Package: TwoStepSDFM (via r-universe)

June 1, 2026

Title Estimate a Sparse Mixed Frequency Gaussian Factor Model Using a Two-Step Procedure

Version 0.3.0

Maintainer Domenic Franjic <franjic@uni-hohenheim.de>

Description Estimate a sparse Gaussian state-space model with mixed frequency data via sparse principal components analysis and the Kalman filter and smoother. For more details see Franjic and Schweikert (2024) <doi:10.2139/ssrn.4733872>.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports Rcpp (>= 1.0.8), zoo, xts, lubridate, ggplot2, patchwork, doSNOW, doParallel, foreach, parallel, Rdpack, grDevices, withr

RdMacros Rdpack

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.0)

LazyData TRUE

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

Date/Publication 2026-06-01 08:23:31 UTC

RemoteUrl https://github.com/sisanchopancho/twostepsdfm

RemoteRef HEAD

RemoteSha 412f4a61e4a2a35d0060339747fad49ec5f34d5a

Contents

`crossVal` *Cross-validate SDFM Hyper-Parameters*

Description

This function uses time series cross-validation (Hyndman and Athanasopoulos 2018) in combination with random hyper-parameter search (Bergstra and Bengio 2012) to validate the hyper-parameters of a sparse dynamic factor model as described in Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.

Usage

```
crossVal(
  data,
  variable_of_interest,
  fcast_horizon,
  delay,
  frequency,
  no_of_factors,
  seed,
  min_ridge_penalty,
  max_ridge_penalty,
  cv_repetitions,
  cv_size,
  lasso_penalty_type,
  min_max_penalty,
  max_factor_lag_order = 10,
  lag_estim_criterion = "BIC",
  decorr_errors = TRUE,
  max_iterations = 1000,
  weights = NULL,
  comp_null = 1e-15,
  spca_conv_crit = 1e-04,
  parallel = FALSE,
  no_of_cores = 1,
  max_ar_lag_order = 5,
  max_predictor_lag_order = 5,
  jitter = 1e-08,
  svd_method = "precise",
  verbose = TRUE
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at mixed frequencies (quarterly and monthly).
`variable_of_interest`	Integer indicating the index of the target variables.
`fcast_horizon`	Integer value indicating the target forecasting horizon.
`delay`	Integer vector of variable delays, measured as the number of months since the latest available observation.
`frequency`	Integer vector of frequencies of the variables in the data set (currently supported: `12` for monthly and `4` for quarterly data).
`no_of_factors`	Integer number of factors.
`seed`	32-bit unsigned integer seed for all random processes inside the function.
`min_ridge_penalty`	Numeric lower bound for the sampled ridge penalty coefficient candidates.

`max_ridge_penalty`
: Numeric upper bound for the sampled ridge penalty coefficient candidates.

`cv_repetitions`
: Integer number of `fcast_horizon`-step-ahead predictions computed for each candidate set.

`cv_size`
: Integer number of candidate sets.

`lasso_penalty_type`
: Character indicating the lasso penalty type. If set to `"selected"`, the ℓ_1-size constraint will be returned as number of non-zero elements of each column of the loading matrix. If set to `"penalty"`, the lasso size constraint will be returned. If set to `"steps"`, the number of LARS-EN steps will be returned.

`min_max_penalty`
: Vector of size two, where the first element indicates the lower and the second element indicates the upper bound of the lasso penalty equivalent. If `lasso_penalty_type` is set to `"selected"` or `"steps"`, both elements must be strictly positive integers.

`max_factor_lag_order`
: Integer maximum order of the VAR process in the transition equation.

`lag_estim_criterion`
: Information criterion used for the estimation of the factor VAR order (`"BIC"` (default), `"AIC"`, `"HIC"`).

`decorr_errors`
: Logical, whether or not the errors should be decorrelated.

`max_iterations`
: Integer maximum number of iterations of the SPCA algorithm.

`weights`
: Numeric vector, weights for each variable weighing the ℓ_1 size constraint.

`comp_null`
: Numeric computational zero.

`spca_conv_crit`
: Numeric conversion criterion for the SPCA algorithm.

`parallel`
: Logical, whether or not to run the cross-validation loop in parallel.

`no_of_cores`
: Integer number of cores to use when run in parallel.

`max_ar_lag_order`
: Integer maximum number of lags of the target variable included in the final ARDL prediction routine.

`max_predictor_lag_order`
: Integer maximum number of lags of the predictors included in the final ARDL prediction routine.

`jitter`
: Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance covariance matrix of the measurement errors.

`svd_method`
: Either `"fast"` or `"precise"`. Option `"fast"` uses Eigen's BDCSVD divide and conquer method for the computation of the singular values. Option `"precise"` (default) implements the slower, but numerically more stable JacobiSVD method.

`verbose`
: Logical, whether to print some progress tracking output to the console.

Details

`fcast_horizon` should be set to the target prediction horizon, as hyper-parameters can differ substantially between different horizons. For nowcasting, use `fcast_horizon = 0`. For backcasting, `fcast_horizon` can be set to a negative number indicating the step-back backcasting horizon.

Internally, candidates of the hyper-parameters are drawn randomly. However, a regular dense DFM will always be considered by default. The ridge penalty is drawn as $\exp(u)$, where u is uniformly distributed between `min_ridge_penalty` and `max_ridge_penalty`. If `lasso_penalty_type = "selected"`, the lasso penalty is drawn as a random vector $\boldsymbol{v}$, where each entry is uniformly distributed. If `lasso_penalty_type = "steps"`, the lasso penalty is drawn as a random value v that is uniformly distributed. If `lasso_penalty_type = "penalty"`, the lasso penalty is drawn as a random vector $\exp(\boldsymbol{v})$, where each entry of $\boldsymbol{v}$ is uniformly distributed. In all three cases, the upper and lower bounds of the uniform distributions governing the lasso penalties are given by the first and second entry of `min_max_penalty`, respectively.

For medium to large data sets in combination with a medium to large `cv_size`, it can be beneficial to set `parallel = TRUE`. This will enable parallelisation via the doParallel, doSNOW, foreach, and parallel packages in R. In this case, `no_of_cores` should be set to the number of physical cores of the user's machine. It is not advisable to use the number of logical cores, as this can considerably deteriorate performance.

This function serves as a direct wrapper to `nowcast`. For more information on the additional function parameters, see the corresponding help page.

Value

An object of class `SDFMcrossVal` with main components:

`CV` A list with components `` `CV Results` `` (matrix of all cross-validation errors and corresponding hyper-parameter values) and `` `Min. CV` `` (row of `CV Results` with the minimum cross-validation error).

`BIC` A list with components `BIC Results` (matrix of all BIC values and corresponding hyper-parameter values) and `Min. BIC` (row of `BIC Results` with the minimum BIC).

Author(s)

Domenic Franjic

References

Bergstra J, Bengio Y (2012). "Random search for hyper-parameter optimization." *Journal of Machine Learning Research*, **13**(2).

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: principles and practice*, 3 edition. OTexts Melbourne.

Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.

See Also

sparsePCA: Routine for fitting estimating a sparse factor loading matrix.

kalmanFilterSmoother: Routine for filtering and smoothing latent factors.

twoStepSDFM: Two-step estimation routine for a sparse dynamic factor model.

twoStepDenseDFM: Two-step estimation routine for a dense dynamic factor model.

Examples

```
data(mixed_freq_factor_model)
no_of_vars <- dim(mixed_freq_factor_model$data)[2]
no_of_factors <- dim(mixed_freq_factor_model$factors)[2]
cv_results <- crossVal(data = mixed_freq_factor_model$data, variable_of_interest = 1,
                       fcast_horizon = 0, delay = mixed_freq_factor_model$delay,
                       frequency = mixed_freq_factor_model$frequency,
                       no_of_factors = no_of_factors, seed = 25032026,
                       min_ridge_penalty = 1e-5, max_ridge_penalty = 10,
                      cv_repetitions = 1, cv_size = 50, lasso_penalty_type = "selected",
                       min_max_penalty = c(5, 45), verbose = FALSE)
print(cv_results)
cv_plots <- plot(cv_results)
cv_plots$`CV Results`
cv_plots$`BIC Results`
```

---

factor_model *Example factor model dataset*

Description

This is a simulated factor model dataset for demonstration and testing with TwoStepSDFM.

Usage

