a, k, m, lambda, sigma_i, prob_prior)
```

`post_coint_kls` *Posterior Draw for Cointegration Models*

Description

Produces a draw of coefficients for cointegration models with a prior on the cointegration space as proposed in Koop et al. (2010) and a draw of non-cointegration coefficients from a normal density.

Usage

```
post_coint_kls(
  y,
  beta,
  w,
  sigma_i,
  v_i,
  p_tau_i,
  g_i,
  x = NULL,
  gamma_mu_prior = NULL,
  gamma_v_i_prior = NULL
)
```

Arguments

`y`	a $K \times T$ matrix of differenced endogenous variables.
`beta`	a $M \times r$ cointegration matrix β.
`w`	a $M \times T$ matrix of variables in the cointegration term.
`sigma_i`	an inverse of the $K \times K$ variance-covariance matrix.
`v_i`	a numeric between 0 and 1 specifying the shrinkage of the cointegration space prior.
`p_tau_i`	an inverted $M \times M$ matrix specifying the central location of the cointegration space prior of $sp(\beta)$.
`g_i`	a $K \times K$ matrix.
`x`	a $N \times T$ matrix of differenced regressors and unrestricted deterministic terms.

`gamma_mu_prior` a $KN \times 1$ prior mean vector of non-cointegration coefficients.

`gamma_v_i_prior` an inverted $KN \times KN$ prior covariance matrix of non-cointegration coefficients.

Details

The function produces posterior draws of the coefficient matrices α, β and Γ for the model

$$y_t = \alpha\beta' w_{t-1} + \Gamma z_t + u_t,$$

where y_t is a K-dimensional vector of differenced endogenous variables. w_t is an $M \times 1$ vector of variables in the cointegration term, which include lagged values of endogenous and exogenous variables in levels and restricted deterministic terms. z_t is an N-dimensional vector of differenced endogenous and exogenous explanatory variabes as well as unrestricted deterministic terms. The error term is $u_t \sim \Sigma$.

Draws of the loading matrix α are obtained using the prior on the cointegration space as proposed in Koop et al. (2010). The posterior covariance matrix is

$$\overline{V}_\alpha = \left[\left(v^{-1}(\beta' P_\tau^{-1}\beta) \otimes G_{-1}\right) + \left(ZZ' \otimes \Sigma^{-1}\right)\right]^{-1}$$

and the posterior mean by

$$\overline{\alpha} = \overline{V}_\alpha + vec(\Sigma^{-1} Y Z'),$$

where Y is a $K \times T$ matrix of differenced endogenous variables and $Z = \beta' W$ with W as an $M \times T$ matrix of variables in the cointegration term.

For a given prior mean vector $\underline{\Gamma}$ and prior covariance matrix $\underline{V_\Gamma}$ the posterior covariance matrix of non-cointegration coefficients in Γ is obtained by

$$\overline{V}_\Gamma = \left[\underline{V}_\Gamma^{-1} + \left(XX' \otimes \Sigma^{-1}\right)\right]^{-1}$$

and the posterior mean by

$$\overline{\Gamma} = \overline{V}_\Gamma \left[\underline{V}_\Gamma^{-1}\underline{\Gamma} + vec(\Sigma^{-1} Y X')\right],$$

where X is an $M \times T$ matrix of explanatory variables, which do not enter the cointegration term.

Draws of the cointegration matrix β are obtained using the prior on the cointegration space as proposed in Koop et al. (2010). The posterior covariance matrix of the unrestricted cointegration matrix B is

$$\overline{V}_B = \left[\left(A' G^{-1} A \otimes v^{-1} P_\tau^{-1}\right) + \left(A' \Sigma^{-1} A \otimes WW'\right)\right]^{-1}$$

and the posterior mean by

$$\overline{B} = \overline{V}_B + vec(W Y_B^{-1} \Sigma^{-1} A),$$

where $Y_B = Y - \Gamma X$ and $A = \alpha(\alpha'\alpha)^{-\frac{1}{2}}$.

The final draws of α and β are calculated using $\beta = B(B'B)^{-\frac{1}{2}}$ and $\alpha = A(B'B)^{\frac{1}{2}}$.

Value

A named list containing the following elements:

`alpha`	a draw of the $K \times r$ loading matrix.
`beta`	a draw of the $M \times r$ cointegration matrix.
`Pi`	a draw of the $K \times M$ cointegration matrix $\Pi = \alpha\beta'$.
`Gamma`	a draw of the coefficients of the non-cointegration regressors in `x`, as the $KN \times 1$ vector $vec(\Gamma)$ of the $K \times N$ matrix; `matrix(Gamma, K)` restores the matrix. Empty if `x` is `NULL`.

References

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224-242. doi:10.1080/07474930903382208

Examples

```
# Load data
data("e6")

# Generate model data
temp <- create_bvecmodel(e6, p = 2, r = 1)
y <- t(temp$data$train$y)
ect <- t(temp$data$train$w)
x <- t(temp$data$train$x)

k <- nrow(y) # Endogenous variables
tt <- ncol(y) # Number of observations
r <- temp$model$rank # Cointegration rank

# Priors. They cover the alpha block as well as the Gamma coefficients,
# so the length is k * r + k * nrow(x).
n_ag <- k * (r + nrow(x))
gamma_mu_prior <- matrix(0, n_ag)
gamma_v_i_prior <- diag(0, n_ag)

# Initial value of Sigma
sigma <- tcrossprod(y) / tt
sigma_i <- solve(sigma)

# Initial values of beta
beta <- matrix(c(1, -4), nrow(ect))

# Draw parameters
coint <- post_coint_kls(y = y, beta = beta, w = ect, x = x, sigma_i = sigma_i,
                        v_i = 0, p_tau_i = diag(1, nrow(ect)), g_i = sigma_i,
                        gamma_mu_prior = gamma_mu_prior,
                        gamma_v_i_prior = gamma_v_i_prior)
```

post_coint_kls_sur *Posterior Draw for Cointegration Models*

Description

Produces a draw of coefficients for cointegration models in SUR form with a prior on the cointegration space as proposed in Koop et al. (2010) and a draw of non-cointegration coefficients from a normal density.

Usage

```
post_coint_kls_sur(
  y,
  beta,
  w,
  sigma_i,
  v_i,
  p_tau_i,
  g_i,
  x = NULL,
  gamma_mu_prior = NULL,
  gamma_v_i_prior = NULL,
  svd = FALSE
)
```

Arguments

y	a $K \times T$ matrix of differenced endogenous variables.
beta	a $M \times r$ cointegration matrix β, where $\beta'\beta = I$.
w	a $M \times T$ matrix of variables in the cointegration term.
sigma_i	the inverse of the constant $K \times K$ error variance-covariance matrix. For time varying variance-covariance matrics a $KT \times K$ can be provided.
v_i	a numeric between 0 and 1 specifying the shrinkage of the cointegration space prior.
p_tau_i	an inverted $M \times M$ matrix specifying the central location of the cointegration space prior of $sp(\beta)$.
g_i	a $K \times K$ or $KT \times K$ matrix. If the matrix is $KT \times K$, the function will automatically produce a $K \times K$ matrix containing the means of the time varying $K \times K$ covariance matrix.
x	a $KT \times NK$ matrix of differenced regressors and unrestricted deterministic terms.
gamma_mu_prior	a $KN \times 1$ prior mean vector of non-cointegration coefficients.
gamma_v_i_prior	an inverted $KN \times KN$ prior covariance matrix of non-cointegration coefficients.

svd — logical. If TRUE the singular value decomposition is used to determine the root of the posterior covariance matrix. Default is FALSE which means that the eigenvalue decomposition is used.

Details

The function produces posterior draws of the coefficient matrices α, β and Γ for the model

$$y_t = \alpha\beta' w_{t-1} + \Gamma z_t + u_t,$$

where y_t is a K-dimensional vector of differenced endogenous variables. w_t is an $M \times 1$ vector of variables in the cointegration term, which include lagged values of endogenous and exogenous variables in levels and restricted deterministic terms. z_t is an N-dimensional vector of differenced endogenous and exogenous explanatory variabes as well as unrestricted deterministic terms. The error term is $u_t \sim \Sigma$.

Draws of the loading matrix α are obtained using the prior on the cointegration space as proposed in Koop et al. (2010). The posterior covariance matrix is

$$\overline{V}_\alpha = \left[\left(v^{-1}(\beta' P_\tau^{-1}\beta) \otimes G_{-1}\right) + \left(ZZ' \otimes \Sigma^{-1}\right)\right]^{-1}$$

and the posterior mean by

$$\overline{\alpha} = \overline{V}_\alpha + vec(\Sigma^{-1} Y Z'),$$

where Y is a $K \times T$ matrix of differenced endogenous variables and $Z = \beta' W$ with W as an $M \times T$ matrix of variables in the cointegration term.

For a given prior mean vector $\underline{\Gamma}$ and prior covariance matrix $\underline{V_\Gamma}$ the posterior covariance matrix of non-cointegration coefficients in Γ is obtained by

$$\overline{V}_\Gamma = \left[\underline{V}_\Gamma^{-1} + \left(XX' \otimes \Sigma^{-1}\right)\right]^{-1}$$

and the posterior mean by

$$\overline{\Gamma} = \overline{V}_\Gamma \left[\underline{V}_\Gamma^{-1}\underline{\Gamma} + vec(\Sigma^{-1} Y X')\right],$$

where X is an $M \times T$ matrix of explanatory variables, which do not enter the cointegration term.

Draws of the cointegration matrix β are obtained using the prior on the cointegration space as proposed in Koop et al. (2010). The posterior covariance matrix of the unrestricted cointegration matrix B is

$$\overline{V}_B = \left[\left(A' G^{-1} A \otimes v^{-1} P_\tau^{-1}\right) + \left(A'\Sigma^{-1} A \otimes WW'\right)\right]^{-1}$$

and the posterior mean by

$$\overline{B} = \overline{V}_B + vec(W Y_B^{-1} \Sigma^{-1} A),$$

where $Y_B = Y - \Gamma X$ and $A = \alpha(\alpha'\alpha)^{-\frac{1}{2}}$.

The final draws of α and β are calculated using $\beta = B(B'B)^{-\frac{1}{2}}$ and $\alpha = A(B'B)^{\frac{1}{2}}$.

Value

A named list containing the following elements:

alpha — a draw of the $K \times r$ loading matrix.

beta — a draw of the $M \times r$ cointegration matrix.

Pi — a draw of the $K \times M$ cointegration matrix $\Pi = \alpha\beta'$.

Gamma — a draw of the coefficients of the non-cointegration regressors in x, as a column vector with one element per column of x. Empty if x is NULL.

References

Koop, G., León-González, R., & Strachan R. W. (2010). Efficient posterior simulation for cointegrated models with priors on the cointegration space. *Econometric Reviews, 29*(2), 224-242. doi:10.1080/07474930903382208

Examples

```
# Load data
data("e6")

# Generate model data
temp <- create_bvecmodel(e6, p = 1, r = 1)
y <- t(temp$data$train$y)
ect <- t(temp$data$train$w)

k <- nrow(y) # Endogenous variables
tt <- ncol(y) # Number of observations

# Initial value of Sigma
sigma <- tcrossprod(y) / tt
sigma_i <- solve(sigma)

# Initial values of beta
beta <- matrix(c(1, -4), k)

# Draw parameters
coint <- post_coint_kls_sur(y = y, beta = beta, w = ect,
                            sigma_i = sigma_i, v_i = 0, p_tau_i = diag(1, nrow(ect)),
                            g_i = sigma_i)
```

---

post_gamma_measurement_variance

Posterior Draws of Error Variances

Description

Produces a draw of the constant diagonal error variance matrix of the measurement equation of a state space model using an inverse gamma posterior density.

Usage

```
post_gamma_measurement_variance(u, shape_prior, rate_prior, inverse)
```

Arguments

u	a $KT \times 1$ vector of errors.
shape_prior	a $K \times 1$ vector of prior shape parameters.

rate_prior	a $K \times 1$ vector of prior rate parameters.
inverse	logical. If TRUE, the function returns the precision matrix, i.e. the inverse of the variance matrix, and otherwise the variance matrix.

Details

For a model with measurement equation

$$y_t = Z_t a_t + u_t$$

with $u_t \sim N(0, \Sigma_u)$ the function produces a draw of the constant diagonal error variance matrix Σ_u.

Value

A matrix.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Examples

```
k <- 10 # Number of endogenous variables
tt <- 1000 # Number of observations

set.seed(1234) # Set RNG seed

# Generate artificial error series with N(0, 1)
u <- matrix(rnorm(k * tt))

# Define priors
shape_prior <- matrix(1, k)
rate_prior <- matrix(.0001, k)

# Obtain posterior draw
post_gamma_measurement_variance(u, shape_prior, rate_prior, inverse = FALSE)
```

---

post_gamma_state_variance

Posterior Draws of Error Variances

Description

Produces a draw of the constant diagonal error variance matrix of the state equation of a state space model using an inverse gamma posterior density.

Usage

```
post_gamma_state_variance(a, a_init, shape_prior, rate_prior, inverse)
```

Arguments

a	a $KT \times 1$ vector of time varying parameter draws.
a_init	a $K \times 1$ vector of initial states.
shape_prior	a $K \times 1$ vector of prior shape parameters.
rate_prior	a $K \times 1$ vector of prior rate parameters.
inverse	logical. If TRUE, the function returns the precision matrix, i.e. the inverse of the variance matrix, and otherwise the variance matrix.

Details

For the state space model with state equation

$$a_t = a_{t-1} + v_t$$

and measurement equation

$$y_t = Z_t a_t + u_t$$

with $v_t \sim N(0, \Sigma^v)$ and $u_t \sim N(0, \Sigma^u)$ the function produces a draw of the constant diagonal error variances matrix of the state equation Σ^v.

Value

A matrix.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Examples

```
k <- 10 # Number of artificial coefficients
tt <- 1000 # Number of observations

set.seed(1234) # Set RNG seed

# Generate artificial data according to a random walk
a <- matrix(rnorm(k), k, tt + 1)
for (i in 2:(tt + 1)) {
  a[, i] <- a[, i - 1] + rnorm(k, 0, sqrt(1 / 100))
}

a_init <- matrix(a[, 1]) # Define initial state
a <- matrix(a[, -1]) # Drop initial state from main sample and make vector

# Define priors
```

```
shape_prior <- matrix(1, k)
rate_prior <- matrix(.0001, k)

# Obtain posterior draw
post_gamma_state_variance(a, a_init, shape_prior, rate_prior, inverse = FALSE)
```

`post_normal` *Posterior Draw from a Normal Distribution*

Description

Produces a draw of coefficients from a normal posterior density.

Usage

```
post_normal(y, x, sigma_i, a_prior, v_i_prior)
```

Arguments

`y`	a $K \times T$ matrix of endogenous variables.
`x`	an $M \times T$ matrix of explanatory variables.
`sigma_i`	the inverse of the $K \times K$ variance-covariance matrix.
`a_prior`	a $KM \times 1$ numeric vector of prior means.
`v_i_prior`	the inverse of the $KM \times KM$ prior covariance matrix.

Details

The function produces a vectorised posterior draw a of the $K \times M$ coefficient matrix A for the model

$$y_t = Ax_t + u_t,$$

where y_t is a K-dimensional vector of endogenous variables, x_t is an M-dimensional vector of explanatory variabes and the error term is $u_t \sim \Sigma$.

For a given prior mean vector $\underline{a}$ and prior covariance matrix $\underline{V}$ the posterior covariance matrix is obtained by

$$\overline{V} = \left[\underline{V}^{-1} + \left(XX' \otimes \Sigma^{-1}\right)\right]^{-1}$$

and the posterior mean by

$$\overline{a} = \overline{V}\left[\underline{V}^{-1}\underline{a} + vec(\Sigma^{-1}YX')\right],$$

where Y is a $K \times T$ matrix of the endogenous variables and X is an $M \times T$ matrix of the explanatory variables.

Value

A vector.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Examples

```
# Load data
data("e1")
data <- diff(log(e1))

# Generate model data
temp <- create_bvarmodel(data, p = 2, deterministic = "const")
y <- t(temp$data$train$y)
x <- t(temp$data$train$x)
k <- nrow(y)
tt <- ncol(y)
m <- k * nrow(x)

# Priors
a_mu_prior <- matrix(0, m)
a_v_i_prior <- diag(0.1, m)

# Initial value of inverse Sigma
sigma_i <- solve(tcrossprod(y) / tt)

# Draw parameters
a <- post_normal(y = y, x = x, sigma_i = sigma_i,
                 a_prior = a_mu_prior, v_i_prior = a_v_i_prior)
```

---

post_normal_sur *Posterior Draw from a Normal Distribution*

Description

Produces a draw of coefficients from a normal posterior density for a model with seemingly unrelated regresssions (SUR).

Usage

```
post_normal_sur(y, z, sigma_i, a_prior, v_i_prior, svd = FALSE)
```

Arguments

y	a $K \times T$ matrix of endogenous variables.
z	a $KT \times M$ matrix of explanatory variables.
sigma_i	the inverse of the constant $K \times K$ error variance-covariance matrix. For time varying variance-covariance matrics a $KT \times K$ can be provided.
a_prior	a $Mx1$ numeric vector of prior means.

v_i_prior	the inverse of the MxM prior covariance matrix.
svd	logical. If TRUE the singular value decomposition is used to determine the root of the posterior covariance matrix. Default is FALSE which means that the eigenvalue decomposition is used.

Details

The function produces a posterior draw of the coefficient vector a for the model

$$y_t = Z_t a + u_t,$$

where $u_t \sim N(0, \Sigma_t)$. y_t is a K-dimensional vector of endogenous variables and $Z_t = z_t' \otimes I_K$ is a $K \times KM$ matrix of regressors with z_t as a vector of regressors.

For a given prior mean vector $\underline{a}$ and prior covariance matrix $\underline{V}$ the posterior covariance matrix is obtained by

$$\overline{V} = \left[\underline{V}^{-1} + \sum_{t=1}^{T} Z_t' \Sigma_t^{-1} Z_t \right]^{-1}$$

and the posterior mean by

$$\overline{a} = \overline{V} \left[\underline{V}^{-1} \underline{a} + \sum_{t=1}^{T} Z_t' \Sigma_t^{-1} y_t \right].$$

Value

A vector.

Examples

```
# Load data
data("e1")
data <- diff(log(e1))

# Generate model data
temp <- create_bvarmodel(data, p = 2, deterministic = "const")
y <- t(temp$data$train$y)
z <- temp$data$train$z
k <- nrow(y)
tt <- ncol(y)
m <- ncol(z)

# Priors
a_mu_prior <- matrix(0, m)
a_v_i_prior <- diag(0.1, m)

# Initial value of inverse Sigma
sigma_i <- solve(tcrossprod(y) / tt)

# Draw parameters
a <- post_normal_sur(y = y, z = z, sigma_i = sigma_i,
                     a_prior = a_mu_prior, v_i_prior = a_v_i_prior)
```

predict.bvarmodel *Predict Method for Objects of Class bvar*

Description

Forecasting a Bayesian VAR object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
predict(object, n_ahead = NULL, ...)
```

Arguments

object	an object of class 'bvarmodel'.
n_ahead	number of steps ahead at which to predict. If NULL (default), every period simulated by add_posterior_forecasts, i.e. the horizon given to add_forecast_input.
...	additional arguments.

