Package ‘sectorgap’

January 22, 2024

Type Package

Title Consistent Economic Trend Cycle Decomposition

Version 0.1.0

Description Determining potential output and the output gap - two inherently unobservable variables - is a major challenge for macroeconomists. 'sectorgap' features a flexible modeling and estimation framework for a multivariate Bayesian state space model identifying economic output fluctuations consistent with subsectors of the economy. The proposed model is able to capture various correlations between output and a set of aggregate as well as subsector indicators. Estimation of the latent states and parameters is achieved using a simple Gibbs sampling procedure and various plotting options facilitate the assessment of the results. For details on the methodology and an illustrative example, see Streicher (2024) <`https://www.research-collection.ethz.ch/handle/20.500.11850/653682`>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Imports stats, KFAS, zoo, ggplot2, MCMCpack, dplyr, tidyr, tempdisagg

Depends R (>= 2.10)

LazyData true

NeedsCompilation no

Author Sina Streicher [aut, cre] (<`https://orcid.org/0000-0001-7848-1842`>)

Maintainer Sina Streicher <`streicher.sina@gmail.com`>

Repository CRAN

Date/Publication 2024-01-22 17:40:02 UTC

R topics documented:

`compute_mcmc_results` *Results for sampled parameters and states*

Description

Computes estimation results for the MCMC sampling output for a specific HPDI and evaluation function (e.g. mean or median).

Usage

```
compute_mcmc_results(
  model,
  settings,
  mcmc,
  data,
  HPDIprob = NULL,
  fit = NULL,
  ...
)
```

Arguments

`model`	state space model object, returned by the function `define_ssmodel`
`settings`	list with model setting, in the format returned by the function `initialize_settings`
`mcmc`	list with draws of parameters and states (including burnin phase)
`data`	list with at least two named components: `tsm` is a multiple time series object that contains all observation series, `weights` is a named list of time series with (nominal) weights, the list names correspond to the different groups, i.e., `group1`, `group2`, `subgroup1`, if present in the model
`HPDIprob`	probability of highest posterior density interval, optional if `fit` is supplied
`fit`	(optional) an object of class `fit` (returned by the function `estimate_ssmodel` and this function).
`...`	additional arguments (in case `fit` is supplied)

Details

If `fit` is supplied, the arguments `model, settings, mcmc` will be taken from this object.

Value

An object of class `ss_fit`.

`data_ch` *Swiss data set*

Description

A dataset containing quarterly Swiss economic data, sourced on November 20, 2023.

Usage

```
data_ch
```

Format

A list object with two lists. The first list cotains all untransformed endogenous variables:

output Gross domestic product at 2020 prices, in million

vaA value added in sector A: Goods-producing industries, at 2020 prices in million

vaB value added in sector B: Service industries, at 2020 prices in million

vaC value added in sector C: Government and adjustments,at 2020 prices in million

exp1 expenditure side sector i: Total consumption, at 2020 prices in million

exp2 expenditure side sector ii: Investment, at 2020 prices in million

exp3 expenditure side sector iii: Exports, at 2020 prices in million

exp4 expenditure side sector iv: Imports, at 2020 prices in million

fteA full-time equivalent empoyment in sector A: Goods-producing industries, in thousand

fteB full-time equivalent empoyment in sector B: Service industries, in thousand

fteC full-time equivalent empoyment in government sector, in thousand

employment full-time equivalent empoyment, in thousand

urate ILO unemployment rate, in percent

inflation consumer price inflation, year on year in percent

Source

KOF Swiss Economic Institute, ETH Zurich, Switzerland define_ssmodel *State space model*

Description

Defines a state space model for the provided settings and data.

Usage

```
define_ssmodel(settings, data)
```

Arguments

settings	list with model setting, in the format returned by the function `initialize_settings`
data	list with at least two named components: `tsm` is a multiple time series object that contains all observation series, `weights` is a named list of time series with (nominal) weights, the list names correspond to the different groups, i.e., `group1`, `group2`, `subgroup1`, if present in the model

Details

`data` is preferably the output of funtion `prepare_data`.