```
factor_model
```

Format

A SimulData containing the following elements:

data If starting_date is provided, a zoo object, else, a (no_of_vars × no_of_obs) numeric matrix holding the simulated data.

factors If starting_date is provided, a zoo object, else a (no_of_factors × no_of_obs) numeric matrix holding the simulated latent factors.

trans_var_coeff Numeric (no_of_factors × (no_of_factors * factor_lag_order)) factor VAR coefficient matrix.

loading_matrix Numeric factor loading matrix.

meas_error If `starting_date` is provided, a `zoo` object, else a (`no_of_vars` × `no_of_obs`) numeric matrix holding the fundamental measurement errors.

meas_error_var_cov Numeric measurement error variance-covariance matrix.

trans_error_var_cov Numeric transition error variance-covariance matrix.

frequency Integer vector of variable frequencies.

delay Integer vector of variable delays, measured as the number of months since the latest available observation.

Source

Generated via `simFM()`. For details see `factor_model$call`.

`imputeMonthlyData` *Impute monthly data.*

Description

Impute missing data in monthly datasets using matrix imputation via a DFM fit,

Usage

```
imputeMonthlyData(
  data,
  delay,
  no_of_factors,
  max_factor_lag_order = 10,
  lag_estim_criterion = "BIC",
  decorr_errors = TRUE,
  comp_null = 1e-15,
  parallel = FALSE,
  fcast_horizon = 0,
  jitter = 1e-08
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`delay`	Integer vector of variable delays.
`no_of_factors`	Integer number of factors.
`max_factor_lag_order`	Integer maximum order of the VAR process in the transition equation.
`lag_estim_criterion`	Information criterion used for the estimation of the factor VAR order (`"BIC"` (default), `"AIC"`, `"HIC"`).

decorr_errors	Logical, whether or not the errors should be decorrelated.
comp_null	Numeric computational zero.
parallel	Logical, whether or not to use Eigen's internal parallel matrix operations.
fcast_horizon	Integer number of additional Filter predictions into the future.
jitter	Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance covariance matrix of the measurement errors.

Details

The function performs a two-step estimation procedure for dense dynamic factor models as described in Giannone D, Reichlin L, Small D (2008). "Nowcasting: The real-time informational content of macroeconomic data." *Journal of Monetary Economics*, **55**(4), 665-676. ISSN 0304-3932. doi:10.1016/j.jmoneco.2008.05.010. and Doz C, Giannone D, Reichlin L (2011). "A two-step estimator for large approximate dynamic factor models based on Kalman filtering." *Journal of Econometrics*, **164**(1), 188-205. ISSN 0304-4076. doi:10.1016/j.jeconom.2011.02.012.. For more details, see `twoStepDenseDFM`. The resulting estimates of the loading matrix and factors are then used to predict any missing observations outside the ragged edges.

Value

A named list containing the following objects:

imputed_data Imputed dataset with class inherited from `data`.

predicted_data In-sample model predictions with class inherited from `data`.

dfm_fit `SDFMFit` object of the DFM fit.

Author(s)

Domenic Franjic

References

Giannone D, Reichlin L, Small D (2008). "Nowcasting: The real-time informational content of macroeconomic data." *Journal of Monetary Economics*, **55**(4), 665-676. ISSN 0304-3932. doi:10.1016/j.jmoneco.2008.05.010.

Doz C, Giannone D, Reichlin L (2011). "A two-step estimator for large approximate dynamic factor models based on Kalman filtering." *Journal of Econometrics*, **164**(1), 188-205. ISSN 0304-4076. doi:10.1016/j.jeconom.2011.02.012.

Examples

```
data(factor_model)
no_of_vars <- dim(factor_model$data)[2]
no_of_factors <- dim(factor_model$factors)[2]
factor_model$data[1:3, ] <- NA
factor_model$data
imputed_data <- imputeMonthlyData(data = factor_model$data, delay = factor_model$delay,
```

```
                                    no_of_factors = no_of_factors)
imputed_data$imputed_data
```

`kalmanFilterSmoother` *Univariate Representation of the Multivariate Kalman Filter and Smoother*

Description

Filter and smooth the latent states/factors of a linear Gaussian state-space model, with measurement equation

$$\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{f}_t + \boldsymbol{\xi}_t, \quad \boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\xi),$$

and transition equation

$$\boldsymbol{f}_t = \sum_{p=0}^{P} \boldsymbol{\Phi}_p \boldsymbol{f}_{t-p} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_f).$$

for t = 1, . . . , T. For filtering and smoothing, the univariate representation of the multivariate Kalman Filter and Smoother is implemented according to Koopman SJ, Durbin J (2000). "Fast filtering and smoothing for multivariate state space models." *Journal of Time Series Analysis*, **21**(3), 281–296..

Usage

```
kalmanFilterSmoother(
  data,
  delay,
  no_of_factors,
  loading_matrix,
  meas_error_var_cov,
  trans_error_var_cov,
  trans_var_coeff,
  factor_lag_order,
  fcast_horizon = 0,
  decorr_errors = TRUE,
  comp_null = 1e-15,
  parallel = FALSE,
  jitter = 1e-08
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`delay`	Integer vector of variable delays.
`no_of_factors`	Integer number of factors.
`loading_matrix`	Numeric (no_of_vars × no_of_factors) loading matrix.

`meas_error_var_cov`
: Numeric (no_of_factors × no_of_factors) variance-covariance matrix of the measurement errors.

`trans_error_var_cov`
: Numeric (no_of_vars × no_of_vars) variance-covariance matrix of the transition errors.

`trans_var_coeff`
: Either a list of length max_factor_lag_order with each entry a numeric (no_of_factors × no_of_factors) VAR coefficient matrix or a matrix of dimensions (no_of_factors x(no_of_factors

  - max_factor_lag_order)) holding the VAR coefficients of the factor VAR process in each (no_of_factors × no_of_factors) block.

`factor_lag_order`
: Integer order of the VAR process in the state equation.

`fcast_horizon`
: Integer number of additional Filter predictions into the future.

`decorr_errors`
: Logical, whether or not the errors should be decorrelated (should be `TRUE` if `meas_error_var_cov` is not diagonal).

`comp_null`
: Computational zero.

`parallel`
: Logical, whether or not to use Eigen's internal parallel matrix operations.

`jitter`
: Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance-covariance matrix of the measurement errors.

Details

To implement the univariate representation of the Kalman Filter and Smoother, the measurement error term has to be cross-sectionally uncorrelated. If `meas_error_var_cov` is not diagonal, one should set `decorr_errors = TRUE` so that the data can be decorrelated internally prior to filtering and smoothing.

When decorrelating, the function first adds `jitter` to the diagonal elements of `meas_error_var_cov` and then tries to compute the Cholesky factor via Eigen's standard LLT decomposition (Guennebaud et al. 2010). If the initial decorrelation fails, it silently switches to Eigen's more robust, but slower, LDLT decomposition with pivoting (Guennebaud et al. 2010). If this also fails, it is likely that `meas_error_var_cov` is not well-behaved. The analysis should be repeated with a larger `jitter` or a more robust variance-covariance matrix (estimator). The success of the internal Cholesky decomposition is reported by `llt_success_code`.

Value

An object of class `KFSFit` with components:

data Original data matrix.

smoothed_factors Object containing the smoothed factor estimates. The object inherits its class from `data`: If `data` is provided as `zoo`, `smoothed_factors` will be a `zoo` object. If `data` is provided as `matrix`, `smoothed_factors` will be a (`no_of_factors` × `no_of_obs`) matrix.

smoothed_state_variance (no_of_factors × (no_of_factors * no_of_obs)) matrix, where each (no_of_factors × no_of_factors) block represents the smoother uncertainty at time point t

factor_var_lag_order Integer order of the VAR process in the state equation.

llt_success_code Integer indicating the status of the Cholesky factorization: 0 = LLT succeeded, -1 = LLT failed but LDLT succeeded, -2 = both failed and errors are treated as uncorrelated.

Author(s)

Domenic Franjic

References

Koopman SJ, Durbin J (2000). "Fast filtering and smoothing for multivariate state space models." *Journal of Time Series Analysis*, **21**(3), 281–296.

Guennebaud G, Jacob B, others (2010). "Eigen." https://libeigen.gitlab.io.

Examples