Details

For the VAR model

$$A_0 y_t = \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=0}^{s} B_i x_{t-i} + CD_t + u_t,$$

with $u_t \sim N(0, \Sigma)$ the function produces n_ahead forecasts.

Value

An object of class 'bvarprd', a list with element fcst, an $h \times K \times S$ array of the S forecast draws, whose first two dimensions are named after the forecast periods and the endogenous variables, and element y, the endogenous variables of the training sample.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), spillover.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add the data the forecasts are produced from
model <- add_forecast_input(model, n_ahead = 10)

# Simulate the forecasts
model <- add_posterior_forecasts(model)

# Collect them
pred <- predict(model)
```

---

predict.bvecmodel *Predict Method for Objects of Class bvecmodel*

Description

Summarises the forecasts of a VEC model, in levels.

Usage

```
## S3 method for class 'bvecmodel'
predict(object, n_ahead = NULL, ...)
```

Arguments

object	an object of class 'bvecmodel', usually, the result of a call to add_posterior_forecasts.
n_ahead	number of steps ahead at which to predict. If NULL (default), every period simulated by add_posterior_forecasts.
...	additional arguments passed to predict.bvarmodel.

Details

The forecasts are of the levels, and so is the history they are shown with: it is recovered from the differences and the error correction term the way vec_to_var recovers it.

Value

A time-series object of class "bvarprd", as returned by predict.bvarmodel.

See Also

Other posterior simulation: add_forecast_input.bvarmodel(), add_forecast_input.bvecmodel(), add_posterior_coefficients.bvarmodel(), add_posterior_coefficients.bvecmodel(), add_posterior_forecast
add_posterior_forecasts.bvecmodel(), add_posterior_loglik.bvarmodel(), add_posterior_loglik.bvecmodel
add_seed(), bayests_files(), bayests_posterior(), bvar(), bvec(), chain_diagnostics()

---

predict.expandingwindow

Predict Method for Objects of Class expandingwindow

Description

Forecasting Bayesian VAR objects in a list of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
predict(object, ...)
```

Arguments

object	an object of class 'expandingwindow'.
...	additional arguments.

Value

A time-series object of class 'expandwindbvarprdlist'.

Examples

```
data("us_macrodata")

model <- create_bvarmodel(data = us_macrodata,
                          p = 1,
                          deterministic = "none",
                          error = "gamma",
                          iterations = 10,
                          burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.
```

```
# Obtain objects for expanding window estimation
model <- use_expanding_window(model, start = 2007)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(shape = 3, rate = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

model <- add_forecast_input(model, n_ahead = 10)
model <- add_posterior_forecasts(model)

# Predict
prd <- predict(model, n_ahead = 4)
```

`predict.modellist` *Predict Method for Objects of Class modellist*

Description

Forecasting Bayesian VAR objects in a list of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
predict(object, ...)
```

Arguments

`object`	an object of class 'modellist'.
`...`	additional arguments.

Value

A time-series object of class 'bvarprdlist'.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))
```

```
# Generate model data
model <- create_bvarmodel(e1, p = 1:2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add forecast input data
model <- add_forecast_input(model, n_ahead = 10)
model <- add_posterior_forecasts(model)

# Generate forecasts
bvar_pred <- predict(model, n_ahead = 10)
```

`prepare_forecast_input` *Prepare Forecast Input*

Description

Generic function that generates data matrices serving as data input for downstream forecasting functions.

Usage

```
prepare_forecast_input(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: prepare_forecast_input.bvarmodel, prepare_forecast_input.bvecmodel.

`prepare_forecast_input.bvarmodel` *Prepare Forecast Input*

Description

Generates data matrices serving as input for forecasting simulation for objects of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
prepare_forecast_input(
  object,
  n_ahead = 10,
  deterministic = NULL,
  exogen = NULL,
  ...
)
```

Arguments

`object`	an object of class 'bvarmodel'.
`n_ahead`	number of steps ahead at which to predict.
`deterministic`	a time-series object with deterministic data. If not specified, the function will try to identify the deterministic terms automatically. If this is not successful, an error message we be returned.
`exogen`	a time-series object with the unmodelled, non-deterministic variables of the model. Required if the model has such variables. See 'Details'.
`...`	additional arguments.

Details

The regressors of a forecast period contain the values of the unmodelled, non-deterministic variables in that period and in the `s` periods before it. Argument `exogen` therefore has to cover the last `s` periods of the estimation sample as well as the `n_ahead` forecast periods, at the frequency of the model. A series that starts with the first forecast period, or ends before the last one, is refused with a message that names the periods it has to cover.

If `deterministic` is not given, the deterministic terms are continued from the estimation sample, which works for a constant, a linear trend, seasonal dummies and the dummy variables of add_dummy_variables.

Value

A list with elements h, the forecast horizon, and x, the out-of-sample regressors: h rows, one per period, by one column per regressor. That is the compact layout, the same one a coefficient matrix is k by; the SUR layout this used to return spread every regressor over k columns and was k^2 the size for no extra content. The lags of the endogenous variables that the estimation sample does not reach are the forecasts of earlier periods, which the forecast fills in as it goes; they are zero here, since the samplers refuse a missing value anywhere in x.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Generate forcast input
fcst_input <- prepare_forecast_input(model, n_ahead = 4)
```

---

prepare_forecast_input.bvecmodel

Prepare Forecast Input

Description

Prepares the regressors of the forecast periods of a VEC model, in levels.

Usage

```
## S3 method for class 'bvecmodel'
prepare_forecast_input(object, ...)
```

Arguments

object	an object of class 'bvecmodel'.
...	arguments passed to prepare_forecast_input.bvarmodel, such as n_ahead, deterministic and exogen.

Details

A VEC model is forecast in levels, so its forecast regressors are those of its VAR representation. They are prepared from the data and specification `vec_to_var` builds; the posterior draws are not converted.

Value

A list with the forecast horizon in element `h` and the regressors of the forecast periods in element `x`, as returned by `prepare_forecast_input.bvarmodel`.

`print.bvarmodel` *Printing Model Information*

Description

print method for objects of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`x`	an object of class 'bvarmodel'.
`digits`	the number of significant digits to use when printing.
`...`	further arguments passed to or from other methods.

Value

A data frame of the model's specification, as printed, invisibly. It is the result of `get_model_specifications` with the columns renamed for display and, where the model has no unmodelled variables, the columns `m` and `s` dropped.

print.bvarspillover *Print a Spillover Index*

Description

Prints the connectedness table of Diebold and Yilmaz (2012).

Usage

```
## S3 method for class 'bvarspillover'
print(x, digits = 1, ...)
```

Arguments

x	an object of class 'bvarspillover', usually the result of a call to spillover.
digits	the number of decimal places. Defaults to 1, which is what this literature reports.
...	not used.

Details

The body of the table is the posterior mean share of the forecast error variance of the variable in the row that is attributed to a shock to the variable in the column, in percent. The from column and the to row are the directional spillovers, and the corner is the total index.

Only the means appear here. The credible intervals are in the total, from, to and net elements of the object.

Value

x, invisibly.

print.bvecmodel *Printing Model Information*

Description

print method for objects of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	an object of class 'bvecmodel'.
digits	the number of significant digits to use when printing.
...	further arguments passed to or from other methods.

Value

A data frame of the model's specification, as printed, invisibly. It is the result of get_model_specifications with the columns renamed for display and, where the model has no unmodelled variables, the columns m and s dropped.

---

print.modellist *Printing Model Information*

Description

print method for objects of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	an object of class 'modellist'.
digits	the number of significant digits to use when printing.
...	further arguments passed to or from other methods.

Value

A data frame with one row per model, as printed, invisibly: the specifications of get_model_specifications, without the columns in which every model agrees.

read_expanding_window_model_from_folder *Import Models from HDF5 Files*

Description

Imports model information and posterior draws from an HDF5 file.

Usage

```
read_expanding_window_model_from_folder(folder)
```

Arguments

folder	Path to a folder with HDF5 files containing model data.

Value

A list of class 'expandingwindow' with one model per file in `folder`, each read by `read_model_from_hdf5`, in the order `list.files` returns the files.

read_model_from_hdf5 *Import Models from HDF5 Files*

Description

Imports model information and posterior draws from an HDF5 file.

Usage

```
read_model_from_hdf5(filename, group = "", draws = NULL)
```

Arguments

filename	Path to an HDF5 file containing model data.
group	the group the model's tree hangs under inside its file. Defaults to `""`, the root of the file, which is where a file holding a single model puts it. See 'Details'.
draws	the draws to read, as their positions in the chain. Defaults to `NULL`, every draw. An integer vector reads those draws of every block of the posterior, and `integer(0)` reads none of them, which gives the model, its data and its priors without the draws. See 'Details'.

Details

With a `group` every path the reader looks for is read under it instead of at the root, so one file can hold several models side by side. `list_models_in_hdf5` reports which groups of a file hold one.

The spelling is the one the BayesTS command line uses for its `--group` flag: a leading slash and no trailing slash, with `""` for the root.

`draws` reads part of a chain. Every block of a posterior holds one row per draw, and only the rows asked for are read from the file, so a caller that works through a long chain in pieces – solving a global model draw by draw, for instance – never holds more of it than the piece it is working on. `integer(0)` reads a model without its draws, which is what a step needs that only looks at what a model is.

A partial read cannot describe the chain it came from, so the blocks it returns are labelled as a chain of their own, from one to the number of draws read. A full read keeps the labels the file carries.

Value

An object of class 'bvarmodel' or 'bvecmodel', depending on the algorithm recorded in the file, with the elements that file holds: `model`, `data`, `priors`, `initial` and, where the model has been estimated, `posterior`. The draws of a sampler are `mcmc` objects; the per-period posterior of a discounted model is a plain matrix, since its rows are periods rather than draws. `bvartools_model` describes the elements one by one.

A model of another package – a dynamic factor model of **dfmtools**, say – is returned as the object of that package's `from_bayests_tree` method, so that one 'modellist' can hold and evaluate models of either.

`read_models_from_folder` *Import Models from a Folder of HDF5 Files*

Description

Imports every model stored below a folder.

Usage

```
read_models_from_folder(folder, draws = NULL)
```

Arguments

`folder`	Path to a folder containing model data.
`draws`	the draws to read of every model, as `read_model_from_hdf5` takes them: `NULL`, the default, for the whole chain, a vector of positions for those draws, or `integer(0)` for none of them.

Details

The folder is walked recursively, and each HDF5 file in it is asked which of its groups hold a model – so a file holding several models contributes all of them. See `list_models_in_hdf5` for what counts as one.

The result is a flat named list rather than a nesting that mirrors the directory tree. Each name is the path of the file relative to `folder` without its extension, followed by the group where a model does not sit at the root of its file. Names are what a caller needs to tell one model from another – which sub-model of a global model it is, say – and a nesting whose depth depended on where the caller pointed could not provide them.

The one exception is an expanding window: the windows that `write_to_hdf5` wrote to one directory come back as one element of class 'expandingwindow', named after that directory, so that the windows of different specifications are not pooled. A folder that holds a single expanding window is returned as that expanding window. A model list written by `write_to_hdf5` comes back in the order it was written in, which the files record; other models are in the order of their names.

The class of each model comes from the `rclass` attribute the writer records, not from its file name.

`draws` is what makes a folder larger than the session readable: with `integer(0)` the models come back with their specification, their data and their priors and no draws, and with a vector of positions the chain is read a piece at a time. For working on the models rather than reading them, `open_models` hands them over one at a time instead.

Value

A named list of class 'modellist', or of class 'expandingwindow' if the models say they belong to one.

Examples

```
# Load data
data("e1")
train <- diff(log(e1)) * 100

# Create and store models
folder <- file.path(tempdir(), "models")
dir.create(folder, showWarnings = FALSE)
for (p in 1:2) {
  model <- create_bvarmodel(data = train, p = p, deterministic = "const",
                            iterations = 10, burnin = 10)
  write_to_hdf5(model, filename = file.path(folder, paste0("model-", p, ".h5")))
}

models <- read_models_from_folder(folder)
names(models)
```

`rescale_error_correction` *Rescale Error Correction*

Description

Rescales the series in the error correction series.

Usage

```
rescale_error_correction(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `rescale_error_correction.bvecmodel`.

`rescale_error_correction.bvecmodel` *Rescale Error Correction*

Description

Puts the series in the error correction term of an object of class 'bvecmodel', and the posterior draws that belong to them, back in terms of the input data.

Usage

```
## S3 method for class 'bvecmodel'
rescale_error_correction(object, ...)
```

Arguments

`object`	object of class 'bvecmodel'.
`...`	arguments passed forward to method.

Details

The function transforms element `object$data$train$w`, the posterior draws and the starting values back to the series of the input data, based on the attributes `"scale"` and `"centre"` that `scale_error_correction` stored in `object$data$train$w`, and drops the attributes.

If D is the diagonal matrix of scaling factors, function `scale_error_correction` replaced w_t by $D^{-1}w_t$, so that the estimated error correction term is $\alpha\beta' D^{-1}w_t$. The draws of β are therefore multiplied by D^{-1}, while the draws of α are not affected by the transformation and are carried over unchanged.

If the series were centred on their means m, the estimated term is $\alpha\beta' D^{-1}(w_t - m)$, with D the identity matrix if they were not scaled. The draws of the unrestricted constant are therefore shifted by $-\alpha\beta' D^{-1}m$, draw by draw and, for a model with time varying parameters, period by period with the loadings and the cointegration vectors of that period. The fitted values and the log-likelihood of every draw are unchanged by that. The starting values of the constant are shifted in the same way. If variable selection covered the deterministic terms, the draws of the inclusion indicators of the constant are carried over as they are, so they describe the constant of the centred series.

What is unchanged is the sample. Beyond it, a model with time varying cointegration vectors cannot be carried on from the rescaled draws: the state equation of β_t was estimated for the transformed series, and the constant absorbs $\alpha_t\beta_t' D^{-1}m$ of the last period only, so a step of β_t would reintroduce the intercept the centring removed. The function therefore sets `object$model$ect_rescaled` for such a model, and `add_posterior_forecasts.bvecmodel` and `add_predictive_loglik.bvecmodel` accept it only with `forecast_states = "hold"`.

Value

An object of class 'bvecmodel'.

---

scale_error_correction

Scale Error Correction

Description

Scales the series in the error correction series.

Usage

```
scale_error_correction(object, ...)
```

Arguments

`object`	an object of a class, for which a method should be called.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: scale_error_correction.bvecmodel.