Value

A state space model object of class `ss_model`, which consists of an object returned by the function `SSModel` of the package `KFAS` and in addition a list item called `names` which contains information on the parameters to be estimated.

Examples

```
data("data_ch")
settings <- initialize_settings()
data <- prepate_data(
  settings = settings,
  tsl = data_ch$tsl,
  tsl_n = data_ch$tsl_n
)
model <- define_ssmodel(
  settings = settings,
  data = data
)
```

estimate_ssmodel *Bayesian estimation via Gibbs sampling*

Description

Estimates the parameters and states of a multi-dimensional state space model by Bayesian methods using a Gibbs sampling procedure.

Usage

```
estimate_ssmodel(
  model,
  settings,
  data,
  prior = initialize_prior(model),
  R = 10000,
  burnin = 0.5,
  thin = 1,
  HPDIprob = 0.68,
  fit = NULL
)
```

Arguments

`model`	state space model object, returned by the function `define_ssmodel`
`settings`	list with model setting, in the format returned by the function `initialize_settings`
`data`	list with at least two named components: `tsm` is a multiple time series object that contains all observation series, `weights` is a named list of time series with (nominal) weights, the list names correspond to the different groups, i.e., `group1`, `group2`, `subgroup1`, if present in the model
`prior`	list of matrices, each list item corresponds to one endogenous variable. See `initialize_prior`
`R`	number of draws, the default is `10000`
`burnin`	share of draws as burnin period, the default is `0.5`
`thin`	thinning parameter defining how many draws are discarded. `1` means no draw is discarded, `2` means each second draw is kept, and so on
`HPDIprob`	probability of highest posterior density interval, the default is `HPDIprob = 0.68`
`fit`	already fitted object of class `ss_fit`, to continue drawing, see details

Details

If `fit` is supplied, the function will continue drawing `R` additional repetitions. In this case, all input variables except for `fit` and `R` are ignored.

Value

An object of class `ss_fit`.

Examples

```
data("data_ch")
settings <- initialize_settings()
data <- prepate_data(
  settings = settings,
  tsl = data_ch$tsl,
  tsl_n = data_ch$tsl_n
)
model <- define_ssmodel(
  settings = settings,
  data = data
)
prior <- initialize_prior(
  model = model,
  settings = settings
)

fit <- estimate_ssmodel(
  model = model,
  settings = settings,
  data = data,
  prior = prior,
  R = 100
)
```

`initialize_prior` *Prior distribution*

Description

Initializes the prior distributions.

Usage

```
initialize_prior(model, settings, lambda_d = 100, lambda_t = 100, df = 6)
```

Arguments

`model`	state space model object, returned by the function `define_ssmodel`
`settings`	list with model setting, in the format returned by the function `initialize_settings`
`lambda_d`	drift smoothing constant (default: 100)
`lambda_t`	trend smoothing constant (default: 100)
`df`	degrees of freedom for inverse gamma distributions

Details

All loadings and autoregressive parameters are assumed to be normal with mean zero and variance 1000.

All variance parameters are assumed to be inverse gamma distributed. The cycle variance has prior mean 1, and the trend variances have prior mean 1/100.

The normal distribution is parametrized via mean and variance.

the inverse gamma distribution is parametrized degrees of freedom `nu` and scale `s`.

The mean of the inverse gamma distribution is given by `beta / (alpha - 1) = beta / 2 = s`, where `s = 2 beta`, `nu = 2 alpha`.