```
data(factor_model)
no_of_factors <- dim(factor_model$factors)[2]
factor_lag_order <- dim(factor_model$trans_var_coeff)[2] / no_of_factors
filter_fit <- kalmanFilterSmoother(data = factor_model$data, delay = factor_model$delay,
                                   no_of_factors = no_of_factors,
                                   loading_matrix = factor_model$loading_matrix,
                                   meas_error_var_cov = factor_model$meas_error_var_cov,
                                 trans_error_var_cov = factor_model$trans_error_var_cov,
                                   trans_var_coeff = factor_model$trans_var_coeff,
                                   factor_lag_order = factor_lag_order,
                                   fcast_horizon = 5, decorr_errors = TRUE,
                                   comp_null = 1e-15, parallel = FALSE,  jitter = 1e-8)
print(filter_fit)
filter_plots <- plot(filter_fit)
filter_plots$`Factor Time Series Plots`
```

---

mixed_freq_factor_model

Mixed-frequency factor model dataset

Description

This dataset contains simulated mixed-frequency factor model data for examples in TwoStepSDFM.

Usage

```
mixed_freq_factor_model
```

Format

A SimulData containing the following elements:

data If starting_date is provided, a zoo object, else, a (no_of_vars × no_of_obs) numeric matrix holding the simulated data.

factors If starting_date is provided, a zoo object, else a (no_of_factors × no_of_obs) numeric matrix holding the simulated latent factors.

trans_var_coeff Numeric (no_of_factors × (no_of_factors * factor_lag_order)) factor VAR coefficient matrix.

loading_matrix Numeric factor loading matrix.

meas_error If starting_date is provided, a zoo object, else a (no_of_vars × no_of_obs) numeric matrix holding the fundamental measurement errors.

meas_error_var_cov Numeric measurement error variance-covariance matrix.

trans_error_var_cov Numeric transition error variance-covariance matrix.

frequency Integer vector of variable frequencies.

delay Integer vector of variable delays, measured as the number of months since the latest available observation.

Source

Generated via Generated via simFM(). For details see mixed_freq_factor_model$call.

---

noOfFactors *Estimate the number of Factors*

Description

Estimate the number of factors of a linear Gaussian latent factor model using the eigenvalue slope test of Onatski A (2009). "Testing hypotheses about the number of factors in large factor models." *Econometrica*, **77**(5), 1447–1479. and the infromation criterion based approach of Bai J, Ng S (2002). "Determining the number of factors in approximate factor models." *Econometrica*, **70**(1), 191–221..

Usage

```
noOfFactors(
  data,
  min_no_factors = 1,
  max_no_factors = 7,
  confidence_threshold = 0.05
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`min_no_factors`	Integer minimum number of factors to be tested.
`max_no_factors`	Integer maximum number of factors to be tested (should be at most min_no_factors + 17).
`confidence_threshold`	Numeric threshold value to stop the testing procedure.

Details

The (Onatski 2009) procedure splits the data matrix along the time dimension into two equally sized (`no_of_vars` × `cut_off`) sub-matrices $\boldsymbol{X}_{1/2}$ and $\boldsymbol{X}_{2/2}$. It then proceeds to build $\tilde{\boldsymbol{X}} := \boldsymbol{X}_{1/2} + i\boldsymbol{X}_{2/2}$, where $i = \sqrt{-1}$. We then compute the eigenvalues of the Gram matrix $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^{\dagger}$, where $\tilde{\boldsymbol{X}}^{\dagger}$ represents the adjoint. Finally, a test based on the computed eigenvalues is performed. This test is an iterative testing procedure, starting by testing the null that the true number of factors is `min_no_factors`. If the test is rejected by comparison of the p-value against `confidence_threshold`, we test whether the true number of factors is `min_no_factors` + 1 until we can no longer reject at `confidence_threshold` or `max_no_factors` is reached.

As the distribution of the eigenvalues under the null is nonstandard (Onatski 2009), simulated critical values are used. They are retrieved from Onatski A (2009). "Alexey Onatskiy – An old link to some of my papers." Formerly available at https://www.econ.cam.ac.uk/people/faculty/ao319/papers; Last accessed.. As the range of the simulated critical values is limited, the minimum and maximum number of potential factors is limited such that `max_no_factors` should be no more than `min_no_factors` + 17. However, it is recommended to operate well below this maximum as the test size decreases with `max_no_factors` - `min_no_factors`.

The (Bai and Ng 2002) information criterion determines the number of factors by minimising a BIC. Here, three different penalty terms are provided. It is up to the user to determine the most appropriate for the problem at hand. In general, however, the second information criterion is used.

Value

An object of class `NoOfFactorsFit` with components:

no_of_factors Integer estimated number of factors.

p_value Numeric p-value of the final test.

confidence_threshold Numeric significance level used.

statistic Numeric test statistic value of the last test.

eigen_values Numeric vector of eigenvectors of the complex data Gram matrix.

Author(s)

Domenic Franjic

References

Onatski A (2009). "Testing hypotheses about the number of factors in large factor models." *Econometrica*, **77**(5), 1447–1479.

Onatski A (2009). "Alexey Onatskiy – An old link to some of my papers." Formerly available at https://www.econ.cam.ac.uk/people/faculty/ao319/papers; Last accessed.

Examples

```
data(factor_model)
no_of_factors_estim <- noOfFactors(data = factor_model$data, min_no_factors = 1,
                                   max_no_factors = 5, confidence_threshold = 0.05)
print(no_of_factors_estim)
factor_estim_plots <- plot(no_of_factors_estim)
factor_estim_plots$`Eigen Value Plot Test Procedure`
factor_estim_plots$`IC plot for IC1`
factor_estim_plots$`IC plot for IC2`
factor_estim_plots$`IC plot for IC3`
```

`nowcast` *Predict Mixed-Frequency Data via Dynamic Factor Models*

Description

Backcast, nowcast, and forecast quarterly target variables via a sparse/dense DFM using additional monthly data with ragged edges. Forecasts are produced using all quarterly targets and a quarterly representation of latent monthly factors (Mariano and Murasawa 2003). Final predictions are computed via equally weighted forecast averaging of ARDL models (Marcellino and Schumacher 2010) for each of the targets and quarterfied factors.

Usage

```
nowcast(
  data,
  variables_of_interest,
  max_fcast_horizon,
  delay,
  selected,
  frequency,
  no_of_factors,
  sparse = TRUE,
  max_factor_lag_order = 10,
  lag_estim_criterion = "BIC",
  decorr_errors = TRUE,
  ridge_penalty = 1e-06,
  lasso_penalty = NULL,
  max_iterations = 1000,
```

```
  max_no_steps = NULL,
  weights = NULL,
  comp_null = 1e-15,
  spca_conv_crit = 1e-04,
  parallel = FALSE,
  max_ar_lag_order = 5,
  max_predictor_lag_order = 5,
  jitter = 1e-08,
  svd_method = "precise"
)
```

Arguments

`data`	Numeric (no_of_vars $\times$ no_of_obs) matrix of data or zoo/xts object sampled at mixed frequencies (quarterly and monthly).
`variables_of_interest`	Integer vector indicating the index of all target variables.
`max_fcast_horizon`	Maximum forecasting horizon of all targets.
`delay`	Integer vector of variable delays, measured as the number of months since the latest available observation.
`selected`	Integer vector of the number of selected variables for each factor.
`frequency`	Integer vector of frequencies of the variables in the data set (currently supported: `12` for monthly and `4` for quarterly data).
`no_of_factors`	Integer number of factors.
`sparse`	Logical, if `TRUE` (default) a sparse DFM is used to estimate the model parameters and latent factors (see `twoStepSDFM`). Else, a dense DFM is used (see `twoStepDenseDFM`).
`max_factor_lag_order`	Integer maximum order of the VAR process in the transition equation.
`lag_estim_criterion`	Information criterion used for the estimation of the factor VAR order (`"BIC"` (default), `"AIC"`, `"HIC"`).
`decorr_errors`	Logical, whether or not the errors should be decorrelated.
`ridge_penalty`	Numeric ridge penalty.
`lasso_penalty`	Numeric vector, lasso penalties for each factor (set to NULL to disable as stopping criterion).
`max_iterations`	Integer maximum number of iterations of the SPCA algorithm.
`max_no_steps`	Integer number of LARS steps (set to NULL to disable as stopping criterion).
`weights`	Numeric vector, weights for each variable weighing the ℓ_1 size constraint.
`comp_null`	Numeric computational zero.
`spca_conv_crit`	Numeric conversion criterion for the SPCA algorithm.
`parallel`	Logical, whether or not to use Eigen's internal parallel matrix operations.