---

scale_error_correction.bvecmodel
Scale Error Correction

Description

Scales, centres, or centres and scales the series in the error correction term of an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
scale_error_correction(object, scale = TRUE, centre = FALSE, ...)
```

Arguments

object	object of class 'bvecmodel'.
scale	logical. If TRUE, the default, the series are divided by scaling factors. See 'Details'.
centre	logical. If TRUE, the sample mean of each stochastic series is subtracted from it. Default is FALSE. See 'Details'.
...	arguments passed forward to method.

Details

The function transforms element object$data$train$w, and rescale_error_correction undoes the transformation once the posterior draws are in.

With scale = TRUE, stochastic series are divided by the standard deviation of the corresponding differenced series. If the time-series object contains a column named "trend", this series is divided by its own standard deviation, i.e. in levels. The scaling factors are stored as a new attribute of object$data$train$w named "scale".

With centre = TRUE, the sample mean of each stochastic series is subtracted from it. The deterministic terms restricted to the cointegration space, which are the last columns of the term, are left as they are. The means are stored as attribute "centre", with zeros for the deterministic terms. If both arguments are TRUE, the series are centred first and scaled second, so that w_t becomes $D^{-1}(w_t - m)$.

Centring requires an unrestricted constant, which takes up what the series lose: $\Pi w_t + c = \Pi(w_t - m) + (c + \Pi m)$. For a model with constant coefficients it is therefore a reparameterisation that leaves the likelihood and the priors on α and β as they are and changes only what the prior of the constant refers to. For a model with time varying cointegration vectors it changes more. A step η_t of their state equation moves $\beta_t' w_t$ by $\eta_t' w_t$, which for series far from zero acts as a random walk intercept that no prior on the deterministic terms controls; see section 'Prior on the cointegration space' of cointspace_prior. On centred series the same step moves the term by $\eta_t'(w_t - m)$, and the

intercept is left to the constant and its own prior. That holds for the estimation. The forecast of such a model, which needs `rescale_error_correction` first, can only hold the cointegration vectors at their last value (`forecast_states = "hold"` in `add_posterior_forecasts.bvecmodel`), since after rescaling a step of them acts on the series as they are.

Starting values of the constant in `object$initial` are shifted by Πm, computed from the starting values of the loadings and of β, so that the chain starts from the model it simulates.

Neither transformation can be applied twice, and neither can be added to a model that already carries the other: `rescale_error_correction` has to be called in between.

Value

An object of class 'bvecmodel'.

`selection_criteria` *Model Selection Criteria*

Description

Generic function used to calculate selection criteria.

Usage

```
selection_criteria(object, ...)

## S3 method for class 'selcrit'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'selcritlist'
print(x, digits = max(3L, getOption("digits") - 3L), relative = 0, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.
`x`	an object used to select a method. Usually, the result of a call to `selection_criteria`.
`digits`	the minimum number of significant digits to be printed in values.
`relative`	an integer specifying the model that is used as the reference of relative forecast performance. Default is `0`, which indicates that results are not displayed in relation to each other.

Details

The in-sample criteria are obtained from the draws of the log-likelihood that add_posterior_loglik adds to a model. With $ll_t^{(i)}$ the log-likelihood of period t in draw i of R draws, $LL^{(i)} = \sum_{t=1}^{T} ll_t^{(i)}$ and κ the number of estimated parameters, these are

- "LL": the log-likelihood $LL^{(i)}$, which is summarised by the mean, the median and the bounds of the credible band of its draws.
- "AIC": $D + 2\kappa$;
- "BIC": $D + \ln(T)\kappa$;
- "HQ": $D + 2\ln(\ln(T))\kappa$;
- "WAIC": $-2\sum_{t=1}^{T}\left(\ln\left(\frac{1}{R}\sum_{i=1}^{R}\exp(ll_t^{(i)})\right) - \mathrm{Var}_i\left[ll_t^{(i)}\right]\right)$;
- "LOOIC": -2 times the expected log pointwise predictive density of leave-one-out cross validation, obtained by Pareto smoothed importance sampling.

D is the deviance of the model at its point estimate. Since AIC, BIC and HQ correct the deviance of a fitted model for the optimism of having fitted it, it is the deviance at the point estimate that their penalties belong to. The mean of the deviance over the posterior is the larger quantity, by about the effective number of parameters, so adding a penalty to it would charge the complexity of the model a second time and tilt every comparison towards the smaller model. D is therefore the deviance at the posterior mean of the parameters, obtained by evaluating the log-likelihood of the model once at that point. For a VEC model the point is the matrix of the rank of the model that is closest to the posterior mean of $\Pi = \alpha\beta'$ in the metric of the likelihood, i.e. that minimises $tr(\bar{Q}\Delta W'W\Delta')$, where Δ is its difference to the posterior mean, $\bar{Q}$ the posterior mean of the error precision and W the error correction term, since α and β are identified only up to a rotation and their own means say nothing. The point does not depend on the scale of the series. Under a flat prior D is close to the deviance at the maximum likelihood estimates. Evaluated at the maximum likelihood estimates AIC reduces to T times the expression in Luetkepohl (2006) plus terms that do not depend on the lag order, so both order a set of lag orders in the same way.

AIC, BIC and HQ are point estimates rather than quantities with a posterior distribution, so their credible bands are NA. WAIC and LOOIC are point estimates as well, and their bands are normal intervals built from their standard errors, which is the usual way they are reported.

The choice between the criteria is one of what κ means for the model at hand. A count of parameters describes a model with constant coefficients and a weak prior, which is the case AIC, BIC and HQ are derived for and the one in which they reproduce the textbook lag order selection. It does not describe a model whose coefficients or variances follow a state equation, where the prior on the state variances decides how much of the nominal freedom is used, and it does not describe a shrinkage prior, which buys back degrees of freedom that κ does not see. WAIC and LOOIC penalise by the flexibility the fit actually used and remain defined in those cases.

LOOIC estimates the same quantity as WAIC by reweighting the posterior towards the one that has not seen a period, which is the more accurate route when it works and which says when it does not: the shape parameter of the Pareto tail fitted to the importance ratios of a period exceeds its threshold when that period is too influential to be reweighted, and the print methods report how many periods this happened for.

For a model whose coefficients or variances follow a state equation, the pointwise log-likelihood that WAIC and LOOIC are built from is evaluated at the states of each period, which have seen

that period's observation. The state of a period is informed mostly by its own observation, so leaving the observation out by reweighting fails for exactly the periods a path bends towards, and neither criterion separates models that differ in how much of that freedom they have, such as the cointegration ranks of a time varying VEC model. The criterion for those comparisons is `"LPL"`, the log predictive likelihood, whose densities condition only on the data before the period they evaluate. Koop, León-González and Strachan (2011) choose between time varying cointegration models with it.

Two things produce those densities and both report them as `"LPL"`. `add_predictive_loglik` takes one per window of an expanding window exercise, carrying the states forward with their state equations, which is the form the comparison of ranks above is made in. A single model carries one per horizon of its forecast in `posterior$forecast$loglik`, written by BayesTS against the values in `data$test$y`, each conditioning on the periods realised before it. They are the same quantity computed two ways and are summarised by the same code, so the same densities give the same number.

All criteria require that the models that are compared were estimated on the same observations. `create_bvarmodel` and `create_bvecmodel` ensure this for a vector of lag orders by trimming the data by the maximum lag, and `align_model_obs` restricts models that were created separately to their common sample.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

References

Koop, G., León-González, R., & Strachan, R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

Luetkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing, 27*(5), 1413–1432. doi:10.1007/s1122201696964

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., & Gabry, J. (2024). Pareto smoothed importance sampling. *Journal of Machine Learning Research, 25*(72), 1–58.

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research, 11*, 3571–3594.

Zhang, J., & Stephens, M. A. (2009). A new and efficient estimation method for the generalized Pareto distribution. *Technometrics, 51*(3), 316–325. doi:10.1198/tech.2009.08017

See Also

Methods: `selection_criteria.bvarmodel`, `selection_criteria.bvecmodel`, `selection_criteria.expandingwindo`
`selection_criteria.externalforecast`, `selection_criteria.modellist`.

`selection_criteria.bvarmodel` *Model Selection Criteria*

Description

Calculates model selection criteria for an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
selection_criteria(object, ci = 0.95, plugin = c("mean", "best"), ...)
```

Arguments

`object`	an object of class 'bvarmodel'.
`ci`	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
`plugin`	the point at which the deviance of `AIC`, `BIC` and `HQ` is evaluated. `"mean"`, the default, uses the posterior mean of the parameters, for a VEC model the best approximation of the model's rank to the posterior mean of Π. `"best"` uses the draw whose log-likelihood is the highest the chain visited, which is closer to the maximum these corrections are derived for, but grows with the length of the chain and is therefore comparable only across models drawn equally long.
`...`	further arguments passed to or from other methods.

Value

A list of class 'selcrit', which also inherits the class of the model, with the element `model` and one data frame per criterion. If the model contains `posterior$loglik`, these are `LL`, `AIC`, `BIC`, `HQ`, `WAIC` and `LOOIC`, each with the columns `mean`, `median`, `qlower` and `qupper`, where bands that do not apply are `NA`. If it contains `posterior$forecast$errors`, these are `FE`, `AFE` and `RSFE`, the forecast errors and their absolute and root squared values, with the columns `variable`, `h`, `mean`, `median`, `qlower` and `qupper`. For `RSFE` only `mean` is a statistic of its own, the root of the mean squared error. Its median and quantiles are those of the squared errors, rooted, and so differ from `AFE`'s only by the interpolation between neighbouring draws that `quantile` and `median` do – which, the root of an average of squares being at least the average of the roots, can only raise them. They add nothing to `AFE`'s. If it contains `posterior$forecast$loglik`, the score of its forecast against `data$test$y`, there is `LPL`, the log predictive likelihood: the log of the mean of the draws of each scored period's predictive density, summed over the periods. Attribute `"terms"` holds those per period and `"nse"` their numerical standard error.

See Also

Other model comparison: `add_forecast_errors.bvarmodel()`, `add_forecast_errors.bvecmodel()`, `add_predictive_loglik()`, `add_predictive_loglik.bvarmodel()`, `add_predictive_loglik.bvecmodel()`,

aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri
create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(),
open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvecmodel(),
selection_criteria.default(), selection_criteria.modellist()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 1:2, deterministic = "const",
                          iterations = 10, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add log-likelihoods
model <- add_posterior_loglik(model)

# Calculate selection criteria
sel <- selection_criteria(model)
sel
```

---

selection_criteria.bvecmodel
Model Selection Criteria

Description

Calculates model selection criteria for an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
selection_criteria(object, ci = 0.95, plugin = c("mean", "best"), ...)
```

Arguments

object	an object of class 'bvecmodel'.
ci	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
plugin	the point at which the deviance of AIC, BIC and HQ is evaluated. "mean", the default, uses the posterior mean of the parameters, for a VEC model the best approximation of the model's rank to the posterior mean of Π. "best" uses the draw whose log-likelihood is the highest the chain visited, which is closer to the maximum these corrections are derived for, but grows with the length of the chain and is therefore comparable only across models drawn equally long.
...	further arguments passed to or from other methods.

Value

A list of class 'selcrit', which also inherits the class of the model, with the element model and one data frame per criterion, as described in selection_criteria.bvarmodel.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.default(), selection_criteria.modellist()

Examples

```
# Load data
data("e6")

# Create model
model <- create_bvecmodel(e6, p = 4,
                          const = "unrestricted",
                          seasonal = "unrestricted",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)
```

```
# Add log-likelihoods
model <- add_posterior_loglik(model)

# Calculate selection criteria
sel <- selection_criteria(model)
sel
```

`selection_criteria.default` *Selection Criteria*

Description

Calculates the criteria that a pointwise log-likelihood and a scored forecast are enough for, whatever the model is.

Usage

```
## Default S3 method:
selection_criteria(object, ci = 0.95, ...)
```

Arguments

`object`	any object with `posterior$loglik`, `posterior$forecast$loglik` or both. A model of a class this package has no method for – a dynamic factor model of **dfmtools**, say – reaches this one.
`ci`	the width of the credible bands, a value between 0 and 1. Defaults to 0.95.
`...`	additional arguments.

Details

The criteria of `selection_criteria.bvarmodel` fall into two groups. `AIC`, `BIC` and `HQ` charge a model for its size, so they need a count of its free parameters, which is a property of the model and cannot be read off its draws. `FE`, `AFE` and `RSFE` need the forecast errors, which need the variables to be named and paired up. Neither group is available here.

What is left needs nothing but the draws themselves:

- `LL`, the log-likelihood of the whole sample per draw, summed over the periods of `posterior$loglik`;
- `WAIC` and `LOOIC`, which penalise by the flexibility the fit used rather than by a count of parameters – see `selection_criteria` for why that is what makes constant, time varying and stochastic volatility specifications comparable at all;
- `LPL`, the log predictive likelihood of `posterior$forecast$loglik`, the score of a forecast against what its horizon realised.

The periods of the "terms" attribute of LPL are numbered from one rather than dated: this method knows nothing of where a model keeps its sample, so it cannot say when the scored periods were. A class that can say should write its own method, as selection_criteria.bvarmodel does.

Value

A list of class 'selcrit', which also inherits the class of object, with the element model and one data frame per criterion that the draws supported, each with the columns mean, median, qlower and qupper. choose_best_model ranks a list of them and print shows them.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.modellist()

Examples

```
# Any object with the draws will do, whatever produced them.
set.seed(7)
model <- list(model = list(k = 2),
              posterior = list(loglik = matrix(stats::rnorm(200, -3), 50, 4),
                               forecast = list(loglik = matrix(stats::rnorm(150, -3), 50, 3))))

criteria <- selection_criteria(model)
names(criteria)
criteria[["LPL"]][["mean"]]
```

---

selection_criteria.expandingwindow
Model Selection Criteria

Description

Calculates model selection criteria for an object of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
selection_criteria(object, ci = 0.95, ...)
```

Arguments

object	an object of class 'expandingwindow'.
ci	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
...	further arguments passed to or from other methods.

Details

The in-sample criteria are those of the last window, the one estimated on the most data, and the forecast error statistics those of all windows.

If the windows hold the draws of add_predictive_loglik, criterion "LPL" is the log predictive likelihood: the sum over the windows of the log of the mean of the draws of the predictive density of the observation that the next window adds. Its band is the normal interval of probability ci around the sum with standard error $\sqrt{n\mathrm{Var}(lpd_t)}$ over the n periods, as for LOOIC. Attribute "terms" holds the log predictive density of each period and its numerical standard error, attribute "nse" the numerical standard error of the sum.

Value

An object of class 'selcrit'.

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

model <- use_expanding_window(model, start = 1982.25)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)
```

```
# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = orig)

# Calculate selection criteria
sel <- selection_criteria(model)
sel
```

`selection_criteria.externalforecast`
Model Selection Criteria

Description

Calculates out-of-sample statistics for an object of class 'externalforecast'.

Usage

```
## S3 method for class 'externalforecast'
selection_criteria(object, ci = 0.95, ...)
```

Arguments

`object`	an object of class 'externalforecast'.
`ci`	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
`...`	further arguments passed to or from other methods.

Details

External forecasts are not estimated, so only out-of-sample statistics are calculated. Since they are point forecasts, each publication contributes a single value to the statistics of a variable and forecast horizon.

Value

An object of class 'selcrit'.

Examples

```
data("us_macrodata")

# Create model
model <- create_bvarmodel(data = us_macrodata, p = 1, deterministic = "none",
                          error = "gamma", iterations = 10, burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.

model <- use_expanding_window(model, start = 2007)

# Artificial external forecasts
fcst <- expand.grid(origin = c(2007, 2007.25),
                    h = 1:2,
                    variable = c("Dp", "r"),
                    stringsAsFactors = FALSE)
fcst[["period"]] <- fcst[["origin"]] + fcst[["h"]] / 4
fcst[["value"]] <- 0

ext <- create_external_forecast(fcst, model, n_ahead = 4)

# Calculate forecast errors
ext <- add_forecast_errors(ext, test_sample = us_macrodata)

# Calculate selection criteria
selection_criteria(ext)
```

---

selection_criteria.modellist

Model Selection Criteria

Description

Calculates model selection criteria for a list of Bayesian models.

Usage

```
## S3 method for class 'modellist'
selection_criteria(object, ...)
```

Arguments

object	an object of class 'modellist'.
...	further arguments passed to or from other methods.

Value

A list of class 'selcritlist' with one object of class 'selcrit' per model, as described in selection_criteria.bvarmodel, which choose_best_model compares.

See Also

Other model comparison: add_forecast_errors.bvarmodel(), add_forecast_errors.bvecmodel(), add_predictive_loglik(), add_predictive_loglik.bvarmodel(), add_predictive_loglik.bvecmodel(), aggregate_forecasts(), align_model_obs.modellist(), analysis_of_stored_models, choose_best_model.selcri create_external_forecast(), folder_steps, map_draws(), map_models(), open_model(), open_models(), plot_forecast_errors_by_period(), pool_forecasts(), selection_criteria.bvarmodel(), selection_criteria.bvecmodel(), selection_criteria.default()

Examples

```
# Load data
data("e1")
orig <- diff(log(e1)) * 100
train <- window(orig, end = c(1982, 2))


# Create model
model <- create_bvarmodel(data = train, p = 0:2, deterministic = "const",
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Add data used for forecast calculation
model <- add_forecast_input(model, n_ahead = 4)

# Add forecasts
model <- add_posterior_forecasts(model)

# Add forecast errors
model <- add_forecast_errors(model, test_sample = orig)

# Calculate selection criteria
sel <- selection_criteria(model)
sel
```

---

spillover *Spillover Index*

Description

A generic function used to calculate the connectedness measures of Diebold and Yilmaz (2012) from the posterior draws of a model.

Usage

```
spillover(object, ...)
```

Arguments

`object`	an object with suitable input data passed forward to method.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `spillover.bvarmodel`, `spillover.expandingwindow`, `spillover.modellist`.

---

spillover.bvarmodel *Spillover Index*

Description

Produces the connectedness measures of Diebold and Yilmaz (2012) for an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
spillover(
  object,
  n_ahead = 10,
  type = "gir",
  ci = 0.95,
  keep_draws = FALSE,
  period = NULL,
  impact = NULL,
  ...
)
```

Arguments

`object`	an object of class 'bvarmodel'.
`n_ahead`	the forecast horizon H of the decomposed forecast error variance, which is made of the impulse responses of periods 0 to $H-1$. Defaults to 10, the horizon of Diebold and Yilmaz (2012). The table therefore corresponds to the rows of period $H-1$ of `fevd`, which counts its periods from the impact period.
`type`	type of the impulse responses the decomposition is based on. Possible choices are generalised `gir` (default), orthogonalised `oir`, sign restricted `sign` and `custom`. All four decompose the reduced form of the model, so a structural model is not supported. See 'Details'.
`ci`	a numeric between 0 and 1 specifying the probability mass covered by the credible intervals. Defaults to 0.95.
`keep_draws`	logical specifying whether the function should return all draws of the indices. Defaults to `FALSE`, so that the credible intervals are returned.
`period`	integer. Index of the period, for which the measures should be generated. Only used for TVP or SV models. Default is `NULL`, so that the posterior draws of the last time period are used. With `type = "sign"` the default is the period the restrictions were imposed in, and another period is refused, since the rotations do not identify it.
`impact`	the impact matrix of a `custom` decomposition, either a single $K \times K$ matrix that identifies every posterior draw the same way, or a list of such matrices with one entry per draw. Ignored for every other value of `type`. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

The function produces the connectedness measures of Diebold and Yilmaz (2012) for the VAR model

$$y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t,$$

with $u_t \sim N(0, \Sigma)$.