Value

A data frame with one row per parameter and the following columns:

`variable`	name of endogneous variable of equation
`parameter_name`	name of parameter
`par1`	first parameter of specified distribution, mean for normal parameters and scale for inverse gamma parameters
`par2`	second parameter of specified distribution, variance for normal parameters and degrees of freedom for inverse gamma parameters
`ini`	initial value for Gibbs sampler, i.e. mean of distribution given `par1` and `par2`
`distribution`	name of prior distribution

Examples

```
data("data_ch")
settings <- initialize_settings()
data <- prepate_data(
  settings = settings,
  tsl = data_ch$tsl,
  tsl_n = data_ch$tsl_n
)
model <- define_ssmodel(
  settings = settings,
  data = data
)
prior <- initialize_prior(
  model = model,
  settings = settings
)
```

initialize_settings *Model settings*

Description

Initializes settings with a basic example.

Usage

```
initialize_settings(
  FUN_transform = function(x) 100 * log(x),
  FUN_transform_inv = function(x) exp(x/100),
  DFUN_transform_inv = function(x) 1/100 * exp(x/100)
)
```

Arguments

`FUN_transform`	transformation function, the default is `function(x) 100 * log(x)`
`FUN_transform_inv`	inverse transformation function, the default is `function(x) exp(x / 100)`
`DFUN_transform_inv`	derivative of inverse transformation function, the default is `function(x) 1 exp(x / 100)`, only used if non-linear constraints are present

Value

A nested list with settings for the following groups:

`agg`	settings for the aggregate variable
`group1`	settings for `group1`, all variables in this group load on the aggregate variable, unless otherwise specified
`group2`	settings for `group2`, all variables in this group load on the aggregate variable, unless otherwise specified
`subgroup1`	settings for `subgroup1`, each variable in this group loads on the respective variable in `group1`
`agggroup`	settings for a group of variables that all load on the same variable
`misc`	settings for variables that require individual settings

Each group contains at least the following list items:

`trend`	`4` is a local linear trend, `3` a local linear trend with AR(1) drift, `2` a local linear drift without shocks to trend growth, `0` implies no trend (e.g. if a variable shares a trend with another one)
`cycle`	`2` is an AR(2) cycle, `1` an AR(1) cycle, and `0` a white noise cycle, each with normal innovations

`transform`	logical indicating if the transformation function should be applied to the variable or group of variables
`variable`	variable name(s)
`variable_label`	variable label(s)
`label`	label of group

The blocks `group1, group2, subgroup1` additionally contain the following list items:

`corr`	`4` implies that trends and drifts are correlated, `2` that only dirfts are correlated, `1` that only trends are correlated, and `0` or `NA` implies no correlation. Only applicable for `group1, group2, subgroup1`
`load_name`	name of the variable that all variables in the group load (for `group1, group2`) and which is used for the aggregation
`load_lag`	lags of the of the variable that all variables in the group load (for `group1, group2`)
`constr_drift`	logical indicating if constraints for the drifts should be enforced
`constr_trends`	logical indicating if constraints for the trends should be enforced
`constr_trends_linear`	logical indicating if constraints for the trends are linear or nonlinear, the default is `FALSE` in which case the constraint is enforced on the level series, else, it is enforced on the growth rates.
`variable_neg`	variable names that are negative and thus need to be subtracted when constructing weights

The block `subgroup1` additionally contain the following list item:

`match_group1`	a character vector of the same length as `variable` indicating the matching variables in `group1`, in the same order as `variable`, `NA` indicates no match

---

`is.settings` *Settings object validity check*

Description

Checks if settings are a valid object of class `settings`.

Usage

```
is.settings(x, dfl = NULL, return.logical = FALSE)
```

Arguments

`x`	settings object
`dfl`	list of data frames, returned by function `settings_to_df`
`return.logical`	If `return.logical = FALSE` (default), an error message is printed if the object is not of class `settings`, if `return.logical = TRUE`, a logical value is returned

Value

A logical value or nothing, depending on the value of `return.logical`.

`plot.ss_fit` *Plots of results*

Description

Creates a set of time series, density, or trace plots.