`max_ar_lag_order`
: Integer maximum number of lags of the target variable included in the final ARDL prediction routine.

`max_predictor_lag_order`
: Integer maximum number of lags of the predictors included in the final ARDL prediction routine.

`jitter`
: Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance covariance matrix of the measurement errors.

`svd_method`
: Either `"fast"` or `"precise"`. Option `"fast"` uses Eigen's BDCSVD divide and conquer method for the computation of the singular values. Option `"precise"` (default) implements the slower, but numerically more stable JacobiSVD method.

Details

This function serves as a prediction wrapper for the `twoStepDenseDFM` and `twoStepSDFM` functions. `data` should be a mixed-frequency data set. Currently, only monthly and quarterly data are supported. With respect to the quarterly data, the function expects the realization of the quarterly observations to occur in the last month of the quarter. Indicate quarterly and monthly variables via `frequency` by setting the corresponding element of `frequency` to `4` for quarterly and to `12` for monthly data.

This function is only able to compute predictions for quarterly variables. To impute the ragged edges of the monthly observations, and potentially compute additional predictions for the monthly variables, call `predict` on the `SDFMFit` object returned by `twoStepDenseDFM` / `twoStepSDFM` (see `predict.SDFMFit`).

`max_fcast_horizon` sets the maximum number of forecasts predicted starting from the final observation of the data set. For each target, the number of backcasts and whether or not a nowcast should be computed is determined internally. This is done in such a way that every missing quarterly observation of the targets is predicted.

`max_ar_lag_order` governs the maximum number of lags of the current target used to predict said target in each ARDL model. `max_predictor_lag_order` governs the maximum number of lags of each additional quarterly predictor, including other potential targets and the aggregated factors, used to predict any given target in each ARDL model. The actual number of lags is internally estimated using the BIC. Setting `max_ar_lag_order = 0` disables the use of target lags in its own prediction function.

`sparse` toggles between a sparse DFM and a dense DFM. If `sparse = FALSE`, all SPCA stopping criteria and other parameters passed to the sparse estimation routine are ignored (for details on these parameters see `twoStepDenseDFM`). Parameters governing the Kalman Filter and Smoother are passed directly to `twoStepDenseDFM` / `twoStepSDFM`. For details see the corresponding help pages.

Value

The `nowcast` function returns named list containing the following objects:

Forecasts Numeric matrix of the target variables and their respective backcasts, nowcasts, and/or forecasts.

SDFM Fit An `SDFMFit` object holding the estimates of the model parameters and the latent factors (see `twoStepSDFM` or `twoStepDenseDFM`).

Author(s)

Domenic Franjic

References

Mariano RS, 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.

Marcellino M, Schumacher C (2010). "Factor MIDAS for nowcasting and forecasting with ragged-edge data: A model comparison for German GDP." *Oxford Bulletin of Economics and Statistics*, **72**(4), 518–550.

Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.

See Also

`sparsePCA`: Routine for fitting estimating a sparse factor loading matrix.

`kalmanFilterSmoother`: Routine for filtering and smoothing latent factors.

`twoStepSDFM`: Two-step estimation routine for a sparse dynamic factor model.

`twoStepDenseDFM`: Two-step estimation routine for a dense dynamic factor model.

Examples

```
data(mixed_freq_factor_model)
no_of_vars <- dim(mixed_freq_factor_model$data)[2]
no_of_factors <- dim(mixed_freq_factor_model$factors)[2]
sparse_nowcast <- nowcast(data = mixed_freq_factor_model$data, variables_of_interest = c(1, 2),
                          max_fcast_horizon = 4, delay = mixed_freq_factor_model$delay,
                          selected = rep(floor(0.5 * no_of_vars), no_of_factors),
                          frequency = mixed_freq_factor_model$frequency,
                          no_of_factors = no_of_factors, sparse = TRUE)
print(sparse_nowcast)
dense_nowcast <- nowcast(data = mixed_freq_factor_model$data, variables_of_interest = c(1, 2),
                         max_fcast_horizon = 4, delay = mixed_freq_factor_model$delay,
                         selected = NULL, frequency = mixed_freq_factor_model$frequency,
                         no_of_factors = no_of_factors, sparse = FALSE)
sparse_plots <- plot(sparse_nowcast)
sparse_plots$`Single Pred. Fcast Density Plots Series 1`
```

plot.KFSFit *Generic plotting function for KFSFit S3 objects*

Description

Create diagnostic plots for a KFSFit object.

Usage

```
## S3 method for class 'KFSFit'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

x	KFSFit object.
axis_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., text = element_text(size = axis_text_size)).
legend_title_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., legend.title = element_text(size = legend_title_text_size)).
...	Additional parameters for the plotting functions.

Value

A named list of patchwork/ggplot objects:

Factor Time Series Plots patchwork/ggplot object graphing the estimated factors over time with 95% confidence bands based on the smoother uncertainty of the Kalman Filter and Smoother.

Author(s)

Domenic Franjic

plot.NoOfFactorsFit *Generic plotting function for NoOfFactorsFit S3 objects*

Description

Create diagnostic plots for an NoOfFactorsFit object,

Usage

```
## S3 method for class 'NoOfFactorsFit'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

x	NoOfFactorsFit object.
axis_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., text = element_text(size = axis_text_size)).
legend_title_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., legend.title = element_text(size = legend_title_text_size)).
...	Additional parameters for the plotting functions.

Value

A named list of plot objects:

Eigen Value Plot ggplot object showing a bar plot of the eigenvalues of the complex data Gram matrix.

Author(s)

Domenic Franjic

---

plot.SDFMcrossVal *Generic plotting function for SDFMcrossVal S3 objects*

Description

Generic plotting function for SDFMcrossVal S3 objects

Usage

```
## S3 method for class 'SDFMcrossVal'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

x	SDFMcrossVal object.
axis_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., text = element_text(size = axis_text_size)).
legend_title_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., legend.title = element_text(size = legend_title_text_size)).
...	Additional parameters for the plotting functions.

Value

A named list of ggplot objects:

`CV Results` ggplot object of the cross-validation error against the log Ridge penalty. The overall sparsity level of the loading matrix induced by the lasso penalty is indicated by point shapes and colours.

`BIC Results` ggplot object of the BIC against the log Ridge penalty. The overall sparsity level of the loading matrix induced by the lasso penalty is indicated by point shapes and colours.

Author(s)

Domenic Franjic

---

plot.SDFMFit *Generic plotting function for SDFMFit S3 objects*

Description

Generic plotting function for SDFMFit S3 objects

Usage

```
## S3 method for class 'SDFMFit'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

`x`	`SDFMFit` object.
`axis_text_size`	Numeric size of x- and y-axis labels. Passed to ggplot2 `theme(..., text = element_text(size = axis_text_size))`.
`legend_title_text_size`	Numeric size of x- and y-axis labels. Passed to ggplot2 `theme(..., legend.title = element_text(size = legend_title_text_size))`.
`...`	Additional parameters for the plotting functions.

Value

A named list of plot objects:

`Factor Time Series Plots` patchwork/ggplot object graphing the estimated factors over time with 95% confidence bands based on the smoother uncertainty of the Kalman Filter and Smoother.

`Loading Matrix Heatmap` ggplot object showing a heatmap of the estimated factor loadings. Zeros are highlighted in black.

`Meas. Error Var.-Cov. Matrix Heatmap` ggplot object showing a heatmap of the measurement error variance-covariance matrix.

`Eigenvalue Plot` ggplot object showing a bar plot of the eigenvalues of the measurement error variance–covariance matrix.

Author(s)

Domenic Franjic

`plot.SDFMnowcast` *Generic plotting function for SDFMnowcast S3 objects*

Description

Generic plotting function for SDFMnowcast S3 objects

Usage

```
## S3 method for class 'SDFMnowcast'
plot(x, axis_text_size = 20, ...)
```

Arguments

`x`	`SDFMnowcast` object.
`axis_text_size`	Numeric size of x- and y-axis labels. Prased to ggplot2 `theme(..., text = element_text(size = axis_text_size))`.
`...`	Additional parameters for the plotting functions.