Let $\theta_{jk}(H)$ be the share of the H step forecast error variance of variable j that is attributed to a shock to variable k. Under `type = "gir"` this is the generalised decomposition of Pesaran and Shin (1998),

$$\theta_{jk}(H) = \frac{\sigma_{kk}^{-1} \sum_{i=0}^{H-1} (e_j' \Phi_i \Sigma e_k)^2}{\sum_{i=0}^{H-1} (e_j' \Phi_i \Sigma \Phi_i' e_j)},$$

where Φ_i is the forecast error impulse response of period i and σ_{kk} the variance of the *shock* variable. These shares do not add up over k, so each row is normalised,

$$\tilde{\theta}_{jk}(H) = \theta_{jk}(H) / \sum_{k} \theta_{jk}(H).$$

Under `type = "oir"` the decomposition uses the Choleski factor of Σ, adds up by construction and depends on the ordering of the variables, which is what the generalised version of Diebold and Yilmaz (2012) avoids. Under `type = "sign"` it uses the Choleski factor rotated by what

add_sign_restrictions accepted for that draw, leaving out the draws no rotation was found for. Under type = "custom" it uses the matrix supplied in argument impact in place of that factor and is otherwise the orthogonalised case, the scaling by σ_{kk}^{-1} included: an impact matrix carries the scale of its own shocks in its columns. The row normalisation is applied to all three, so it will produce shares that sum to one even from an impact matrix that does not factorise Σ and is therefore not a check that one does.

From the normalised table the measures are

total the total spillover index, $100 \sum_{j \neq k} \tilde{\theta}_{jk}/k$, the share of forecast error variance across the system that comes from other variables.

from spillovers received by each variable from all others, $100(1 - \tilde{\theta}_{jj})/k$.

to spillovers transmitted by each variable to all others, $100 \sum_{j \neq k} \tilde{\theta}_{jk}/k$ summed down column k.

net to less from. Positive values mark net transmitters of shocks and negative values net receivers.

pairwise net pairwise spillovers, $100(\tilde{\theta}_{kj} - \tilde{\theta}_{jk})/k$.

Every measure is computed once per posterior draw and only then summarised. That is not interchangeable with computing it from the posterior mean table: the measures are ratios, so the index of the mean is not the mean of the index. It is also what gives the index a credible interval, which the original point estimate based version does not have.

Objects of class 'bvecmodel' have to be transformed with vec_to_var first, as for fevd.

Value

An object of class 'bvarspillover', a list containing

total	the total spillover index. The median and the bounds of the credible interval, or all draws if keep_draws = TRUE.
to, from, net	the directional measures, one column per variable, in the same form.
table	the posterior mean of the normalised decomposition table, in percent, with the from column, the to row and the total index in the corner attached, so it is $(k+1) \times (k+1)$.
pairwise	the posterior mean of the net pairwise spillovers.
specification	a list recording n_ahead, type, ci, period, k and the number of draws.

References

Diebold, F. X., & Yilmaz, K. (2012). Better to give than to receive: Predictive directional measurement of volatility spillovers. *International Journal of Forecasting, 28*(1), 57–66. doi:10.1016/j.ijforecast.2011.02.006

Diebold, F. X., & Yilmaz, K. (2014). On the network topology of variance decompositions: Measuring the connectedness of financial firms. *Journal of Econometrics, 182*(1), 119–134. doi:10.1016/j.jeconom.2014.04.012

Pesaran, H. H., & Shin, Y. (1998). Generalized impulse response analysis in linear multivariate models. *Economics Letters, 58*, 17–29. doi:10.1016/S01651765(97)002140

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvecmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
                          iterations = 100, burnin = 10)
# Chosen number of iterations and burnin should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
object <- add_posterior_coefficients(model)

# Obtain the connectedness measures
sp <- spillover(object)
sp

# Net directional spillovers
plot(sp)
```

---

spillover.bvecmodel *Spillover Index*

Description

Guard method for objects of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
spillover(object, ...)
```

Arguments

object	an object of class 'bvecmodel'.
...	additional arguments.

Value

Nothing. The method raises an error, since spillover indices of a VEC model are obtained from its VAR representation: apply vec_to_var first and then spillover to the resulting 'bvarmodel'.

See Also

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), split_quantile_grid(), vec_to_var.bvecmodel()

---

spillover.expandingwindow

Rolling Spillover Index

Description

Produces the total spillover index of Diebold and Yilmaz (2012) over the windows of an object of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
spillover(object, ...)
```

Arguments

object	an object of class 'expandingwindow' whose models carry posterior draws.
...	arguments passed forward to spillover.bvarmodel.

Details

The index over a moving sample is the chart this literature is built around, and use_expanding_window already produces the models it needs. Each window contributes one value, dated at its own last observation, so the result reads as the connectedness of the system as it was known up to that date.

Note that the windows expand rather than roll: every one of them starts at the beginning of the sample, so later values are estimated on more data. Diebold and Yilmaz use a fixed width instead, which trades that away for a constant amount of information per point.

Value

A time-series object of class 'bvarspilloverts' with the median of the total index and the bounds of its credible interval, one row per window. Windows whose estimation failed are NA.

References

Diebold, F. X., & Yilmaz, K. (2012). Better to give than to receive: Predictive directional measurement of volatility spillovers. *International Journal of Forecasting, 28*(1), 57–66. doi:10.1016/j.ijforecast.2011.02.006

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100
e1 <- window(e1, end = c(1978, 4))

# Generate model data
model <- create_bvarmodel(e1, p = 1, deterministic = "const",
                          iterations = 50, burnin = 10)
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Estimate over expanding windows
windows <- use_expanding_window(model, start = c(1976, 1))
windows <- lapply(windows, function(w) {
  add_posterior_coefficients(add_initial_values(w))
})
class(windows) <- append("expandingwindow", class(windows))

# The index as the sample grows
si <- spillover(windows, n_ahead = 5)
plot(si)
```

---

spillover.modellist *Spillover Index*

Description

Produces the connectedness measures of Diebold and Yilmaz (2012) for every model in an object of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
spillover(object, ...)
```

Arguments

object	an object of class 'modellist'.
...	arguments passed forward to spillover.bvarmodel.

Value

A list of objects of class 'bvarspillover'. Elements belonging to models whose estimation failed are NULL.

---

split_quantile_grid *Single Quantiles of a Quantile Grid*

Description

Splits a structural quantile VAR, a model created with quantile_grid = TRUE in create_bvarmodel, into one model per quantile.

Usage

```
split_quantile_grid(object)
```

Arguments

object	an object of class 'bvarmodel' holding a grid of quantiles, usually the result of a call to add_posterior_coefficients.

Details

The draws of a grid are stacked level by level, and each level's draws are the chain the single-quantile model would have drawn. The models returned hold those draws with the specification of that model, so that everything written for a single quantile – summary, plot, add_posterior_loglik – reads them as such. What describes the grid as a whole, its forecasts and its log likelihood, is not carried over. The impulse responses of a structural grid are differences of forecasts at a fixed level with and without a scenario; see forecast_quantile in add_posterior_forecasts.

Value

A list of class 'modellist' with one object of class 'bvarmodel' per quantile, in increasing order.

See Also

create_bvarmodel

Other post-estimation analysis: add_sign_restrictions.bvarmodel(), add_sign_zero_restrictions.bvarmodel(), fevd.bvarmodel(), fevd.bvecmodel(), historical_decomposition(), historical_decomposition.bvarmodel(), irf.bvarmodel(), irf.bvecmodel(), multipliers(), multipliers.bvarmodel(), multipliers.bvecmodel(), predict.bvarmodel(), spillover.bvarmodel(), spillover.bvecmodel(), vec_to_var.bvecmodel()

Examples

```
data("us_macrodata")

object <- create_bvarmodel(data = us_macrodata, p = 1, deterministic = "const",
                           structural = TRUE, error = "ald",
                           quantile = c(0.1, 0.5, 0.9), quantile_grid = TRUE,
                           iterations = 20, burnin = 10)
object <- add_priors(object, coef = list(v_i = 1),
                     sigma = list(shape = 3, rate = .01))
object <- add_initial_values(object)
object <- add_posterior_coefficients(object)

levels <- split_quantile_grid(object)
summary(levels[[3]])
```

---

ssvs *Stochastic Search Variable Selection*

Description

ssvs employs stochastic search variable selection as proposed by George et al. (2008) to produce a draw of the precision matrix of the coefficients in a VAR model.

Usage

```
ssvs(a, tau0, tau1, prob_prior, include = NULL)
```

Arguments

a	an M-dimensional vector of coefficient draws.
tau0	an M-dimensional vector of prior standard deviations for restricted coefficients in vector a.
tau1	an M-dimensional vector of prior standard deviations for unrestricted coefficients in vector a.
prob_prior	an M-dimensional vector of prior inclusion probabilites for the coefficients in vector a.
include	an integer vector specifying the positions of coefficients in vector a, which should be included in the SSVS algorithm. If NULL (default), SSVS will be applied to all coefficients.

Details

The function employs stochastic search variable selection (SSVS) as proposed by George et al. (2008) to produce a draw of the diagonal inverse prior covariance matrix $\underline{V}^{-1}$ and the corresponding vector of inclusion parameters λ of the vectorised coefficient matrix $a = vec(A)$ for the VAR model

$$y_t = Ax_t + u_t,$$

where y_t is a K-dimensional vector of endogenous variables, x_t is a vector of explanatory variabes and the error term is $u_t \sim \Sigma$.

Value

A named list containing two components:

`v_i`	an $M \times M$ inverse prior covariance matrix.
`lambda`	an M-dimensional vector of inclusion parameters.

References

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Examples

```
# Load data
data("e1")
data <- diff(log(e1))

# Generate model data
temp <- create_bvarmodel(data, p = 2, deterministic = "const")
y <- t(temp$data$train$y)
x <- t(temp$data$train$x)
k <- nrow(y)
tt <- ncol(y)
m <- k * nrow(x)

# Obtain SSVS priors using the semiautomatic approach
priors <- ssvs_prior(temp, semiautomatic = c(0.1, 10))
tau0 <- priors$tau0
tau1 <- priors$tau1

# Prior for inclusion parameter
prob_prior <- matrix(0.5, m)

# Priors
a_mu_prior <- matrix(0, m)
a_v_i_prior <- diag(c(tau1^2), m)

# Initial value of Sigma
sigma_i <- solve(tcrossprod(y) / tt)

# Draw parameters
a <- post_normal(y = y, x = x, sigma_i = sigma_i,
                 a_prior = a_mu_prior, v_i_prior = a_v_i_prior)

# Run SSVS
lambda <- ssvs(a = a, tau0 = tau0, tau1 = tau1,
               prob_prior = prob_prior)
```

ssvs_prior *Stochastic Search Variable Selection Prior*

Description

Calculates the priors for a Bayesian VAR model, which employs stochastic search variable selection (SSVS).

Usage

```
ssvs_prior(object, ...)
```

Arguments

object	an object of class "bvarmodel" or "bvecmodel", usually, a result of a call to create_bvarmodel or create_bvecmodel.
...	arguments passed forward to method.

Value

A list containing the vectors of prior standard deviations for restricted and unrestricted variables, respectively.

References

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

See Also

Methods: ssvs_prior.bvarmodel, ssvs_prior.bvecmodel, ssvs_prior.externalforecast.

Examples

```
# Prepare data
data("e1")
data <- diff(log(e1))

# Generate model input
object <- create_bvarmodel(data)

# Obtain SSVS prior
prior <- ssvs_prior(object, semiautomatic = c(.1, 10))
```

`ssvs_prior.bvarmodel` *Stochastic Search Variable Selection Prior*

Description

Calculates the priors for a Bayesian VAR model, which employs stochastic search variable selection (SSVS).

Usage

```
## S3 method for class 'bvarmodel'
ssvs_prior(object, tau = c(0.05, 10), semiautomatic = NULL, ...)
```

Arguments

`object`	an object of class `"bvarmodel"`, usually, a result of a call to `create_bvarmodel` or `create_bvecmodel`.
`tau`	a numeric vector of two elements containing the prior standard errors of restricted variables (τ_0) as its first element and unrestricted variables (τ_1) as its second. Default is `c(0.05, 10)`.
`semiautomatic`	an optional numeric vector of two elements containing the factors by which the standard errors associated with an unconstrained least squares estimate of the VAR model are multiplied to obtain the prior standard errors of restricted (τ_0) and unrestricted (τ_1) variables. This is the semiautomatic approach described in George et al. (2008).
`...`	arguments passed forward to method.

Value

A list containing the vectors of prior standard deviations for restricted and unrestricted variables, respectively.

References

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Examples

```
# Prepare data
data("e1")
train <- diff(log(e1))

# Generate model input
model <- create_bvarmodel(train, p = 2, deterministic = "const",
                          iterations = 5000, burnin = 1000)
```

```
# Obtain SSVS prior
prior <- ssvs_prior(model, semiautomatic = c(.1, 10))
```

`ssvs_prior.bvecmodel` *Stochastic Search Variable Selection Prior*

Description

Calculates the priors for a Bayesian VAR model, which employs stochastic search variable selection (SSVS).

Usage

```
## S3 method for class 'bvecmodel'
ssvs_prior(object, tau = c(0.05, 10), semiautomatic = NULL, ...)
```

Arguments

`object`	an object of class `"bvarmodel"`, usually, a result of a call to `create_bvarmodel` or `create_bvecmodel`.
`tau`	a numeric vector of two elements containing the prior standard errors of restricted variables (τ_0) as its first element and unrestricted variables (τ_1) as its second. Default is `c(0.05, 10)`.
`semiautomatic`	an optional numeric vector of two elements containing the factors by which the standard errors associated with an unconstrained least squares estimate of the VAR model are multiplied to obtain the prior standard errors of restricted (τ_0) and unrestricted (τ_1) variables. This is the semiautomatic approach described in George et al. (2008).
`...`	arguments passed forward to method.

Value

A list containing the vectors of prior standard deviations for restricted and unrestricted variables, respectively. The first Kr elements, those of the loadings, are one: the loadings are not selected over, and their prior is the one the cointegration space prior implies, which the samplers form themselves every draw, so these entries are placeholders that are never read.

References

George, E. I., Sun, D., & Ni, S. (2008). Bayesian stochastic search for VAR model restrictions. *Journal of Econometrics, 142*(1), 553–580. doi:10.1016/j.jeconom.2007.08.017

Examples

```
# Prepare data
data("e6")

# Generate model input
object <- create_bvecmodel(e6, p = 2, r = 1)

# Obtain SSVS prior
prior <- ssvs_prior(object, semiautomatic = c(.1, 10))
```

`stochvol_ksc_1998` *Stochastic Volatility*

Description

Produces a draw of log-volatilities based on Kim, Shephard and Chib (1998).

Usage