Usage

```
## S3 method for class 'ss_fit'
plot(
  x,
  plot_type = "timeseries",
  estimate = "median",
  data = data,
  n_col = 3,
  n_sep = 5,
  file_path = NULL,
  title = TRUE,
  save = FALSE,
  device = "jpg",
  width = 10,
  height = 3,
  units = "in",
  highlighted_area = NULL,
  plot_start = NULL,
  plot_end = NULL,
  alpha = 0.05,
  include_burnin = FALSE,
  ...
)
```

Arguments

`x`	object of class `ss_fit`
`plot_type`	type of plots, options are `"timeseries"`, `"density"`, `"trace"`
`estimate`	character specifying the posterior estimate. Valid options are `"mean"` and `"median"`, the default is `estimate = "median"`.
`data`	list with at least two named components: `tsm` is a multiple time series object that contains all observation series, `weights` is a named list of time series with (nominal) weights, the list names correspond to the different groups, i.e., `group1`, `group2`, `subgroup1`, if present in the model

`n_col`	number of columns for grid plots
`n_sep`	increments of x axis ticks in years
`file_path`	file path for plots
`title`	boolean indicating if plots should contain titles
`save`	boolean indicating if plots should be saved, if `FALSE`, the plots will be printed instead, default is `save = FALSE` (ignored if `file_path` is provided)
`device`	character string with format used in `ggsave`
`width`	plot width in `units`, for grid plots adjusted for the number of plot columns `n_col`
`height`	plot height in `units`, for grid plots adjusted for the number of plot rows implied by `n_col`
`units`	units for plot size (`"in", "cm", "mm", or "px"`)
`highlighted_area`	data frame with two columns called `start` and `end` containing start and end date, e.g. `1990.25` and `1992.75` for 1990 Q2 until 1992 Q4 (only used if `plot_type = "timeseries"`)
`plot_start`	start of x axis in years, e.g., `1990.5` (only used if `plot_type = "timeseries"`)
`plot_end`	end of x axis in years, e.g., `2010.25` (only used if `plot_type = "timeseries"`)
`alpha`	cut off value for posterior (only used if `plot_type = "density"`)
`include_burnin`	logical indicating if burnin phase should be included (only used if `plot_type = "trace"`)
`...`	ignored

Value

nothing

`prepate_data` *Input data*

Description

Prepares the required input data, it performs the transformations to the raw data and computes the necessary weights for the constraints.

Usage

```
prepate_data(
  settings,
  tsl,
  tsl_n = NULL,
  tsl_p = NULL,
  ts_start = NULL,
  ts_end = NULL,
  extend_weights = FALSE
)
```

Arguments

`settings`	list with model setting, in the format returned by the function `initialize_settings`
`tsl`	time series list with all untransformed endogenous series
`tsl_n`	time series list with nominal level series for aggregate output `agg` and its subcomponents in `group1`, `group2`
`tsl_p`	time series list with price series for aggregate output `agg` and its subcomponents in `group1`, `group2`
`ts_start`	start date, e.g. `c(2000, 2)` or `2000.25`
`ts_end`	end date, e.g. `c(2000, 2)` or `2000.25`
`extend_weights`	logical indicating if missing weights at beginning/end of sample should be filled with the last/first available value

Details

Either `tsl_n` or `tsl_p` must be supplied.

Weights are forward/backward extended with the first/last value if the supplied time series do not cover the entire period.

Value

A list with five components:

`tsm`	multiple time series object with all (transformed) endogeneous variables
`real`	multiple time series object with real series of `agg`, `group1`, `group2`
`nominal`	multiple time series object with nominal series of `agg`, `group1`, `group2`
`prices`	multiple time series object with price series of `agg`, `group1`, `group2`
`weights_growth`	list of multiple time series objects with weights for the growth constraints, i.e., for series `group1`, `group2`, `subgroup1` if applicable
`weights_level`	list of multiple time series objects with weights for the non linear level constraints, i.e., for series `group1`, `group2`, `subgroup1` if applicable

Examples

```
data("data_ch")
settings <- initialize_settings()
data <- prepate_data(
  settings = settings,
  tsl = data_ch$tsl,
  tsl_n = data_ch$tsl_n
)
```

print.prior *Print* prior *object*

Description

Prints the model specifications of an object of class `prior`.