Value

A named list storing of `ggplot` objects:

`Single Pred. Fcast Density Plots x patchwork` / `ggplot` objects graphing the distribution of forecasts generated by the predictors for each prediction (backcasts, nowcasts, forecasts) for each target, respectively. Altogether, there will be as many such objects as there are targets, with `x` replaced by the column name of the target.

Author(s)

Domenic Franjic

plot.SimulData *Generic plotting function for SimulData S3 objects*

Description

Create diagnostic plots for an SimulData object.

Usage

```
## S3 method for class 'SimulData'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

x	SimulData object.
axis_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., text = element_text(size = axis_text_size)).
legend_title_text_size	Numeric size of x- and y-axis labels. Prased to ggplot2 theme(..., legend.title = element_text(size = legend_title_text_size)).
...	Additional parameters for the plotting functions.

Value

A named list of plot objects:

Factor Time Series Plots patchwork/ggplot object showing the simulated factors over time.

Loading Matrix Heatmap ggplot object showing a heatmap of the simulated factor loadings. Zeros are highlighted in black.

Meas. Error Var.-Cov. Matrix Heatmap ggplot object showing a heatmap of the measurement error variance-covariance matrix.

Meas. Error Var.-Cov. Eigenvalue Plot ggplotobject showing a bar plot of the eigenvalues of the measurement error variance-covariance matrix.

Data Var.-Cov. Matrix Heatmap ggplot object showing a heatmap of the data variance-covariance matrix.

Data Var.-Cov. Eigenvalue Plot ggplot object showing a bar plot of the eigenvalues of the data variance-covariance matrix.

`plot.SPCAFit` *Generic plotting function for SPCAFit S3 objects*

Description

Create diagnostic plots for an `SPCAFit` object.

Usage

```
## S3 method for class 'SPCAFit'
plot(x, axis_text_size = 20, legend_title_text_size = 20, ...)
```

Arguments

`x`	`SPCAFit` object.
`axis_text_size`	Numeric size of x- and y-axis labels. Passed to ggplot2 `theme(..., text = element_text(size = axis_text_size))`.
`legend_title_text_size`	Numeric size of x- and y-axis labels. Passed to ggplot2 `theme(..., legend.title = element_text(size = legend_title_text_size))`.
`...`	Additional parameters for the plotting functions.

Value

A named list of plot objects:

`Factor Time Series Plots` `patchwork`/`ggplot` object showing the estimated factors over time.

`Loading Matrix Heatmap` `ggplot` object showing a heatmap of the estimated factor loadings. Zeros are highlighted in black.

`Meas. Error Var.-Cov. Matrix Heatmap` `ggplot` object showing a heatmap of the measurement error variance–covariance matrix.

`Eigenvalue Plot` `ggplot` object showing a bar plot of the eigenvalues of the measurement error variance–covariance matrix.

Author(s)

Domenic Franjic

`predict.SDFMFit` *Generic prediction function for* `SDFMFit` *S3 objects*

Description

Predict all missing observations due to ragged edges in the data set plus horizon steps ahead.

Usage

```
## S3 method for class 'SDFMFit'
predict(object, horizon = 0, ...)
```

Arguments

`object`	`SDFMFit` object.
`horizon`	Number of forecasting steps into the future. Must be smaller than or equal to `x$factor_fcast_horizon`.
`...`	Additional parameters for the prediction function.

Value

A named list of plot objects:

data Object containing the original data. The object inherits its class from `object$data`: If data is provided as `zoo`, `data` will be a `zoo` object. If `data` is provided as matrix, `data` will be a (`no_of_factors`×`no_of_obs`) matrix.

data_missing_pred Object containing only the predictions of all missing observations plus the forecasts. Inherits its class from `object$data` as above.

data_imputed Object containing the observed data, predictions of all missing observations plus the forecasts. Inherits its class from `object$data` as above.

Author(s)

Domenic Franjic

Examples

```
data(factor_model)
no_of_vars <- dim(factor_model$data)[2]
no_of_factors <- dim(factor_model$factors)[2]
sdfm_fit <- twoStepSDFM(data = factor_model$data, delay = factor_model$delay,
                        selected = rep(floor(0.5 * no_of_vars), no_of_factors),
                        no_of_factors = no_of_factors, fcast_horizon = 5)
dfm_fit <- twoStepDenseDFM(data = factor_model$data, delay = factor_model$delay,
                           no_of_factors = no_of_factors, fcast_horizon = 5)
predict(sdfm_fit, horizon = 5)
predict(dfm_fit, horizon = 5)
```

print.KFSFit *Generic printing function for KFSFit S3 objects*

Description

Print a compact summary of a `KFSFit` object.

Usage

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

Arguments

`x`	`KFSFit` object.
`...`	Additional parameters.

Value

No return value, called for side effects.

Author(s)

Domenic Franjic

print.NoOfFactorsFit *Generic printing function for NoOfFactorsFit S3 objects*

Description

Print a compact summary of an `NoOfFactorsFit` object.

Usage

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

Arguments

`x`	`NoOfFactorsFit` object.
`...`	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

`print.SDFMcrossVal` *Generic print function for SDFMcrossVal S3 objects*

Description

Generic print function for SDFMcrossVal S3 objects

Usage

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

Arguments

`x`	`SDFMcrossVal` object.
`...`	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

`print.SDFMFit` *Generic printing function for SDFMFit S3 objects*

Description

Print a compact summary of an `SDFMFit` object.

Usage

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

Arguments

`x`	`SDFMFit` object.
`...`	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

`print.SDFMnowcast` *Generic print function for SDFMnowcast S3 objects*

Description

Generic print function for SDFMnowcast S3 objects

Usage

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

Arguments

`x`	`SDFMnowcast` object.
`...`	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

`print.SimulData` *Generic printing function for SimulData S3 objects*

Description

Print a compact summary of an `SimulData` object.

Usage

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

Arguments

x	SimulData object.
...	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

---

print.SPCAFit *Generic printing function for SPCAFit S3 objects*

Description

Print a compact summary of an SPCAFit object.

Usage

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

Arguments

x	SPCAFit object.
...	Additional parameters for the plotting functions.

Value

No return value; Prints a summary to the console.

Author(s)

Domenic Franjic

simFM *Simulate Dynamic Factor Models.*

Description

Simulate data from a linear Gaussian state-space model (latent factor model), with measurement equation

$$\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{f}_t + \boldsymbol{\xi}_t, \quad \boldsymbol{\xi}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_\xi),$$

and transition equation

$$\boldsymbol{f}_t = \sum_{p=1}^{P} \boldsymbol{\Phi}_p \boldsymbol{f}_{t-p} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_f).$$

for t = 1, . . . , T, as is used in, among others, Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*..

Usage