```
stochvol_ksc_1998(y, h, sigma, h_init, constant)
```

Arguments

`y`	a $T \times K$ matrix containing the time series.
`h`	a $T \times K$ matrix of the current draw of log-volatilities.
`sigma`	a $K \times 1$ vector of variances of log-volatilities, where the ith element corresponds to the ith column in `y`. Must be finite and positive.
`h_init`	a $K \times 1$ vector of the initial states of log-volatilities, where the ith element corresponds to the ith column in `y`.
`constant`	a $K \times 1$ vector of constants that should be added to y^2 before taking the natural logarithm. The ith element corresponds to the ith column in `y`. Must be finite and positive. See 'Details'.

Details

For each column in `y` the function produces a posterior draw of the log-volatility h for the model

$$y_t = e^{\frac{1}{2}h_t}\epsilon_t,$$

where $\epsilon_t \sim N(0,1)$ and h_t is assumed to evolve according to a random walk

$$h_t = h_{t-1} + u_t,$$

with $u_t \sim N(0, \sigma^2)$.

The implementation follows the algorithm of Kim, Shephard and Chib (1998) and performs the following steps:

1. Perform the transformation $y_t^* = ln(y_t^2 + constant)$.
2. Obtain a sample from the seven-component normal mixture for approximating the log-χ_1^2 distribution.
3. Obtain a draw of log-volatilities.

The posterior precision of the log-volatility is tridiagonal – the random walk contributes the two bands of $D'D$ and the mixture only a diagonal – so the draw is obtained from a banded Cholesky factorisation, which costs $O(T)$ rather than the $O(T^3)$ of a dense one.

The probabilities of the mixture components in step 2 are obtained in logs and shifted by their period-wise maximum before they are exponentiated. Weighting the component densities directly underflows to zero for all seven components once an observation lies far enough out in their tails, which leaves the probabilities of that period undefined and selects a component that does not exist.

The arguments are checked before the first draw and a violation raises an error. `sigma` and `constant` have to be finite and positive, since the algorithm divides by the former and takes the logarithm of a sum containing the latter, and all three of `sigma`, `h_init` and `constant` have to have one element per column of `y`.

The draw uses R's random number generator, so `set.seed` makes it reproducible.

Value

A $T \times K$ matrix of log-volatility draws.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Kim, S., Shephard, N., & Chib, S. (1998). Stochastic volatility. Likelihood inference and comparison with ARCH models. *Review of Economic Studies 65*(3), 361–393. doi:10.1111/1467-937X.00050

Examples

```
data("us_macrodata")
y <- diff(us_macrodata)
h_init <- log(diag(var(y)))
h <- t(matrix(h_init, 3, nrow(y)))
sigma_h <- rep(.05, 3)
const <- rep(.0001, 3)
stochvol_ksc_1998(y, h, sigma_h, h_init, const)
```

`stochvol_ocsn_2007` *Stochastic Volatility*

Description

Produces a draw of log-volatilities based on Omori, Chib, Shephard and Nakajima (2007).

Usage

```
stochvol_ocsn_2007(y, h, sigma, h_init, constant)
```

Arguments

`y`	a $T \times K$ matrix containing the time series.
`h`	a $T \times K$ matrix of the current draw of log-volatilities.
`sigma`	a $K \times 1$ vector of variances of log-volatilities, where the ith element corresponds to the ith column in `y`. Must be finite and positive.
`h_init`	a $K \times 1$ vector of the initial states of log-volatilities, where the ith element corresponds to the ith column in `y`.
`constant`	a $K \times 1$ vector of constants that should be added to y^2 before taking the natural logarithm. The ith element corresponds to the ith column in `y`. Must be finite and positive. See 'Details'.

Details

For each column in `y` the function produces a posterior draw of the log-volatility h for the model

$$y_t = e^{\frac{1}{2}h_t}\epsilon_t,$$

where $\epsilon_t \sim N(0,1)$ and h_t is assumed to evolve according to a random walk

$$h_t = h_{t-1} + u_t,$$

with $u_t \sim N(0, \sigma^2)$.

The implementation follows the algorithm of Omori, Chib, Shephard and Nakajima (2007) and performs the following steps:

1. Perform the transformation $y_t^* = ln(y_t^2 + constant)$.
2. Obtain a sample from the ten-component normal mixture for approximating the log-χ_1^2 distribution.
3. Obtain a draw of log-volatilities.

The ten components approximate the log-χ_1^2 distribution more closely than the seven of `stochvol_ksc_1998`, at a proportional cost in the first step.

The posterior precision of the log-volatility is tridiagonal – the random walk contributes the two bands of $D'D$ and the mixture only a diagonal – so the draw is obtained from a banded Cholesky factorisation, which costs $O(T)$ rather than the $O(T^3)$ of a dense one.

The probabilities of the mixture components in step 2 are obtained in logs and shifted by their period-wise maximum before they are exponentiated. Weighting the component densities directly underflows to zero for all ten components once an observation lies far enough out in their tails, which leaves the probabilities of that period undefined and selects a component that does not exist.

The arguments are checked before the first draw and a violation raises an error. `sigma` and `constant` have to be finite and positive, since the algorithm divides by the former and takes the logarithm of a sum containing the latter, and all three of `sigma`, `h_init` and `constant` have to have one element per column of `y`.

The draw uses R's random number generator, so `set.seed` makes it reproducible.

Value

A $T \times K$ matrix of log-volatility draws.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

Kim, S., Shephard, N., & Chib, S. (1998). Stochastic volatility. Likelihood inference and comparison with ARCH models. *Review of Economic Studies 65*(3), 361–393. doi:10.1111/1467-937X.00050

Omori, Y., Chib, S., Shephard, N., & Nakajima, J. (2007). Stochastic volatility with leverage. Fast and efficient likelihood inference. *Journal of Econometrics 140*(2), 425–449. doi:10.1016/j.jeconom.2006.07.008

Examples

```
data("us_macrodata")
y <- diff(us_macrodata)
h_init <- log(diag(var(y)))
h <- t(matrix(h_init, 3, nrow(y)))
sigma_h <- rep(.05, 3)
const <- rep(.0001, 3)
stochvol_ocsn_2007(y, h, sigma_h, h_init, const)
```

---

summary.bvarmodel *Summarising Bayesian VAR Coefficients*

Description

summary method for class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
summary(object, ci = 0.95, period = NULL, ...)

## S3 method for class 'summary.bvarmodel'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

object	an object of class 'bvarmodel'.
ci	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
period	integer. Index of the period, for which the summary statistics should be generated. Only used for TVP or SV models. Default is NULL, so that the posterior draws of the last time period are used.
...	further arguments passed to or from other methods.
x	an object of class 'summary.bvarmodel', usually, a result of a call to summary.bvarmodel.
digits	the number of significant digits to use when printing.

Value

summary.bvarmodel returns a list of class 'summary.bvarmodel', which contains the following components:

coefficients	A list of various summary statistics of the posterior draws of the VAR coefficients.
sigma	A list of various summary statistics of the posterior draws of the variance-covariance matrix.
specifications	a list containing information on the model specification.

---

summary.bvecmodel *Summarising Bayesian VEC Coefficients*

Description

summary method for class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
summary(object, ci = 0.95, period = NULL, ...)

## S3 method for class 'summary.bvecmodel'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

object	an object of class 'bvecmodel'.
ci	a numeric between 0 and 1 specifying the probability of the credible band. Defaults to 0.95.
period	integer. Index of the period, for which the summary statistics should be generated. Only used for TVP or SV models. Default is NULL, so that the posterior draws of the last time period are used.
...	further arguments passed to or from other methods.
x	an object of class 'summary.bvecmodel', usually, a result of a call to summary.bvecmodel.
digits	the number of significant digits to use when printing.

Value

summary.bvecmodel returns a list of class 'summary.bvecmodel', which contains the following components:

a	A list of various summary statistics of the posterior draws of the VEC coefficients.
sigma	A list of various summary statistics of the posterior draws of the variance-covariance matrix.
model	a list containing information on the model specification.

---

summary.expandingwindow

Summarising Bayesian Models

Description

summary method for class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
summary(object, ...)
```

Arguments

object	an object of class 'expandingwindow'.
...	further arguments passed to or from other methods.

Details

The method passes the last element of the expanding window model list to its corresponding summary function; assuming that this object contains the model with the highest amount of available observations.

Value

A list of class 'summary.bvarmodel' or 'summary.bvecmodel'.

---

summary.modellist *Summarising Bayesian Models*

Description

summary method for class 'modellist'.

Usage

```
## S3 method for class 'modellist'
summary(object, ...)
```

Arguments

object	an object of class 'modellist'.
...	further arguments passed to or from other methods.

Value

A list of class 'summary.modellist'.

---

sur_const_to_tvp *SUR Matrix Transformation*

Description

Transforms a dense matrix of dimensions $KT \times M$ into a sparse block diagonal matrix of dimensions $KT \times MT$.

Usage

```
sur_const_to_tvp(z, k, tt)
```

Arguments

z	a $KT \times M$ matrix.
k	integer of the number of endogenous variables.
tt	integer of the number of observations.

Value

A sparse block diagonal matrix of dimensions $KT \times MT$.

Examples

```
# Specify the dimensions of the dense matrix
k <- 2
tt <- 5
m <- 3

# Generate artificial data
z <- matrix(NA, k * tt, m)
for (i in 1:tt) {
  z[(i - 1) * k + 1:k, ] <- i
}

# Perform transformation
sur_const_to_tvp(z, k, tt)
```

`thin.bvarmodel` *Thinning Posterior Draws*

Description

Thins the MCMC posterior draws in an object of class 'bvarmodel'.

Usage

```
## S3 method for class 'bvarmodel'
thin(x, thin = 10, ...)
```

Arguments

`x`	an object of class 'bvarmodel'.
`thin`	an integer specifying the thinning interval between successive values of posterior draws.
`...`	further arguments passed to or from other methods.

Value

An object of class 'bvarmodel'.

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(e1, p = 2, deterministic = "const",
```

```
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Thinning
model <- thin(model, 2)
```

`thin.bvecmodel` *Thinning Posterior Draws*

Description

Thins the MCMC posterior draws in an object of class 'bvecmodel'.

Usage

```
## S3 method for class 'bvecmodel'
thin(x, thin = 10, ...)
```

Arguments

`x`	an object of class 'bvecmodel'.
`thin`	an integer specifying the thinning interval between successive values of posterior draws.
`...`	further arguments passed to or from other methods.

Value

An object of class 'bvecmodel' with every block of draws thinned.

Examples

```
# Load data
data("e6")

# Create model
model <- create_bvecmodel(e6, p = 2, r = 1,
                          const = "unrestricted",
```

```
                          iterations = 20, burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Thinning
model <- thin(model, 2)
```

`thin.expandingwindow` *Thinning Posterior Draws*

Description

Thins the MCMC posterior draws of the elements in an object of class 'expandingwindow'.

Usage

```
## S3 method for class 'expandingwindow'
thin(x, thin = 10, ...)
```

Arguments

`x`	an object of class 'expandingwindow'.
`thin`	an integer specifying the thinning interval between successive values of posterior draws.
`...`	further arguments passed to or from other methods.

Value

An object of class 'expandingwindow'.

thin.modellist *Thinning Posterior Draws*

Description

Thins the MCMC posterior draws of the elements in an object of class 'modellist'.

Usage

```
## S3 method for class 'modellist'
thin(x, thin = 10, ...)
```

Arguments

x	an object of class 'modellist'.
thin	an integer specifying the thinning interval between successive values of posterior draws.
...	further arguments passed to or from other methods.

Value

An object of class 'modellist'.

time_variation_test *Test for Time Variation*

Description

Generic function used to compute Bayes factors for time variation in the coefficients and volatilities of a model with time varying parameters.

Usage

```
time_variation_test(object, ...)
```

Arguments

object	an object with suitable posterior draws passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: time_variation_test.bvarmodel, time_variation_test.bvecmodel, and for a list of models, the windows of an expanding window comparison or a folder of stored models, time_variation_test.modellis time_variation_test.expandingwindow and time_variation_test.bvarfolder.

`time_variation_test.bvarmodel`

Test for Time Variation in a VAR or VEC Model

Description

Computes the Bayes factors of Chan (2018) for time variation in each coefficient, each covariance coefficient and, under stochastic volatility, each log-volatility of a VAR or VEC model with time varying parameters, from the draws of a single estimation of that model.

Usage

```
## S3 method for class 'bvarmodel'
time_variation_test(object, joint = TRUE, batches = 20, ...)

## S3 method for class 'bvecmodel'
time_variation_test(object, joint = TRUE, batches = 20, ...)

## S3 method for class 'modellist'
time_variation_test(object, ...)

## S3 method for class 'expandingwindow'
time_variation_test(object, ...)
```

Arguments

object	an object of class "bvarmodel" with tvp = TRUE, whatever its error term, or of class "bvecmodel" with the same, whose priors were set with coef$omega_v or, under stochastic volatility, sigma$omega_v in add_priors and whose posterior was drawn by add_posterior_coefficients.
joint	logical. Should the Bayes factor for the time variation of every state of a block at once be reported as well? Default is TRUE.
batches	integer. The number of batches the draws are split into for the numerical standard errors. Default is 20.
...	further arguments, which are ignored.

Details

Under the prior `omega_v` a random walk $x_t = x_{t-1} + v_t$, $v_t \sim N(0, \omega^2)$, is estimated in the non-centred form $x_t = x_0 + \omega \tilde{x}_t$ of Frühwirth-Schnatter and Wagner (2010), with $\tilde{x}_t$ a standard random walk and the prior $\omega \sim N(0, V_\omega)$. The state does not move exactly when $\omega = 0$, which is a point inside the prior, so the Bayes factor of the model in which the state varies against the one in which it is constant is the Savage-Dickey density ratio

$$BF = \frac{p(\omega = 0)}{p(\omega = 0 | y)}.$$

The numerator is the density of the prior at zero. The denominator is estimated by the average over the draws of the density at zero of the conditional posterior of ω, which the sampler stores for every draw in `omega_log_zero` (Chan 2018). A positive log Bayes factor favours time variation. On the scale of Kass and Raftery (1995), values of the log Bayes factor between 1 and 3 are positive evidence, between 3 and 5 strong evidence and above 5 very strong evidence, and the same values with a negative sign are evidence for a constant state.

The joint Bayes factor of a block compares the model in which every state of the block varies with the model in which none does. It is not a test of whether at least one state varies: every state that is constant costs the joint model about the same as one of its own Bayes factors against time variation, so a block in which one state moves and many do not can come out against time variation as a whole. Conversely, the joint Bayes factor can be far larger than the individual ones: each state may have posterior mass near zero on its own while little of the joint posterior sits where all of them are near zero at once. Chan (2018, Table 2 and footnote 8) reports log Bayes factors of about 3 for each of two volatilities of Italian inflation and of 235 for the two together.

The estimate is least precise where the Bayes factor is large, since the ordinates are then an average of small numbers (Chan 2018, section 2.3). The numerical standard error of each log Bayes factor is obtained by the delta method from batch means, and says how much of the value is owed to the length of the chain. Where it is large, more draws are needed before the value is read as more than its sign.

Each block chooses its prior separately in `add_priors`, and only the blocks estimated under `omega_v` are reported.

For a VEC model the coefficients are the loadings, labelled by the error correction term they load on (`ect1`, `ect2`, ...), followed by the other coefficients. The cointegration space itself follows a state equation with a fixed variance, which has no prior to test against and is not reported.

Value

A data frame of class `"bvartimevar"` with one row per state and, if `joint = TRUE`, one per block, and the columns

`block` `"coefficients"`, `"covariances"` or `"volatilities"`.

`equation` the endogenous variable of the equation the state belongs to; `NA` for the joint rows.

`term` the regressor of a coefficient, the variable whose error a covariance coefficient loads on, `"log-volatility"`, or `"(joint)"`.

`log_bf` the log Bayes factor in favour of time variation.

`nse` its numerical standard error.

References

Chan, J. C. C. (2018). Specification tests for time-varying parameter models with stochastic volatility. *Econometric Reviews, 37*(8), 807–823. doi:10.1080/07474938.2016.1167948

Frühwirth-Schnatter, S., & Wagner, H. (2010). Stochastic model specification search for Gaussian and partial non-Gaussian state space models. *Journal of Econometrics, 154*(1), 85–100. doi:10.1016/j.jeconom.2009.07.003

Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association, 90*(430), 773–795. doi:10.1080/01621459.1995.10476572

Examples

```
# Load data
data("e1")
e1 <- diff(log(e1)) * 100

# Generate model
model <- create_bvarmodel(e1, p = 1, deterministic = "const", tvp = TRUE,
                          error = "sv", iterations = 100, burnin = 50)

# The non-centred prior on the coefficients and the log-volatilities
model <- add_priors(model,
                    coef = list(v_i = 1 / 10, omega_v = 0.001),
                    sigma = list(mu = 0, v_i = 1 / 100, omega_v = 0.1,
                                 state_variance = 0.05, offset = 1e-4))
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Bayes factors for time variation
time_variation_test(model)
```

---

transform_variables *Apply Transformations*

Description

Transforms the columns of a time-series object using the seven transformation codes of FRED-MD/FRED-QD.

Usage

```
transform_variables(x, code)
```

Arguments

x	a time-series object.
code	a named integer vector of transformation codes, one of 1:7 – see 'Details'. Names must match columns of x; a column with no entry is left untransformed (code 1). For a single series code may also be one unnamed code.

Details

The seven codes, applied column by column, are

1 no transformation, x_t

2 first difference, Δx_t

3 second difference, $\Delta^2 x_t$

4 logarithm, $\log(x_t)$

5 first difference of the logarithm, $\Delta \log(x_t)$

6 second difference of the logarithm, $\Delta^2 \log(x_t)$

7 first difference of the growth rate, $\Delta(x_t/x_{t-1} - 1)$

Codes 4:6 need a strictly positive series.

A transformation loses as many leading observations as it reaches back – one for codes 2 and 5, two for 3, 6 and 7, the last being the difference of a growth rate that is itself taken over one period – rather than shortening the series: those leading periods become NA, so every column keeps the time index of x and can still be combined with ts.intersect or passed to create_bvarmodel.

Value

A time-series object of the same shape as x: a vector series for a vector series, and a matrix of series with the same columns otherwise.

References

McCracken, M. W., & Ng, S. (2016). FRED-MD: A monthly database for macroeconomic research. *Journal of Business & Economic Statistics, 34*(4), 574–589.

McCracken, M. W., & Ng, S. (2021). FRED-QD: A quarterly database for macroeconomic research. *Federal Reserve Bank of St. Louis Review, 103*(1), 1–44.

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvarmodel(), add_initial_values.bvecmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), aggregation_weights(), combine_models(), create_bvarmodel(), create_bvecmodel(), use_expanding_window.bv
use_expanding_window.bvecmodel()

Examples