Usage

```
## S3 method for class 'prior'
print(x, call = TRUE, check = TRUE, ...)
```

Arguments

x	object of class `prior`
call	logical, if `TRUE`, the call will be printed
check	logical, if `TRUE`, the model class will be checked
...	ignored.

Value

No return value

print.settings *Print* settings *object*

Description

Prints the model settings.

Usage

```
## S3 method for class 'settings'
print(x, call = TRUE, check = TRUE, ...)
```

Arguments

x	object of class `settings`
call	logical, if `TRUE`, the call will be printed
check	logical, if `TRUE`, the model class will be checked
...	ignored.

Value

No return value

print.ss_fit *Print* `ss_fit` *object.*

Description

Prints the model specifications of an object of class `ss_fit`.

Usage

```
## S3 method for class 'ss_fit'
print(x, call = TRUE, check = TRUE, ...)
```

Arguments

`x`	object of class `ss_fit`
`call`	logical, if `TRUE`, the call will be printed
`check`	logical, if `TRUE`, the model class will be checked
`...`	ignored.

Value

No return value

print.ss_model *Print* `ss_model` *object*

Description

Prints the model specifications of an object of class `ss_model`.

Usage

```
## S3 method for class 'ss_model'
print(x, call = TRUE, check = TRUE, ...)
```

Arguments

`x`	object of class `ss_model`
`call`	logical, if `TRUE`, the call will be printed
`check`	logical, if `TRUE`, the model class will be checked
`...`	ignored.

Value

No return value

`recessions_ch` *Swiss recessions*

Description

Recession periods in Switzerland since 1990.

Usage

```
recessions_ch
```

Format

A data frame with two columns:

start start date of recession, in quarters

end end date of recession, in quarters

`recessions_us` *US recessions*

Description

Recession periods in the United States since 1960.

Usage

```
recessions_us
```

Format

A data frame with two columns:

start start date of recession, in quarters

end end date of recession, in quarters

Source

National Bureau of Economic Research (NBER)

transform_results *Format results*

Description

Formats the output series into a tibble in long format and computes contribution series.

Usage

```
transform_results(
  fit,
  data,
  settings,
  estimate = "median",
  HPDIprob = 0.68,
  transformed = TRUE
)
```

Arguments

`fit`	fitted object
`data`	list with at least two named components: `prices` is a multiple time series object that contains price indices for all relevant series, `weights`, is a named list of time series with (nominal) weights, the list names correspond to the different groups, i.e., `group1`, `group2`, `subgroup1`, if present in the model
`settings`	list with model setting, in the format returned by the function `initialize_settings`
`estimate`	character specifying the posterior estimate. Valid options are `"mean"` and `"median"`, the default is `estimate = "median"`.
`HPDIprob`	probability of highest posterior density interval, the default is `HPDIprob = 0.68`
`transformed`	boolean indicating if the transformed series should be used.

Details

`data` is preferably the output of funtion `prepare_data`.

Value

A data frame with results in long format.

Examples

```
data("data_ch")
settings <- initialize_settings()
data <- prepate_data(
  settings = settings,
  tsl = data_ch$tsl,
  tsl_n = data_ch$tsl_n
)
model <- define_ssmodel(
  settings = settings,
  data = data
)
prior <- initialize_prior(
  model = model,
  settings = settings
)

fit <- estimate_ssmodel(
  model = model,
  settings = settings,
  data = data,
  prior = prior,
  R = 100
)
df <- transform_results(
  fit = fit,
  data = data,
  estimate = "median"
)
```

Index