```
simFM(
  no_of_obs,
  no_of_vars,
  no_of_factors,
  loading_matrix,
  meas_error_mean,
  meas_error_var_cov,
  trans_error_var_cov,
  trans_var_coeff,
  factor_lag_order,
  delay = NULL,
  quarterfy = FALSE,
  quarterly_variable_ratio = 0,
  corr = FALSE,
  beta_param = Inf,
  seed = 20022024,
  burn_in = 1000,
  rescale = TRUE,
  starting_date = NULL,
  check_stationarity = FALSE,
  stationarity_check_threshold = 1e-05,
  parallel = FALSE
)
```

Arguments

no_of_obs	Integer number of observations.
no_of_vars	Integer number of Variables.

`no_of_factors` Integer number of factors.

`loading_matrix` Numeric (`no_of_vars` × `no_of_factors`) loading matrix.

`meas_error_mean`
Numeric vector of the means of the measurement errors.

`meas_error_var_cov`
Numeric (`no_of_vars` × `no_of_vars`) variance-covariance matrix of the measurement errors.

`trans_error_var_cov`
Numeric (`no_of_factors` × `no_of_factors`) variance-covariance matrix of the transition errors.

`trans_var_coeff`
Either a list of length `max_factor_lag_order` with each entry a numeric (`no_of_factors` × `no_of_factors`) VAR coefficient matrix or a matrix of dimensions (`no_of_factors` × (`no_of_factors` * `max_factor_lag_order`)) holding the VAR coefficients of the factor VAR process in each (`no_of_factors` × `no_of_factors`) block.

`factor_lag_order`
Integer order of the VAR process in the transition equation.

`delay` Integer vector of delays imposed onto the end of the data (ragged edges).

`quarterfy` Logical, whether or not some of the data should be aggregated to quarterly representations.

`quarterly_variable_ratio`
Ratio of variables ought to be quarterfied.

`corr` Logical, whether or not the measurement error should be randomly correlated inside the function using a random correlation matrix with off-diagonal elements governed by a beta-distribution.

`beta_param` Parameter of the beta-distribution governing the off-diagonal elements of the variance-covariance matrix of the measurement error.

`seed` 32-bit unsigned integer seed for all random processes inside the function.

`burn_in` Integer burn-in period of the simulated data ought to be discarded at the beginning of the sample.

`rescale` Logical, whether or not the variance of the measurement error should be rescaled by the common component to equalise the signal-to-noise ratio.

`starting_date` A date type object indicating the start of the dataset. If NULL (default), the function returns matrices with observations along the second dimension (i.e., time in columns). If specified, the function treats the data as a time series and returns a `zoo` object.

`check_stationarity`
Logical, whether or not the stationarity properties of the factor VAR process should be checked.

`stationarity_check_threshold`
Threshold of the stationarity check for when to deem an eigenvalue numerically negative.

`parallel` Logical, make use of Eigen internal parallel matrix operations.

Details

The delay vector indicates the number of observations at the end of the sample that will be set to NA for each variable. Here, delay refers to the number of months for monthly data and the number of quarters for quarterly data. For example, consider delay <- c(1, 1) and assume the variable with index 1 will be quarterfied. In that case, the variable with index 1 will be delayed by 1 quarter, i.e., it will be missing 3 observations at the end of the panel. The variable with index 2 will be delayed by 1 month, i.e., it will be missing 1 observation at the end of the panel. This convention differs from the delay object of the SimulData class this function returns. There, delay represents the number of months since the most recent publication. For monthly variables, these values coincide, but for quarterly variables they are inherently different.

If quarterfy = TRUE, floor(quarterly_variable_ratio * no_of_vars) variables will be aggregated to a quarterly representation using the geometric mean according to Mariano RS, 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..

If corr = TRUE, the matrix meas_error_var_cov is internally replaced by a random variance-covariance matrix: $\tilde{\boldsymbol{\Sigma}} := \boldsymbol{SRS}$, where $\boldsymbol{S}$ is a diagonal matrix with entries equal to sqrt(diag(meas_error_var_cov)) and $\boldsymbol{R}$ is a random correlation matrix. $\boldsymbol{R}$ is drawn according to Lewandowski D, Kurowicka D, Joe H (2009). "Generating random correlation matrices based on vines and extended onion method." *Journal of Multivariate Analysis*, **100**(9), 1989–2001. (see also https://stats.stackexchange.com/questions/2746/how-to-efficiently-generate-random-positive-semidefinite-correlation-matrices). The parameter beta_param governs the degree of cross-correlation of the off-diagonal elements. For more information see the literature cited above.

The random draws of the fundamental error terms are drawn within the C++ backend. Therefore, seed must be provided and set.seed() will not guarantee reproduceability.

Value

Returns a SimulData containing the following elements:

data If starting_date is provided, a zoo object, else, a (no_of_vars × no_of_obs) numeric matrix holding the simulated data.

factors If starting_date is provided, a zoo object, else a (no_of_factors × no_of_obs) numeric matrix holding the simulated latent factors.

trans_var_coeff Numeric (no_of_factors × (no_of_factors * factor_lag_order)) factor VAR coefficient matrix.

loading_matrix Numeric factor loading matrix.

meas_error If starting_date is provided, a zoo object, else a (no_of_vars × no_of_obs) numeric matrix holding the fundamental measurement errors.

meas_error_var_cov Numeric measurement error variance-covariance matrix.

trans_error_var_cov Numeric transition error variance-covariance matrix.

frequency Integer vector of variable frequencies.

delay Integer vector of variable delays, measured as the number of months since the latest available observation.

Author(s)

Domenic Franjic

References

Mariano RS, 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.

Lewandowski D, Kurowicka D, Joe H (2009). "Generating random correlation matrices based on vines and extended onion method." *Journal of Multivariate Analysis*, **100**(9), 1989–2001.

Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.

Examples

```
seed <- 02102025
set.seed(seed)
no_of_obs <- 100
no_of_vars <- 50
no_of_factors <- 3
trans_error_var_cov <- diag(1, no_of_factors)
loading_matrix <- matrix(round(rnorm(no_of_vars * no_of_factors)), no_of_vars, no_of_factors)
meas_error_mean <- rep(0, no_of_vars)
meas_error_var_cov <- diag(1, no_of_vars)
trans_var_coeff <- cbind(diag(0.5, no_of_factors), -diag(0.25, no_of_factors))
factor_lag_order <- 2
delay <- c(floor(rexp(no_of_vars, 1)))
quarterfy <- FALSE
quarterly_variable_ratio  <- 0
corr <- TRUE
beta_param <- 2
burn_in <- 999
starting_date <- "1970-01-01"
rescale <- TRUE
check_stationarity <- TRUE
stationarity_check_threshold <- 1e-10
factor_model <- simFM(no_of_obs = no_of_obs, no_of_vars = no_of_vars,
                      no_of_factors = no_of_factors, loading_matrix = loading_matrix,
                      meas_error_mean = meas_error_mean,
                      meas_error_var_cov = meas_error_var_cov,
                      trans_error_var_cov = trans_error_var_cov,
                      trans_var_coeff = trans_var_coeff,
                      factor_lag_order = factor_lag_order, delay = delay,
                      quarterfy = quarterfy,
                      quarterly_variable_ratio  = quarterly_variable_ratio, corr = corr,
                      beta_param = beta_param, seed = seed, burn_in = burn_in,
                      starting_date = starting_date, rescale = rescale,
                      check_stationarity = check_stationarity,
                      stationarity_check_threshold = stationarity_check_threshold)
print(factor_model)
spca_plots <- plot(factor_model)
spca_plots$`Factor Time Series Plots`
```

```
spca_plots$`Loading Matrix Heatmap`
spca_plots$`Meas. Error Var.-Cov. Matrix Heatmap`
spca_plots$`Meas. Error Var.-Cov. Eigenvalue Plot`
spca_plots$`Data Var.-Cov. Matrix Heatmap`
spca_plots$`Data Var.-Cov. Eigenvalue Plot`
```

`sparsePCA` *Sparse Principal Components Analysis*

Description

Estimate sparse sparse principal components via SPCA according to Zou H, Hastie T, Tibshirani R (2006). "Sparse principal component analysis." *Journal of Computational and Graphical Statistics*, **15**(2), 265–286..

Usage

```
sparsePCA(
  data,
  delay,
  selected,
  no_of_factors,
  ridge_penalty = 1e-06,
  lasso_penalty = NULL,
  max_iterations = 1000,
  weights = NULL,
  max_no_steps = NULL,
  comp_null = 1e-15,
  spca_conv_crit = 1e-04,
  parallel = FALSE,
  svd_method = "precise",
  normalise = TRUE,
  comp_var_expl = TRUE
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`delay`	Integer vector of variable delays, measured as the number of months since the latest available observation.
`selected`	Integer vector of the number of selected variables for each factor.
`no_of_factors`	Integer number of factors.
`ridge_penalty`	Numeric ridge penalty.

`lasso_penalty`	Numeric vector, lasso penalties for each factor (set to NULL to disable as stopping criterion).
`max_iterations`	Integer maximum number of iterations.
`weights`	Numeric vector, weights for each variable weighing the ℓ_1 size constraint.
`max_no_steps`	Integer number of LARS steps (set to NULL to disable as stopping criterion).
`comp_null`	Numeric computational zero.
`spca_conv_crit`	Conversion threshold for the SPCA algorithm.
`parallel`	Logical, whether or not to use Eigen's internal parallel matrix operations.
`svd_method`	Either "fast" or "precise". Option "fast" uses Eigen's BDCSVD divide and conquer method for the computation of the singular values. Option "precise" (default) implements the slower, but numerically more stable JacobiSVD method.
`normalise`	Logical, whether to normalise the loading matrix as in Zou H, Hastie T (2020). *elasticnet: Elastic-Net for Sparse Estimation and Sparse PCA*. R package version 1.3, https://CRAN.R-project.org/package=elasticnet.. Default is `TRUE`.
`comp_var_expl`	Logical, whether to compute the relative variance explained by each factor. Default is `TRUE``.