```
data("us_macrodata")

code <- c(Dp = 2, u = 1, r = 2)
x <- transform_variables(us_macrodata, code)
```

`uk_macrodata` *UK interest and inflation rate data*

Description

The data set contains quarterly time series for two UK interest rates and inflation from 1957Q4 to 2009Q3. It was produced from the supplementary material to Koop, León-González and Strachan (2011).

Usage

```
data("uk_macrodata")
```

Format

A named time-series object with 210 rows and 3 variables:

rs short term rate: Treasury bills. This should be series 'FITB_PA' of the IMF's International Financial Statistics database.

rs long term rate: Government bonds. This should be series 'FIGB_PA' of the IMF's International Financial Statistics database.

Dp inflation: Quarterly change in the log CPI annualized by multiplying the change by 400.

References

Koop, G., León-González, R., & Strachan R. W. (2011). Bayesian inference in a time varying cointegration model. *Journal of Econometrics, 165*(2), 210–220. doi:10.1016/j.jeconom.2011.07.007

`us_macrodata` *US macroeconomic data*

Description

The data set contains quarterly time series for the US CPI inflation rate, unemployment rate, and Fed Funds rate from 1959Q2 to 2007Q4. It was produced from file "US_macrodata.csv" of the data sets associated with Chan, Koop, Poirier and Tobias (2019). Raw data are available at https://web.ics.purdue.edu/~jltobias/second_edition/Chapter20/code_for_exercise_1/US_macrodata.csv.

Usage

```
data("us_macrodata")
```

Format

A named time-series object with 195 rows and 3 variables:

Dp CPI inflation rate.

u unemployment rate.

r Fed Funds rate.

References

Chan, J., Koop, G., Poirier, D. J., & Tobias J. L. (2019). *Bayesian econometric methods* (2nd ed.). Cambridge: Cambridge University Press.

---

use_expanding_window *Expanding Window Estimation*

Description

Creates objects for expanding window posterior simulation.

Usage

```
use_expanding_window(object, ...)
```

Arguments

`object`	an object containing model specifications and input data. Usually, the output of a call to `create_bvarmodel` or `create_bvecmodel`.
`...`	arguments passed forward to method.

Value

The value returned by the method for the class of `object`, as described on the pages of the methods.

See Also

Methods: `use_expanding_window.bvarmodel`, `use_expanding_window.bvecmodel`, `use_expanding_window.modellis`

use_expanding_window.bvarmodel
Expanding Window Estimation

Description

Creates objects for expanding window posterior simulation.

Usage

```
## S3 method for class 'bvarmodel'
use_expanding_window(object, start, ...)
```

Arguments

object	an object of class 'bvarmodel' containing model specifications and input data. Usually, the output of a call to create_bvarmodel.
start	the start period of the prediction of the first iteration of the expanding window approach.
...	arguments passed forward to method.

Value

A list of class 'expandingwindow' with one object of class 'bvarmodel' per window. The training sample of the first ends in the period before start, and each further window adds one period. Posterior draws, starting values and forecast input that object already carries belong to the whole sample and are not copied into the windows; a warning says so.

See Also

Other model set-up: add_dummy_variables.bvarmodel(), add_initial_values.bvarmodel(), add_initial_values.bvecmodel(), add_prior_options(), add_priors.bvarmodel(), add_priors.bvecmodel(), aggregation_weights(), combine_models(), create_bvarmodel(), create_bvecmodel(), transform_variables(), use_expanding_window.bvecmodel()

Examples

```
data("us_macrodata")

# Starting period of the forecasting exercise
start_period <- 2007

# Create model
model <- create_bvarmodel(data = us_macrodata,
                          p = 1,
                          deterministic = "none",
                          seasonal = FALSE,
                          tvp = FALSE,
```

```
                          error = "gamma",
                          iterations = 10,
                          burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.

# Create multiple model objects for expanding window
model <- use_expanding_window(model, start = start_period)
```

`use_expanding_window.bvecmodel` *Expanding Window Estimation*

Description

Creates objects for expanding window posterior simulation.

Usage

```
## S3 method for class 'bvecmodel'
use_expanding_window(object, start, ...)
```

Arguments

`object`	an object of class 'bvecmodel' containing model specifications and input data. Usually, the output of a call to `create_bvecmodel`.
`start`	the start period of the prediction of the first iteration of the expanding window approach.
`...`	arguments passed forward to method.

Value

A list of class 'expandingwindow' with one object of class 'bvecmodel' per window. The training sample of the first ends in the period before `start`, and each further window adds one period. Posterior draws, starting values and forecast input that `object` already carries belong to the whole sample and are not copied into the windows; a warning says so.

See Also

Other model set-up: `add_dummy_variables.bvarmodel()`, `add_initial_values.bvarmodel()`, `add_initial_values.bvecmodel()`, `add_prior_options()`, `add_priors.bvarmodel()`, `add_priors.bvecmodel()`, `aggregation_weights()`, `combine_models()`, `create_bvarmodel()`, `create_bvecmodel()`, `transform_variables()`, `use_expanding_window.bvarmodel()`

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 1, r = 1, const = "restricted",
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Create multiple model objects for expanding window
model <- use_expanding_window(model, start = 1998)
```

---

use_expanding_window.modellist

Expanding Window Estimation

Description

Creates objects for expanding window posterior simulation.

Usage

```
## S3 method for class 'modellist'
use_expanding_window(object, start, ...)
```

Arguments

object	a list of objects containing model specifications and input data. Usually, the output of a call to a model creation function such as create_bvarmodel or create_bvecmodel.
start	the start period of the prediction of the first iteration of the expanding window approach.
...	arguments passed forward to method.

Value

A list of class 'modellist', which consists of objects of class 'expandingwindow'.

Examples

```
data("us_macrodata")

# Starting period of the forecasting exercise
start_period <- 2007

# Create model
```

```
model <- create_bvarmodel(data = us_macrodata,
                          p = 1:4,
                          deterministic = "none",
                          seasonal = FALSE,
                          tvp = FALSE,
                          error = "gamma",
                          iterations = 10,
                          burnin = 2)
# Chosen number of iterations and burn-in draws should be much higher.

# Create multiple model objects for expanding window
model <- use_expanding_window(model, start = start_period)
```

---

vec_to_var *Transform a VEC Model to a VAR in Levels*

Description

A generic function used to transform a vector error correction model into its VAR form. The function invokes particular methods which depend on the class of the first argument.

Usage

```
vec_to_var(object, ...)
```

Arguments

object	an object with suitable input data passed forward to method.
...	arguments passed forward to method.

Value

The value returned by the method for the class of object, as described on the pages of the methods.

See Also

Methods: vec_to_var.bvecmodel, vec_to_var.modellist.

`vec_to_var.bvecmodel` *Transform a VEC Model to a VAR in Levels*

Description

An object of class `'bvecmodel'` is transformed into an object of class `'bvarmodel'`, which contains the VAR representation of the model in levels.

Usage

```
## S3 method for class 'bvecmodel'
vec_to_var(object, ...)
```

Arguments

`object`	an object of class `'bvecmodel'`, usually, the result of a call to `create_bvecmodel`, optionally already estimated with `add_posterior_coefficients`.
`...`	arguments passed forward to method.

Details

A VEC model and its VAR representation in levels are the same model in two parameterisations, so a posterior draw of the one is a posterior draw of the other. The transformation is therefore a change of basis, which is applied draw by draw:

$$A_1 = A_0 + \Pi^y + \Gamma_1, \quad A_i = \Gamma_i - \Gamma_{i-1}, \quad A_p = -\Gamma_{p-1}$$

for the endogenous variables and, analogously, one block later for the unmodelled, non-deterministic variables

$$B_0 = \Upsilon_0, \quad B_1 = \Pi^x - \Upsilon_0 + \Upsilon_1, \quad B_j = \Upsilon_j - \Upsilon_{j-1}, \quad B_s = -\Upsilon_{s-1}.$$

A_0 is the identity matrix unless the model is structural, in which case it is the matrix of contemporaneous coefficients. It enters A_1 because it multiplies Δy_t in the VEC model and therefore y_t in its VAR representation, which leaves $A_0 y_{t-1}$ on the right-hand side. The contemporaneous coefficients themselves are not affected by the transformation and are carried over unchanged. Deterministic terms that entered the model unrestricted carry over unchanged. Those restricted to the cointegration space become ordinary regressors of the VAR, entering after the unrestricted ones, and are dropped when the cointegration rank is zero, in which case they had no effect on the model. The coefficient transformation itself is performed by the C++ implementation of the package's model library, which is also used to obtain forecasts of VEC models.

The data matrices are reconstructed from the data of the VEC model, so the resulting object covers exactly the same periods. Since a VEC model of lag order p and its VAR representation use the same number of observations, no observations are gained or lost. The levels of the endogenous variables are recovered from their differences and their first lag in the error correction term, which requires both to be on the same scale. If the error correction term was scaled or centred with `scale_error_correction`, the function therefore expects `rescale_error_correction` to have been applied before the transformation.

Note that the covariance matrix of the error term is not affected by the transformation, since both parameterisations describe the same error term.

Note also that the elements `priors` and `initial` of the input object are not carried over, because they describe the coefficients of the VEC model. Use `add_priors` and `add_initial_values` if the resulting model should be estimated in its VAR form. The same applies to the results of variable selection algorithms: a coefficient of the VAR representation is the sum of multiple coefficients of the VEC model, so no single inclusion indicator describes it and the draws of those indicators are dropped.

A model with time varying parameters is a sequence of VEC models, one per period, and so is its VAR representation: the transformation is applied to the draws of each period in turn, which yields a VAR model with time varying parameters whose coefficient draws are the paths of the coefficients in levels. Its forecasts start from the coefficients of the last period of the sample, and its impulse responses and variance decompositions are those of the period given in their argument `period`. The draws of the variances of the state equations of the VEC coefficients and of ρ are dropped, since they describe how the coefficients of the VEC model drift and have no counterpart among the coefficients in levels. For the same reason its forecasts hold the coefficients and volatilities at the last period rather than simulating them forward: `model$forecast_states` is set to `"hold"`.

A model with constant coefficients and stochastic volatility has a VAR representation with the same covariance block and the same log-volatility random walk, so its forecasts simulate the volatility forward like those of any other VAR model with stochastic volatility; see `add_posterior_forecasts.bvarmodel`. A model of that kind estimated with an earlier version of the package lacks `posterior$u_sigma_inv$sigma`, the variance of the log-volatility innovations, and its VAR representation holds the volatility instead.

Value

An object of class `'bvarmodel'` with the elements `model` and `data` of the VAR in levels and, if `object` was estimated, element `posterior` with the transformed draws, including `loglik` if it was present. Priors and initial values are not carried over.

References

Lütkepohl, H. (2006). *New introduction to multiple time series analysis* (2nd ed.). Berlin: Springer.

See Also

Other post-estimation analysis: `add_sign_restrictions.bvarmodel()`, `add_sign_zero_restrictions.bvarmodel()`, `fevd.bvarmodel()`, `fevd.bvecmodel()`, `historical_decomposition()`, `historical_decomposition.bvarmodel()`, `irf.bvarmodel()`, `irf.bvecmodel()`, `multipliers()`, `multipliers.bvarmodel()`, `multipliers.bvecmodel()`, `predict.bvarmodel()`, `spillover.bvarmodel()`, `spillover.bvecmodel()`, `split_quantile_grid()`

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 2, r = 1, const = "restricted",
                          iterations = 10, burnin = 10)
```

```
# Chosen number of iterations and burn-in should be much higher.

# Add prior specifications
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Obtain posterior draws
model <- add_posterior_coefficients(model)

# Transform to a VAR model in levels
object <- vec_to_var(model)

# The result can be used like any other VAR model, e.g. for forecasting
object <- add_forecast_input(object, n_ahead = 4)
object <- add_posterior_forecasts(object)
```

`vec_to_var.expandingwindow`

Transform VEC Models to VARs in Levels

Description

The models of the estimation windows of an object of class `'expandingwindow'` are transformed into their VAR representation in levels, which is the form in which forecasts, forecast errors and out-of-sample selection criteria are obtained for VEC models.

Usage

```
## S3 method for class 'expandingwindow'
vec_to_var(object, ...)
```

Arguments

`object`	an object of class `'expandingwindow'`.
`...`	arguments passed forward to method.

Value

An object of class `'expandingwindow'`.

vec_to_var.modellist *Transform a VEC Model to a VAR in Levels*

Description

The elements of an object of class 'modellist' are transformed into their VAR representation in levels.

Usage

```
## S3 method for class 'modellist'
vec_to_var(object, ...)
```

Arguments

object	an object of class 'modellist'.
...	arguments passed forward to method.

Value

An object of class 'modellist'.

window.bvarmodel *Time Series Windows*

Description

Restricts the observations in the training sample of a model of class 'bvarmodel' to the specified time window.

Usage

```
## S3 method for class 'bvarmodel'
window(x, start = NULL, end = NULL, ...)
```

Arguments

x	an object of class 'bvarmodel'.
start	the start time of the period of interest.
end	the end time of the period of interest.
...	further arguments passed to or from other methods.

Details

Posterior draws that form a path over the periods of the training sample are cut to the periods that remain, so that they still refer to the same observations as the data. These are the coefficients of a model with time varying parameters, the draws of the error term that vary by period – as under stochastic volatility – and the pointwise log-likelihood. Draws that do not vary by period are left as they are. The posterior of a discounted model, which holds the moments of every period rather than draws, is cut to the same periods. In either case the draws or moments that remain were estimated from the whole sample; re-estimate the model to condition on the window alone.

Value

An object of class 'bvarmodel'.

---

window.bvecmodel *Time Series Windows*

Description

Restricts the observations in the training sample of a model of class 'bvecmodel' to the specified time window.

Usage

```
## S3 method for class 'bvecmodel'
window(x, start = NULL, end = NULL, ...)
```

Arguments

x	an object of class 'bvecmodel'.
start	the start time of the period of interest.
end	the end time of the period of interest.
...	further arguments passed to or from other methods.

Details

Posterior draws that form a path over the periods of the training sample are cut to the periods that remain, so that they still refer to the same observations as the data. These are the coefficients and cointegration vectors of a model with time varying parameters, the draws of the error term that vary by period – as under stochastic volatility – and the pointwise log-likelihood. Draws that do not vary by period, such as those of ρ, are left as they are.

Value

An object of class 'bvecmodel'.

window.modellist *Time Series Windows*

Description

Forwards model input to the same function for individual models.

Usage

```
## S3 method for class 'modellist'
window(x, start = NULL, end = NULL, ...)
```

Arguments

x	an object of class 'modellist'.
start	the start time of the period of interest.
end	the end time of the period of interest.
...	further arguments passed to or from other methods.

Value

An object of class 'modellist'.

write_bayests_tree *Write and Read the Tree of a BayesTS Model File*

Description

Writes a nested list into an HDF5 file as a model's tree, or reads such a tree back as a nested list. These are the steps beneath `write_to_hdf5` and `read_model_from_hdf5`, exported for packages that keep models of their own in BayesTS files – the dynamic factor models of **dfmtools**, for instance.

Usage

```
write_bayests_tree(tree, filename, group = "")

read_bayests_tree(filename, group = "", draws = NULL)
```

Arguments

tree	a named list. An element that is itself a list is a group; any other element is a dataset. See 'Details'.
filename	path to the HDF5 file.
group	the group the tree hangs under inside its file. Defaults to "", the root of the file, which is where a file holding a single model puts it. The spelling is that of the BayesTS command line's --group flag: a leading slash and no trailing slash.
draws	the draws to read from the blocks of /posterior, as their positions in the chain. Defaults to NULL, every draw; see read_model_from_hdf5.

Details

A model package translates its object into the layout BayesTS reads – /model, /data, /priors, /initial and /posterior, named and shaped as the sampler expects – and hands the result to write_bayests_tree(). What is left to this function is what every model shares: refusing to overwrite a model, undoing a write that fails half-way, and the attributes that let a value be read back as what it was. **It does not check the tree against any sampler**; a file that BayesTS refuses is the translation's mistake, and bayests check is the way to find it.

The tree is mapped onto the file as follows:

- An element that is a list becomes a group of the same name, and its elements go below it. NULL elements are left out.
- The element .attributes of a list is not a group but the attributes of the group it is in: a named list of values, each written as an attribute. This is where /model keeps the specification – algorithm, k, p and the rest – and where rclass, the class read_model_from_hdf5 returns, belongs, and rpackage, the package that defines that class, whose namespace the reader loads for its methods.
- Any other element becomes a dataset. HDF5 stores dimensions in the reverse of R's order, so a matrix of tt rows and k columns is a (k, tt) dataset, which is what BayesTS calls (tt, k) "on paper". A vector is a one-dimensional dataset and is marked so that it is read back as a vector. A time series carries its column names and tsp, and draws of class mcmc their start, end and thinning interval, so that both are read back as what they were.

Without a group the tree is the whole file, so an existing file is refused; with one, only that group has to be free, which is how a second model is added beside the first. A write that cannot be completed raises the error and undoes what the call created.

read_bayests_tree() is the inverse, for any model file, including one written by the BayesTS command line: groups become lists, the attributes of each group its element .attributes, and datasets values, restored through the attributes above where the file carries them and as matrices where it does not.

A file whose /model carries rclass is read by read_model_from_hdf5 through from_bayests_tree, which is the method a model package provides to turn the tree back into its object.

Value

write_bayests_tree() returns the path to the written file, invisibly. read_bayests_tree() returns the tree as a nested list.

See Also

from_bayests_tree, write_to_hdf5, read_model_from_hdf5.

Examples

```
path <- tempfile(fileext = ".h5")
tree <- list(
  "model" = list(".attributes" = list("algorithm" = "Example", "k" = 2L)),
  "data" = list("train" = list("y" = matrix(rnorm(20), 10, 2)))
)
write_bayests_tree(tree, filename = path)
str(read_bayests_tree(path))
```

---

write_to_hdf5 *Export to HDF5 File*

Description

A generic function used to export the content of models into HDF5 files. The function invokes particular methods which depend on the class of the first argument.

Usage

```
write_to_hdf5(object, ...)
```

Arguments

object	an object of a class, for which a method should be called.
...	arguments passed forward to method.

Details

Before the method is called, the size of object is compared with options(bvartools.size_warning), and a warning is given if it is larger, since that is about the disk space the files will need. See expected_model_size.

Value

The value returned by the method for the class of object, as described on the pages of the methods.

See Also

Methods: write_to_hdf5.bvarmodel, write_to_hdf5.bvecmodel, write_to_hdf5.expandingwindow, write_to_hdf5.modellist.

`write_to_hdf5.bvarmodel` *Export to HDF5 File*

Description

Exports the content of an object of class 'bvarmodel' to an HDF5 file.

Usage

```
## S3 method for class 'bvarmodel'
write_to_hdf5(object, filename, group = "", ...)
```

Arguments

`object`	list of class 'bvarmodel'.
`filename`	path to the file, in which output should be stored.
`group`	the group the model's tree should hang under inside its file. Defaults to `""`, the root of the file, which is where a file holding a single model puts it. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

With a `group` every path is written under it instead of at the root, so one file can hold several models side by side. The spelling is the one the BayesTS command line uses for its `--group` flag: a leading slash and no trailing slash, with `""` for the root. Intermediate groups are created as needed, and `list_models_in_hdf5` reports which groups of a file hold a model.

What must not already be there is the model rather than the file. Without a `group` the model is the whole file, so an existing file is refused; with one, only that group has to be free, which is what lets a second model be added beside the first.

A write that cannot be completed raises the error rather than absorbing it, and undoes what the call created: a file it made is removed, a group it added to an existing file is unlinked on its own so the models beside it survive. A retry then meets the original problem rather than the leftovers.

Value

The path to the written file, invisibly.

Examples

```
# Load data
data("e1")
train <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(data = train,
```

```
                          p = 1,
                          deterministic = "const",
                          iterations = 10,
                          burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1),
                    sigma = list(df = 3, scale = 1))

# Add initial values
model <- add_initial_values(model)

# Save model
path_to_model <- tempfile(fileext = ".h5")
write_to_hdf5(model, filename = path_to_model)
```

`write_to_hdf5.bvecmodel` *Export to HDF5 File*

Description

Exports the content of an object of class 'bvarmodel' to an HDF5 file.

Usage

```
## S3 method for class 'bvecmodel'
write_to_hdf5(object, filename, group = "", ...)
```

Arguments

`object`	list of class 'bvarmodel'.
`filename`	path to the file, in which output should be stored.
`group`	the group the model's tree should hang under inside its file. Defaults to `""`, the root of the file, which is where a file holding a single model puts it. See 'Details'.
`...`	further arguments passed to or from other methods.

Details

With a `group` every path is written under it instead of at the root, so one file can hold several models side by side. The spelling is the one the BayesTS command line uses for its `--group` flag: a leading slash and no trailing slash, with `""` for the root. Intermediate groups are created as needed, and `list_models_in_hdf5` reports which groups of a file hold a model.

What must not already be there is the model rather than the file. Without a group the model is the whole file, so an existing file is refused; with one, only that group has to be free, which is what lets a second model be added beside the first.

A write that cannot be completed raises the error rather than absorbing it, and undoes what the call created: a file it made is removed, a group it added to an existing file is unlinked on its own so the models beside it survive. A retry then meets the original problem rather than the leftovers.

Value

The path to the written file, invisibly.

Examples

```
# Load data
data("e6")
e6 <- e6 * 100

# Generate model
model <- create_bvecmodel(e6, p = 1, r = 1, const = "restricted",
                          iterations = 10, burnin = 10)
# Chosen number of iterations and burn-in should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1, v_i_det = 1 / 10),
                    coint = list(v_i = 0, p_tau_i = 1),
                    sigma = list(df = "k", scale = 1))

# Add initial values
model <- add_initial_values(model)

# Save model
path_to_model <- tempfile(fileext = ".h5")
write_to_hdf5(model, filename = path_to_model)
```

---

write_to_hdf5.expandingwindow

Export to HDF5 File

Description

Exports the content of an object of class 'expandingwindow' to multiple HDF5 file.

Usage

```
## S3 method for class 'expandingwindow'
write_to_hdf5(object, folder, ...)
```

Arguments

object	an object of class 'expandingwindow'.
folder	path to the folder where the individual models in argument object should be saved.
...	further arguments passed to or from other methods.

Value

The paths of the written files, invisibly, one per window, in the subfolder of folder created for the exercise.

Examples

```
# Load data
data("e1")
train <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(data = train,
                          p = 0:2,
                          deterministic = "const",
                          iterations = 10,
                          burnin = 10)
# Number of iterations and burnin should be much higher.

# Use expanding window
model <- use_expanding_window(model, start = 1982.5)

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1),
                    sigma = list(df = 3, scale = 1))

# Add initial values
model <- add_initial_values(model)

# Save model
path_to_target_directory <- tempdir()
write_to_hdf5(model, folder = path_to_target_directory)
```

---

write_to_hdf5.modellist

Export to HDF5 File

Description

Exports the content of an object of class 'modellist' to multiple HDF5 file.

Usage

```
## S3 method for class 'modellist'
write_to_hdf5(object, folder, ...)
```

Arguments

object	an object of class 'modellist'.
folder	path to the folder where the individual models in argument object should be saved.
...	further arguments passed to or from other methods.

Value

The paths of the written files, invisibly, one per model and, for a member that is itself a collection, one per model in it.

Examples

```
# Load data
data("e1")
train <- diff(log(e1)) * 100

# Create model
model <- create_bvarmodel(data = train,
                          p = 0:4,
                          deterministic = "const",
                          iterations = 10,
                          burnin = 10)
# Number of iterations and burnin should be much higher.

# Add priors
model <- add_priors(model,
                    coef = list(v_i = 1),
                    sigma = list(df = 3, scale = 1))