Details

The function takes three stopping criteria: `selected`, `lasso_penalty`, and `max_no_steps`. With `selected` the SPCA algorithm stops if each column of the estimated loading matrix has the corresponding number of non-zero loadings. This allows the user to directly control the degree of sparsity of each factor loading. With `lasso_penalty`, the SPCA algorithm stops as soon as the side-constraints of the inherent elastic-net problem are no longer satisfied. With `max_no_steps`, the SPCA algorithm only takes that many LARS steps for each factor loading's individual elastic-net problem before stopping. If all criteria are provided, the first one satisfied will stop the algorithm. For details see also (Zou et al. 2006) and (Zou and Hastie 2020).

Loosely, each SPCA algorithm iteration solves an elastic-net type problem for each column of the loading matrix. One can extend this problem to the adaptive elastic-net (Zou and Zhang 2009). The variable `weights` lets the user provide weights for each observation. These weights must be strictly greater than zero and are normalised internally to represent relative weights. For more information on the computational implementation of the weight extension in the context of SPCA see Zou Q, Zhang P (2024). "On General Weighted Adaptive Sparse Principal Component Analysis." In *Proceedings of the 2024 4th International Conference on Computational Modeling, Simulation and Data Analysis*, 335–340..

In each SPCA algorithm iteration, the function executes an SVD. To this end, Eigen provides two alternatives (Guennebaud et al. 2010): Option `precise` makes use of JacobiSVD. This method is numerically more stable, but computationally costly, especially for medium to large matrices. Option `fast` makes use of BDCSVD. This divide-and-conquer approach can lead to significant performance gains with respect to large matrices. BDCSVD, however, can be numerically unstable when Eigen is compiled with aggressive speed optimisations. In the context of the `R`, this should be of no concern. By default, `R` and most packages are compiled with "mild" `-O2` optimisation and without any additional aggressive optimisation flags. Nonetheless, one should checker whether both variants provide reasonably close results before switching to `fast`. For more information see Guennebaud G, Jacob B, others (2010). "Eigen." https://libeigen.gitlab.io..

Value

An object of class `SPCAFit` with components:

data Original data matrix.

loading_matrix_estim Numeric matrix of estimated factor loadings.

factor_estim Object containing the SPCA factor estimates. The object inherits its class from `data`: If `data` is provided as `zoo`, `factor_estim` will be a `zoo` object. If `data` is provided as `matrix`, `factor_estim` will be a (`no_of_factors` $\times$ `no_of_obs`) matrix.

total_var_expl Numeric total variance explained.

pct_var_expl Numeric vector relative variance explained by each factor.

Author(s)

Domenic Franjic

References

Zou H, Hastie T, Tibshirani R (2006). "Sparse principal component analysis." *Journal of Computational and Graphical Statistics*, **15**(2), 265–286.

Zou H, Zhang HH (2009). "On the adaptive elastic-net with a diverging number of parameters." *Annals of statistics*, **37**(4), 1733.

Guennebaud G, Jacob B, others (2010). "Eigen." https://libeigen.gitlab.io.

Zou H, Hastie T (2020). *elasticnet: Elastic-Net for Sparse Estimation and Sparse PCA*. R package version 1.3, https://CRAN.R-project.org/package=elasticnet.

Zou Q, Zhang P (2024). "On General Weighted Adaptive Sparse Principal Component Analysis." In *Proceedings of the 2024 4th International Conference on Computational Modeling, Simulation and Data Analysis*, 335–340.

Examples

```
data(factor_model)
set.seed(17032026)
no_of_factors <- 3
no_of_vars <- dim(factor_model$data)[2]
selected <- rep(floor(0.5 * no_of_vars), no_of_factors)
lasso_penalty <- exp(runif(no_of_factors, -10, 1))
max_no_steps <- 1000
spca_fit <- sparsePCA(data = factor_model$data, delay = factor_model$delay,
                      selected = selected, no_of_factors = no_of_factors,
                      ridge_penalty = 1e-2, lasso_penalty = lasso_penalty,
                      max_iterations = 1000, weights = NULL,
                      max_no_steps = max_no_steps, comp_null = 1e-15,
                      spca_conv_crit = 1e-04, parallel = FALSE,
                      svd_method = "precise", normalise = FALSE,
                      comp_var_expl = TRUE)
print(spca_fit)
spca_plots <- plot(spca_fit)
spca_plots$`Factor Time Series Plots`
```

```
spca_plots$`Loading Matrix Heatmap`
spca_plots$`Meas. Error Var.-Cov. Matrix Heatmap`
spca_plots$`Eigenvalue Plot`
spca_plots$`Variance Explained Chart`
```

`twoStepDenseDFM` *Two Step Dense Dynamic Factor Model Estimator.*

Description

Estimate a dense dynamic factor model with measurement equation

$$\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{f}_t + \boldsymbol{\xi}_t, \quad \boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\xi),$$

and transition equation

$$\boldsymbol{f}_t = \sum_{p=0}^{P} \boldsymbol{\Phi}_p \boldsymbol{f}_{t-p} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_f).$$

using principal components analysis and the Kalman Filter and Smoother according to Giannone D, Reichlin L, Small D (2008). "Nowcasting: The real-time informational content of macroeconomic data." *Journal of Monetary Economics*, **55**(4), 665-676. ISSN 0304-3932. doi:10.1016/j.jmoneco.2008.05.010. and Doz C, Giannone D, Reichlin L (2011). "A two-step estimator for large approximate dynamic factor models based on Kalman filtering." *Journal of Econometrics*, **164**(1), 188-205. ISSN 0304-4076. doi:10.1016/j.jeconom.2011.02.012..

Usage

```
twoStepDenseDFM(
  data,
  delay,
  no_of_factors,
  max_factor_lag_order = 10,
  lag_estim_criterion = "BIC",
  decorr_errors = TRUE,
  comp_null = 1e-15,
  parallel = FALSE,
  fcast_horizon = 0,
  jitter = 1e-08
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`delay`	Integer vector of variable delays.
`no_of_factors`	Integer number of factors.

`max_factor_lag_order`
: Integer maximum order of the VAR process in the transition equation.

`lag_estim_criterion`
: Information criterion used for the estimation of the factor VAR order (`"BIC"` (default), `"AIC"`, `"HIC"`).

`decorr_errors`
: Logical, whether or not the errors should be decorrelated.

`comp_null`
: Numeric computational zero.

`parallel`
: Logical, whether or not to use Eigen's internal parallel matrix operations.

`fcast_horizon`
: Integer number of additional Filter predictions into the future.

`jitter`
: Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance covariance matrix of the measurement errors.

Details

The function performs a two-step estimation procedure for dense dynamic factor models as described in Giannone D, Reichlin L, Small D (2008). "Nowcasting: The real-time informational content of macroeconomic data." *Journal of Monetary Economics*, **55**(4), 665-676. ISSN 0304-3932. doi:10.1016/j.jmoneco.2008.05.010. and Doz C, Giannone D, Reichlin L (2011). "A two-step estimator for large approximate dynamic factor models based on Kalman filtering." *Journal of Econometrics*, **164**(1), 188-205. ISSN 0304-4076. doi:10.1016/j.jeconom.2011.02.012.. In the first step, the factor loading matrix is estimated using PCA. In the second step the latent factors are estimated using the univariate representation of the Kalman Filter and Smoother (Koopman and Durbin 2000).

With respect to the univariate representation of the Kalman filter and smoother, `decorr_errors` indicates whether the data should be decorrelated internally prior to filtering and smoothing. `jitter` is added to the diagonal elements of the measurement variance–covariance matrix. For more details, see `kalmanFilterSmoother`.

Value

An object of class SDFMFit with main components:

data Original data object.

loading_matrix_estim Numeric matrix of estimated factor loadings.

smoothed_factors Object containing the SPCA factor estimates. The object inherits its class from data: If data is provided as `zoo`, `factor_estim` will be a `zoo` object. If data is provided as matrix, `factor_estim` will be a (`no_of_factors`×`no_of_obs` matrix.

smoothed_state_variance (`no_of_factors`×(`no_of_factors` * `no_of_obs`)) matrix, where each (`no_of_factors` ×`no_of_factors`) block represents the smoother uncertainty at time point$t$.

factor_var_lag_order Integer order of the VAR process in the state equation.

error_var_cov_cholesky_factor Numeric lower-triangular Cholesky factor of the estimated measurement error variance–covariance matrix.

llt_success_code Integer indicating the status of the Cholesky factorization: `0` = LLT succeeded, `-1` = LLT failed but LDLT succeeded, `-2` = both failed and errors are treated as uncorrelated.

Author(s)

Domenic Franjic

References

Koopman SJ, Durbin J (2000). "Fast filtering and smoothing for multivariate state space models." *Journal of Time Series Analysis*, **21**(3), 281–296.

Giannone D, Reichlin L, Small D (2008). "Nowcasting: The real-time informational content of macroeconomic data." *Journal of Monetary Economics*, **55**(4), 665-676. ISSN 0304-3932. doi:10.1016/j.jmoneco.2008.05.010.

Guennebaud G, Jacob B, others (2010). "Eigen." https://libeigen.gitlab.io.

Doz C, Giannone D, Reichlin L (2011). "A two-step estimator for large approximate dynamic factor models based on Kalman filtering." *Journal of Econometrics*, **164**(1), 188-205. ISSN 0304-4076. doi:10.1016/j.jeconom.2011.02.012.

See Also

sparsePCA: Routine for fitting estimating a sparse factor loading matrix.

kalmanFilterSmoother: Routine for filtering and smoothing latent factors.

twoStepSDFM: Two-step estimation routine for a sparse dynamic factor model.

Examples

```
data(factor_model)
no_of_vars <- dim(factor_model$data)[2]
no_of_factors <- dim(factor_model$factors)[2]
dfm_fit <- twoStepDenseDFM(data = factor_model$data, delay = factor_model$delay,
                           no_of_factors = no_of_factors)
print(dfm_fit)
dfm_plots <- plot(dfm_fit)
dfm_plots$`Factor Time Series Plots`
dfm_plots$`Loading Matrix Heatmap`
dfm_plots$`Meas. Error Var.-Cov. Matrix Heatmap`
dfm_plots$`Meas. Error Var.-Cov. Eigenvalue Plot`
```

`twoStepSDFM` *Two Step Sparse Dynamic Factor Model Estimator.*

Description

Estimate a sparse dynamic factor model with measurement equation

$$\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{f}_t + \boldsymbol{\xi}_t, \quad \boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\xi),$$

and transition equation

$$\boldsymbol{f}_t = \sum_{p=0}^{P} \boldsymbol{\Phi}_p \boldsymbol{f}_{t-p} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_f).$$

using sparse principal components analysis and the Kalman Filter and Smoother according to Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*..

Usage

```
twoStepSDFM(
  data,
  delay,
  selected,
  no_of_factors,
  max_factor_lag_order = 10,
  lag_estim_criterion = "BIC",
  decorr_errors = TRUE,
  ridge_penalty = 1e-06,
  lasso_penalty = NULL,
  max_iterations = 1000,
  max_no_steps = NULL,
  weights = NULL,
  comp_null = 1e-15,
  spca_conv_crit = 1e-04,
  parallel = FALSE,
  fcast_horizon = 0,
  jitter = 1e-08,
  svd_method = "precise"
)
```

Arguments

`data`	Numeric (no_of_vars × no_of_obs) matrix of data or zoo/xts object sampled at the same frequency.
`delay`	Integer vector of variable delays, measured as the number of months since the latest available observation.
`selected`	Integer vector of the number of selected variables for each factor.
`no_of_factors`	Integer number of factors.
`max_factor_lag_order`	Integer maximum order of the VAR process in the transition equation.
`lag_estim_criterion`	Information criterion used for the estimation of the factor VAR order (`"BIC"` (default), `"AIC"`, `"HIC"`).
`decorr_errors`	Logical, whether or not the errors should be decorrelated.
`ridge_penalty`	Ridge penalty.

`lasso_penalty`	Numeric vector, lasso penalties for each factor (set to NULL to disable as stopping criterion).
`max_iterations`	Integer maximum number of iterations.
`max_no_steps`	Integer number of LARS steps (set to NULL to disable as stopping criterion).
`weights`	Numeric vector, weights for each variable weighing the ℓ_1 size constraint.
`comp_null`	Numeric computational zero.
`spca_conv_crit`	Conversion threshold for the SPCA algorithm.
`parallel`	Logical, whether or not to use Eigen's internal parallel matrix operations.
`fcast_horizon`	Integer number of additional Filter predictions into the future.
`jitter`	Numerical jitter for stability of internal solver algorithms. The jitter is added to the diagonal entries of the variance covariance matrix of the measurement errors.
`svd_method`	Either "fast" or "precise". Option "fast" uses Eigen's BDCSVD divide and conquer method for the computation of the singular values. Option "precise" (default) implements the slower, but numerically more stable JacobiSVD method (Guennebaud et al. 2010).

Details

The function performs a two-step estimation procedure for sparse dynamic factor models as described in Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.. In the first step, the factor loading matrix is estimated using SPCA (Zou et al. 2006). This will shrink some of the loadings towards or exactly to zero. In the second step the latent factors are estimated using the univariate representation of the Kalman Filter and Smoother (Koopman and Durbin 2000).

The function takes three stopping criteria for the SPCA algorithm: `selected`, `lasso_penalty`, and `max_no_steps`. The argument `weights` allows specifying weights for the ℓ_1 constraint. `svd_method` controls the decomposition method for internal SVDs. For a detailed description of these arguments and the SPCA step, see `sparsePCA`.

With respect to the univariate representation of the Kalman filter and smoother, `decorr_errors` indicates whether the data should be decorrelated internally prior to filtering and smoothing. `jitter` is added to the diagonal elements of the measurement variance–covariance matrix. For more details, see `kalmanFilterSmoother`.

For more information on the two-step estimation procedure see Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*..

Value

An object of class SDFMFit with main components: #'

data Original data object.

loading_matrix_estim Numeric matrix of estimated factor loadings.

smoothed_factors Object containing the SPCA factor estimates. The object inherits its class from data: If data is provided as `zoo`, `factor_estim` will be a `zoo` object. If data is provided as matrix, `factor_estim` will be a (`no_of_factors`×`no_of_obs` matrix.

smoothed_state_variance (`no_of_factors`×(`no_of_factors` * `no_of_obs`)) matrix, where each (`no_of_factors` ×`no_of_factors`) block represents the smoother uncertainty at time point$t$.

factor_var_lag_order Integer order of the VAR process in the state equation.

error_var_cov_cholesky_factor Numeric lower-triangular Cholesky factor of the estimated measurement error variance–covariance matrix.

llt_success_code Integer indicating the status of the Cholesky factorization: `0` = LLT succeeded, `-1` = LLT failed but LDLT succeeded, `-2` = both failed and errors are treated as uncorrelated.

Author(s)

Domenic Franjic

References

Koopman SJ, Durbin J (2000). "Fast filtering and smoothing for multivariate state space models." *Journal of Time Series Analysis*, **21**(3), 281–296.

Zou H, Hastie T, Tibshirani R (2006). "Sparse principal component analysis." *Journal of Computational and Graphical Statistics*, **15**(2), 265–286.

Guennebaud G, Jacob B, others (2010). "Eigen." https://libeigen.gitlab.io.

Franjic D, Schweikert K (2024). "Nowcasting Macroeconomic Variables with a Sparse Mixed Frequency Dynamic Factor Model." *Available at SSRN 4733872*.

See Also

`sparsePCA`: Routine for fitting estimating a sparse factor loading matrix.

`kalmanFilterSmoother`: Routine for filtering and smoothing latent factors.

`twoStepDenseDFM`: Two-step estimation routine for a dense dynamic factor model.

Examples

```
data(factor_model)
no_of_vars <- dim(factor_model$data)[2]
no_of_factors <- dim(factor_model$factors)[2]
sdfm_fit <- twoStepSDFM(data = factor_model$data, delay = factor_model$delay,
                        selected = rep(floor(0.5 * no_of_vars), no_of_factors),
                        no_of_factors = no_of_factors)
print(sdfm_fit)
sdfm_plots <- plot(sdfm_fit)
sdfm_plots$`Factor Time Series Plots`
sdfm_plots$`Loading Matrix Heatmap`
sdfm_plots$`Meas. Error Var.-Cov. Matrix Heatmap`
sdfm_plots$`Meas. Error Var.-Cov. Eigenvalue Plot`
```

Index