# Add initial values
model <- add_initial_values(model)

# Save model
path_to_target_directory <- tempdir()
write_to_hdf5(model, folder = path_to_target_directory)
```

Index

∗ **artificial data**
generate_artificial_var, 152
generate_artificial_vec, 156
∗ **datasets**
at_macrodata, 94
e1, 140
e6, 141
uk_macrodata, 284
us_macrodata, 284
∗ **model comparison**
add_forecast_errors.bvarmodel, 12
add_forecast_errors.bvecmodel, 13
add_predictive_loglik, 51
add_predictive_loglik.bvarmodel, 54
add_predictive_loglik.bvecmodel, 56
aggregate_forecasts, 85
align_model_obs.modellist, 90
analysis_of_stored_models, 91
choose_best_model.selcritlist, 113
create_external_forecast, 135
folder_steps, 149
map_draws, 178
map_models, 179
open_model, 191
open_models, 192
plot_forecast_errors_by_period, 208
pool_forecasts, 212
selection_criteria.bvarmodel, 248
selection_criteria.bvecmodel, 249
selection_criteria.default, 251
selection_criteria.modellist, 255
∗ **model set-up**
add_dummy_variables.bvarmodel, 9
add_initial_values.bvarmodel, 22
add_initial_values.bvecmodel, 24
add_prior_options, 57
add_priors.bvarmodel, 60
add_priors.bvecmodel, 65
aggregation_weights, 88
combine_models, 121
create_bvarmodel, 125
create_bvecmodel, 131
transform_variables, 282
use_expanding_window.bvarmodel, 286
use_expanding_window.bvecmodel, 287
∗ **post-estimation analysis**
add_sign_restrictions.bvarmodel, 77
add_sign_zero_restrictions.bvarmodel, 80
fevd.bvarmodel, 145
fevd.bvecmodel, 148
historical_decomposition, 163
historical_decomposition.bvarmodel, 163
irf.bvarmodel, 170
irf.bvecmodel, 173
multipliers, 186
multipliers.bvarmodel, 187
multipliers.bvecmodel, 190
predict.bvarmodel, 229
spillover.bvarmodel, 257
spillover.bvecmodel, 260
split_quantile_grid, 263
vec_to_var.bvecmodel, 290
∗ **posterior simulation**
add_forecast_input.bvarmodel, 17
add_forecast_input.bvecmodel, 19
add_posterior_coefficients.bvarmodel, 29
add_posterior_coefficients.bvecmodel, 31
add_posterior_forecasts.bvarmodel,

37
add_posterior_forecasts.bvecmodel, 40
add_posterior_loglik.bvarmodel, 45
add_posterior_loglik.bvecmodel, 46
add_seed, 75
bayests_files, 95
bayests_posterior, 96
bvar, 98
bvec, 105
chain_diagnostics, 111
predict.bvecmodel, 230

add_dummy_variables, 9, *52*, *102*, *234*
add_dummy_variables.bvarmodel, 9, *9*
add_dummy_variables.bvarmodel(), *24, 25*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_dummy_variables.bvecmodel, *9*
add_dummy_variables.bvecmodel *(*add_dummy_variables.bvarmodel*)*, 9
add_dummy_variables.expandingwindow, *9*
add_dummy_variables.expandingwindow *(*add_dummy_variables.bvarmodel*)*, 9
add_dummy_variables.modellist, *9*
add_dummy_variables.modellist *(*add_dummy_variables.bvarmodel*)*, 9
add_forecast_errors, 11, *38*, *86*, *103, 104*, *137*, *208*, *215*
add_forecast_errors.bvarmodel, 12, *12*, *14–16*
add_forecast_errors.bvarmodel(), *14*, *53*, *55*, *57*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *248*, *250*, *252*, *256*
add_forecast_errors.bvecmodel, 13, *56*
add_forecast_errors.bvecmodel(), *13*, *53*, *55*, *57*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *248*, *250*, *252*, *256*
add_forecast_errors.expandingwindow, *12*, 14
add_forecast_errors.modellist, *12*, 16
add_forecast_input, *10*, 17, *38*, *40, 41*, *43*, *102, 103*, *136*, *143*, *229*
add_forecast_input.bvarmodel, 17, *17*, *19–21*
add_forecast_input.bvarmodel(), *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *108*, *112*, *231*
add_forecast_input.bvecmodel, *17*, 19
add_forecast_input.bvecmodel(), *18*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *108*, *112*, *231*
add_forecast_input.expandingwindow, *17*, 20
add_forecast_input.externalforecast, *17*
add_forecast_input.externalforecast *(*create_external_forecast*)*, 135
add_forecast_input.forecastpool *(*pool_forecasts*)*, 212
add_forecast_input.modellist, *17*, 21
add_forecast_input.poolwindow *(*pool_forecasts*)*, 212
add_initial_values, 22, *29*, *31, 32*, *69*, *75*, *97*, *100*, *103*, *108*, *112*, *119*, *133*, *291*
add_initial_values.bvarfolder *(*folder_steps*)*, 149
add_initial_values.bvarmodel, 22, *22*, *25–27*
add_initial_values.bvarmodel(), *11*, *25*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_initial_values.bvecmodel, *22*, 24
add_initial_values.bvecmodel(), *11*, *24*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_initial_values.expandingwindow, *22*, 26
add_initial_values.externalforecast, *22*
add_initial_values.externalforecast *(*create_external_forecast*)*, 135
add_initial_values.forecastpool *(*pool_forecasts*)*, 212
add_initial_values.modellist, *22*, 27
add_initial_values.poolwindow *(*pool_forecasts*)*, 212
add_posterior_coefficients, *19*, 28, *38–41*, *43*, *45*, *47, 48*, *50*, *75*, *97*, *100*, *103*, *108*, *111*, *128*, *133*, *142, 143*, *159*, *195*, *263*, *280*, *290*

add_posterior_coefficients.bvarfolder *(folder_steps)*, 149
add_posterior_coefficients.bvarmodel, *28*, 29, *32*, *34, 35*
add_posterior_coefficients.bvarmodel(), *18*, *20*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *108*, *112*, *231*
add_posterior_coefficients.bvecmodel, *28*, 31
add_posterior_coefficients.bvecmodel(), *18*, *20*, *30*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *108*, *112*, *231*
add_posterior_coefficients.expandingwindow, *28*, 33
add_posterior_coefficients.externalforecast, *28*
add_posterior_coefficients.externalforecast *(create_external_forecast)*, 135
add_posterior_coefficients.forecastpool *(pool_forecasts)*, 212
add_posterior_coefficients.modellist, *28*, 35
add_posterior_coefficients.poolwindow *(pool_forecasts)*, 212
add_posterior_forecasts, *14*, *34*, *36*, 37, *42*, *44*, *49, 50*, *54–56*, *85*, *102*, *104*, *128*, *229, 230*, *263*
add_posterior_forecasts.bvarmodel, 37, *37*, *40, 41*, *43*, *291*
add_posterior_forecasts.bvarmodel(), *18*, *20*, *30*, *33*, *41*, *46, 47*, *76*, *96, 97*, *101*, *108*, *112*, *231*
add_posterior_forecasts.bvecmodel, 40, *243*, *245*
add_posterior_forecasts.bvecmodel(), *18*, *20*, *30*, *33*, *39*, *46, 47*, *76*, *96, 97*, *101*, *109*, *112*, *231*
add_posterior_forecasts.expandingwindow, *37*, 41
add_posterior_forecasts.externalforecast, *37*
add_posterior_forecasts.externalforecast *(create_external_forecast)*, 135
add_posterior_forecasts.forecastpool *(pool_forecasts)*, 212
add_posterior_forecasts.modellist, *37*, 43
add_posterior_forecasts.poolwindow *(pool_forecasts)*, 212
add_posterior_loglik, 45, *52*, *54*, *104*, *128*, *246*, *263*
add_posterior_loglik.bvarfolder *(folder_steps)*, 149
add_posterior_loglik.bvarmodel, 45, *45*, *48*, *50*
add_posterior_loglik.bvarmodel(), *18*, *20*, *30*, *33*, *39*, *41*, *47*, *76*, *96, 97*, *101*, *109*, *112*, *231*
add_posterior_loglik.bvecmodel, *45*, 46
add_posterior_loglik.bvecmodel(), *18*, *20*, *30*, *33*, *39*, *41*, *46*, *76*, *96, 97*, *101*, *109*, *112*, *231*
add_posterior_loglik.expandingwindow, *45*, 48
add_posterior_loglik.externalforecast, *45*
add_posterior_loglik.externalforecast *(create_external_forecast)*, 135
add_posterior_loglik.forecastpool *(pool_forecasts)*, 212
add_posterior_loglik.modellist, *45*, 49
add_posterior_loglik.poolwindow *(pool_forecasts)*, 212
add_predictive_loglik, 51, *114*, *211, 212*, *215*, *247*, *253*
add_predictive_loglik(), *13, 14*, *55*, *57*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *248*, *250*, *252*, *256*
add_predictive_loglik.bvarmodel, *53*, 54, *56*
add_predictive_loglik.bvarmodel(), *13, 14*, *53*, *57*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *248*, *250*, *252*, *256*
add_predictive_loglik.bvecmodel, *53*, 56, *243*
add_predictive_loglik.bvecmodel(), *13, 14*, *53*, *55*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *248*, *250*, *252*, *256*
add_predictive_loglik.forecastpool *(pool_forecasts)*, 212
add_predictive_loglik.poolwindow *(pool_forecasts)*, 212
add_prior_options, *30*, 57, *103*, *129*

add_prior_options(), *11*, *24, 25*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_priors, *10*, *23, 24*, *27*, *29*, *31*, *57, 58*, 60, *70*, *100*, *103*, *108*, *117*, *119*, *143*, *280, 281*, *291*
add_priors.bvarfolder *(*folder_steps*)*, 149
add_priors.bvarmodel, 60, *60*, *67*, *69*, *73, 74*, *103*
add_priors.bvarmodel(), *11*, *24, 25*, *59*, *72*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_priors.bvecmodel, *60*, 65, *103*
add_priors.bvecmodel(), *11*, *24, 25*, *59*, *64*, *89*, *122*, *130*, *134*, *283*, *286, 287*
add_priors.expandingwindow, *60*, 73
add_priors.externalforecast, *60*
add_priors.externalforecast *(*create_external_forecast*)*, 135
add_priors.forecastpool *(*pool_forecasts*)*, 212
add_priors.modellist, *60*, 74
add_priors.poolwindow *(*pool_forecasts*)*, 212
add_seed, *23*, *25*, *29*, *32*, *34*, *36*, *42*, *44*, *48*, *50*, 75, *97*, *102*, *150*
add_seed(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *96, 97*, *101*, *109*, *112*, *231*
add_seed.bvarfolder *(*folder_steps*)*, 149
add_seed.externalforecast *(*create_external_forecast*)*, 135
add_seed.forecastpool *(*pool_forecasts*)*, 212
add_seed.poolwindow *(*pool_forecasts*)*, 212
add_sign_restrictions, 76, *81, 82*, *84*, *104*, *146*, *164*, *171*, *259*
add_sign_restrictions.bvarmodel, 77, *77*
add_sign_restrictions.bvarmodel(), *84*, *147*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *263*, *291*
add_sign_restrictions.expandingwindow, *77*
add_sign_restrictions.expandingwindow *(*add_sign_restrictions.bvarmodel*)*, *77*
add_sign_restrictions.modellist, *77*
add_sign_restrictions.modellist *(*add_sign_restrictions.bvarmodel*)*, *77*
add_sign_zero_restrictions, *78*, 80, *92–94*, *164*
add_sign_zero_restrictions.bvarmodel, 80, *80*
add_sign_zero_restrictions.bvarmodel(), *79*, *147*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *263*, *291*
add_sign_zero_restrictions.expandingwindow, *80*
add_sign_zero_restrictions.expandingwindow *(*add_sign_zero_restrictions.bvarmodel*)*, 80
add_sign_zero_restrictions.modellist, *80*
add_sign_zero_restrictions.modellist *(*add_sign_zero_restrictions.bvarmodel*)*, 80
aggregate_forecasts, 85, *136, 137*
aggregate_forecasts(), *13, 14*, *53*, *55*, *57*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*
aggregation_weights, 88, *126*
aggregation_weights(), *11*, *24, 25*, *59*, *64*, *72*, *122*, *130*, *134*, *283*, *286, 287*
align_model_obs, 89, *247*
align_model_obs.modellist, *89*, 90
align_model_obs.modellist(), *13, 14*, *53*, *55*, *57*, *87*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*
analysis_of_stored_models, *13, 14*, *53*, *55*, *57*, *87*, *90*, 91, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*
arias_rubio_ramirez_waggoner_2018, 92
at_macrodata, 94

barplot, *198*, *203*
bayests_files, 95, *150*, *180*, *193*
bayests_files(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *97*, *101*, *109*, *112*, *231*
bayests_posterior, *29*, *32*, *34*, *36*, *42*, *44*, *48*, *50*, *95*, 96
bayests_posterior(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96*, *101*, *109*, *112*, *231*
bvar, *30*, 98, *104*

bvar(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *109*, *112*, *231*
bvartools, *104*
bvartools_model, *13*, *30*, *33*, *39*, *46*, *55*, *57*, 102, *130*, *134*, *143*, *240*
bvec, *104*, 105
bvec(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *112*, *231*

chain_diagnostics, *29*, *31*, 110
chain_diagnostics(), *18*, *20*, *30*, *33*, *39*, *41*, *46, 47*, *76*, *96, 97*, *101*, *109*, *231*
check_discount_specification *(*discount_models*)*, 139
choose_best_model, 113, *150*, *252*, *255*
choose_best_model.selcritlist, 113, *113*
choose_best_model.selcritlist(), *13, 14*, *53*, *55*, *57*, *87*, *90, 91*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*
clusterSetRNGStream, *34*, *36*, *42*, *44*, *49, 50*
coint_kls2010_reparameterise_two, 115
coint_prepare_sur_data, 116
cointspace_prior, *40*, *67*, 117, *244*
combine_models, 121, *138*, *215*
combine_models(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *130*, *134*, *283*, *286, 287*
covar_prepare_data, 122
covar_vector_to_matrix, 124
create_bvarmodel, *10*, *23*, *29, 30*, *38*, *58*, *60, 61*, *74*, *88*, *94*, *98–100*, *102, 103*, 125, *139*, *143*, *153–155*, *164*, *166*, *182*, *247*, *263*, *266–268*, *283*, *285, 286*, *288*
create_bvarmodel(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *122*, *134*, *283*, *286, 287*
create_bvecmodel, *10*, *24*, *31*, *65*, *74*, *94*, *102*, *106–108*, 131, *139*, *143*, *158, 159*, *168*, *184*, *247*, *266–268*, *285*, *287, 288*, *290*
create_bvecmodel(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *122*, *130*, *283*, *286, 287*
create_external_forecast, *86*, 135
create_external_forecast(), *13, 14*, *53*, *55*, *57*, *87*, *90, 91*, *114*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*

discount_models, 139

e1, 140
e6, 141
expected_model_size, *28*, 141, *143*, *297*
expected_model_size.bvarmodel, 142, *142*
expected_model_size.bvecmodel, *142*
expected_model_size.bvecmodel *(*expected_model_size.bvarmodel*)*, 142
expected_model_size.expandingwindow, *142*
expected_model_size.expandingwindow *(*expected_model_size.bvarmodel*)*, 142
expected_model_size.modellist, *142*
expected_model_size.modellist *(*expected_model_size.bvarmodel*)*, 142

fevd, *78*, *84*, *112*, *129*, 144, *148*, *196, 197*, *258, 259*
fevd.bvarfile *(*analysis_of_stored_models*)*, 91
fevd.bvarmodel, *91*, *144*, 145
fevd.bvarmodel(), *79*, *84*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *263*, *291*
fevd.bvecmodel, 148
fevd.bvecmodel(), *79*, *84*, *147*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *263*, *291*
folder_steps, *13, 14*, *53*, *55*, *57*, *87*, *90, 91*, *114*, *138*, 149, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*, *256*
from_bayests_tree, 151, *240*, *296, 297*

generate_artificial_var, 152, *158, 159*
generate_artificial_var(), *160*
generate_artificial_vec, *155*, 156
generate_artificial_vec(), *155*
generate_lower_block_diagonal, 160
get_forecast_errors, 161
get_model_specifications, 162, *236*, *238*

historical_decomposition, 163, *198*
historical_decomposition(), *79*, *84*, *147*, *149*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *263*, *291*
historical_decomposition.bvarmodel, 163, *163*

historical_decomposition.bvarmodel(), *79*, *84*, *147*, *149*, *163*, *172*, *173*, *187*, *189*, *190*, *229*, *260*, *261*, *263*, *291*

inclusion_prior, 165
inclusion_prior.bvarmodel, 166, *166*
inclusion_prior.bvecmodel, *166*, 167
inclusion_prior.externalforecast, *166*
inclusion_prior.externalforecast *(*create_external_forecast*)*, 135
irf, *78*, *84*, *112*, *129*, 169, *173*, *189*, *190*, *196*, *199*
irf.bvarfile *(*analysis_of_stored_models*)*, 91
irf.bvarmodel, *91*, *100*, *164*, *169*, 170
irf.bvarmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *173*, *187*, *189*, *190*, *229*, *260*, *261*, *263*, *291*
irf.bvecmodel, 173
irf.bvecmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *187*, *189*, *190*, *229*, *260*, *261*, *263*, *291*
is_discount_model *(*discount_models*)*, 139

kalman_durbin_koopman_2002, 173

legend, *211*
list.files, *239*
list_models_in_hdf5, 175, *240*, *241*, *298*, *299*
loglik_normal, 176

makeCluster, *34*, *36*, *42*, *43*, *48*, *50*
map_draws, 178
map_draws(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *180*, *192*, *194*, *209*, *215*, *249*, *250*, *252*, *256*
map_models, *150*, 179, *193*
map_models(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *192*, *194*, *209*, *215*, *249*, *250*, *252*, *256*
mcmc, *12*, *30*, *32*, *38*, *41*, *46*, *47*, *55*, *56*, *104*, *240*, *296*
median, *248*
minnesota_prior, 181
minnesota_prior.bvarmodel, *181*, 182
minnesota_prior.bvecmodel, *181*, 184
minnesota_prior.externalforecast, *181*
minnesota_prior.externalforecast *(*create_external_forecast*)*, 135
minnesota_prior.modellist, *181*, 186
model_files *(*open_models*)*, 192
multipliers, 186
multipliers(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *173*, *189*, *190*, *229*, *260*, *261*, *263*, *291*
multipliers.bvarmodel, 187, *187*, *190*
multipliers.bvarmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *173*, *187*, *190*, *229*, *260*, *261*, *263*, *291*
multipliers.bvecmodel, *187*, 190
multipliers.bvecmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *173*, *187*, *189*, *229*, *260*, *261*, *263*, *291*

na.omit, *126*

open_model, *91*, *178*, 191, *193*
open_model(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *194*, *209*, *215*, *249*, *250*, *252*, *256*
open_models, *104*, *149*, *179*, 192, *241*
open_models(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *209*, *215*, *249*, *250*, *252*, *256*

persistence_profiles, 194, *196*
persistence_profiles.bvarmodel, 195
persistence_profiles.bvecmodel, *195*
persistence_profiles.bvecmodel *(*persistence_profiles.bvarmodel*)*, 195
persistence_profiles.expandingwindow *(*persistence_profiles.bvarmodel*)*, 195
persistence_profiles.modellist *(*persistence_profiles.bvarmodel*)*, 195
plot, *209*, *211*, *263*
plot.bvarfevd, 197
plot.bvarhd, 198
plot.bvarirf, 199
plot.bvarmodel, 200
plot.bvarprd, 201
plot.bvarprdlist, 202
plot.bvarspillover, 203
plot.bvarspilloverts, 204
plot.bvecmodel, 204
plot.expandwindbvarprdlist, 205

plot.modellist, 206
plot.selcrit *(*plot.selcritlist*)*, 207
plot.selcritlist, 207, *209*
plot.ts, *203*, *204*, *206*
plot_forecast_errors_by_period, 208, *212*
plot_forecast_errors_by_period(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *215*, *249*, *250*, *252*, *256*
plot_predictive_loglik, *207*, 210
pool_forecasts, 212
pool_forecasts(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *249*, *250*, *252*, *256*
post_bvs, 216
post_coint_kls, 218
post_coint_kls_sur, 221
post_gamma_measurement_variance, 223
post_gamma_state_variance, 224
post_normal, 226
post_normal_sur, 227
predict, *38*
predict.bvarmodel, *100*, *201*, 229, *230*, *231*
predict.bvarmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *173*, *187*, *189*, *190*, *260*, *261*, *263*, *291*
predict.bvecmodel, 230
predict.bvecmodel(), *18*, *20*, *30*, *33*, *39*, *41*, *46*, *47*, *76*, *96*, *97*, *101*, *109*, *112*
predict.expandingwindow, *206*, 231
predict.modellist, *203*, 232
prepare_forecast_input, *18*, *19*, 233
prepare_forecast_input.bvarmodel, 234, *234–236*
prepare_forecast_input.bvecmodel, *234*, 235
print.bvarfile *(*open_model*)*, 191
print.bvarfolder *(*open_models*)*, 192
print.bvarmodel, 236
print.bvarspillover, 237
print.bvecmodel, 237
print.externalforecast *(*create_external_forecast*)*, 135
print.forecastpool *(*pool_forecasts*)*, 212
print.modellist, *207*, 238
print.poolwindow *(*pool_forecasts*)*, 212
print.selcrit *(*selection_criteria*)*, 245
print.selcritlist *(*selection_criteria*)*, 245
print.summary.bvarmodel *(*summary.bvarmodel*)*, 272
print.summary.bvecmodel *(*summary.bvecmodel*)*, 273

quantile, *248*

read_bayests_tree, *151*
read_bayests_tree *(*write_bayests_tree*)*, 295
read_expanding_window_model_from_folder, 239
read_model_from_hdf5, *75*, *97*, *104*, *151*, *176*, *179*, *180*, *191*, 239, *239*, *240*, *295–297*
read_models_from_folder, *152*, *193*, 240
rescale_error_correction, *41*, *52*, *56*, *71*, *120*, 242, *244*, *245*, *290*
rescale_error_correction.bvecmodel, 242, *242*

scale_error_correction, *41*, *70*, *71*, *119*, *120*, 243, *243*, *290*
scale_error_correction.bvecmodel, 244, *244*
selection_criteria, *12*, *46*, *47*, *51*, *54*, *86*, *113*, *114*, *138*, *207*, *211*, *212*, *215*, 245, *245*, *251*
selection_criteria.bvarfolder *(*folder_steps*)*, 149
selection_criteria.bvarmodel, *247*, 248, *250–252*, *255*
selection_criteria.bvarmodel(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *250*, *252*, *256*
selection_criteria.bvecmodel, *247*, 249
selection_criteria.bvecmodel(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249*, *252*, *256*
selection_criteria.default, 251
selection_criteria.default(), *13*, *14*, *53*, *55*, *57*, *87*, *90*, *91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249*, *250*, *256*

selection_criteria.expandingwindow, *247*, 252
selection_criteria.externalforecast, *247*, 254
selection_criteria.modellist, *247*, 255
selection_criteria.modellist(), *13, 14*, *53*, *55*, *57*, *87*, *90, 91*, *114*, *138*, *150*, *178*, *180*, *192*, *194*, *209*, *215*, *249, 250*, *252*
set.seed, *34*, *36*, *42*, *44*, *49, 50*, *78*, *83*, *174*, *270*, *272*
spillover, *78*, *84*, *147*, *203*, *237*, 256, *261*
spillover.bvarmodel, 257, *257*, *261*, *263*
spillover.bvarmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *261*, *263*, *291*
spillover.bvecmodel, 260
spillover.bvecmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260*, *263*, *291*
spillover.expandingwindow, *257*, 261
spillover.modellist, *257*, 262
split_quantile_grid, *128*, 263
split_quantile_grid(), *79*, *84*, *147*, *149*, *163*, *165*, *172, 173*, *187*, *189, 190*, *229*, *260, 261*, *291*
ssvs, 264
ssvs_prior, 266
ssvs_prior.bvarmodel, *266*, 267
ssvs_prior.bvecmodel, *266*, 268
ssvs_prior.externalforecast, *266*
ssvs_prior.externalforecast *(*create_external_forecast*)*, 135
stochvol_ksc_1998, 269, *271*
stochvol_ocsn_2007, 271
summary, *78*, *82, 83*, *263*
summary.bvarmodel, *100*, *104*, 272, *273*
summary.bvecmodel, *108*, 273, *274*
summary.expandingwindow, 274
summary.modellist, 275
sur_const_to_tvp, 275

thin, *126*, *132*
thin.bvarfolder *(*folder_steps*)*, 149
thin.bvarmodel, 276
thin.bvecmodel, 277
thin.expandingwindow, 278
thin.externalforecast *(*create_external_forecast*)*, 135
thin.forecastpool *(*pool_forecasts*)*, 212
thin.modellist, 279
thin.poolwindow *(*pool_forecasts*)*, 212
time, *137*
time_variation_test, *67*, 279
time_variation_test.bvarfolder, *280*
time_variation_test.bvarfolder *(*folder_steps*)*, 149
time_variation_test.bvarmodel, 280, *280*
time_variation_test.bvecmodel, *280*
time_variation_test.bvecmodel *(*time_variation_test.bvarmodel*)*, 280
time_variation_test.expandingwindow, *280*
time_variation_test.expandingwindow *(*time_variation_test.bvarmodel*)*, 280
time_variation_test.modellist, *280*
time_variation_test.modellist *(*time_variation_test.bvarmodel*)*, 280
transform_variables, *85*, 282
transform_variables(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *286, 287*
ts, *10*
ts.intersect, *283*

uk_macrodata, 284
us_macrodata, 284
use_expanding_window, *10*, *26*, *34*, *36*, *42, 43*, *48*, *50–52*, *73*, *104*, *138*, *261*, 285
use_expanding_window.bvarmodel, *285*, 286
use_expanding_window.bvarmodel(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *287*
use_expanding_window.bvecmodel, *285*, 287
use_expanding_window.bvecmodel(), *11*, *24, 25*, *59*, *64*, *72*, *89*, *122*, *130*, *134*, *283*, *286*
use_expanding_window.modellist, *285*, 288

vec_to_var, *10*, *14*, *19*, *40*, *56*, *71*, *120*, *148*, *173*, *190*, *196*, *231*, *236*, *259*, *261*, 289
vec_to_var.bvecmodel, *108*, *289*, 290

vec_to_var.bvecmodel(), *79*, *84*, *147*, *149*, *163*, *165*, *172*, *173*, *187*, *189*, *190*, *229*, *260*, *261*, *263*
vec_to_var.expandingwindow, 292
vec_to_var.modellist, *289*, 293

window.bvarmodel, 293
window.bvecmodel, 294
window.modellist, 295
write_bayests_tree, *151*, 295
write_to_hdf5, *75*, *95*, *97*, *104*, *128*, *134*, *142*, *143*, *152*, *191*, *192*, *241*, *295*, 297, *297*
write_to_hdf5.bvarmodel, *297*, 298
write_to_hdf5.bvecmodel, *297*, 299
write_to_hdf5.expandingwindow, *297*, 300
write_to_hdf5.forecastpool *(*pool_forecasts*)*, 212
write_to_hdf5.modellist, *297*, 301
write_to_hdf5.poolwindow *(*pool_forecasts*)*, 212