Package: quadVAR (via r-universe)

November 20, 2024

Title Quadratic Vector Autoregression

Version 0.1.1

Description Estimate quadratic vector autoregression models with the strong hierarchy using the RAMP algorithm, compare the performance with linear models, and construct networks with partial derivatives.

License GPL (>= 3)

URL https://github.com/Sciurus365/quadVAR, https://sciurus365.github.io/quadVAR/

BugReports https://github.com/Sciurus365/quadVAR/issues

Imports cli, dplyr, ggplot2, magrittr, ncvreg, qgraph, RAMP, rlang, shiny, shinythemes, stats, stringr, tibble, tidyr

Suggests nonlinearTseries, SIS, testthat (>= 3.0.0)

Remotes Sciurus365/RAMP

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.0

Config/pak/sysreqs libglpk-dev make libicu-dev libjpeg-dev libpng-dev libxml2-dev zlib1g-dev

Repository https://sciurus365.r-universe.dev

RemoteUrl https://github.com/Sciurus365/quadVAR

RemoteRef HEAD

RemoteSha c0ac7d4ff4e4cef0112a33f607a045bd0d514cc7

Contents

`block_cv` *Use Block Cross-Validation to Evaluate Models*

Description

This function uses block cross-validation to evaluate a model. The data is split into blocks, and the model is fit on all but one block and evaluated on the remaining block. This process is repeated for each block, and the mean squared error is calculated for each model.

Usage

```
block_cv(
  data,
  dayvar = NULL,
  model,
  block = 10,
  lowerbound = -Inf,
  upperbound = Inf,
  detail = FALSE,
  metric = "MSE"
)
```

Arguments

`data`	A data frame.
`dayvar`	A character string. The name of the variable that represents the day. This is required because this function use dayvar to specify the time point before the test block should not be used to predict the time point after the test block. If dayvar is not specified, in the original dataset, then please add one constant variable as dayvar, and specify it both here and in the function passed to `model`.

`model`	A function. The model to be evaluated. The function should take a data frame as its first argument and return a `quadVAR` object. It can be, for example, `function(x) quadVAR(x, vars = c("var1", "var2"))`
`block`	An integer. The number of blocks to use in the cross-validation. The default is 10.
`lowerbound`	A numeric value or a vector with the same length as the number of variables that specifies the lower bound of the predicted values. If the predicted value is less than this value, it will be replaced by this value. The default value is -Inf.
`upperbound`	A numeric value or a vector with the same length as the number of variables that specifies the upper bound of the predicted values. If the predicted value is greater than this value, it will be replaced by this value. The default value is Inf.
`detail`	A logical. If `TRUE`, the function will return the predictions for each model. The default is `FALSE`, which only returns the mean squared error for each model.
`metric`	A character vector. The metric to be used to evaluate the model. The default is "MSE", which calculates the mean squared error. The other option is "MAE", which calculates the mean absolute error. Only effective when `detail = FALSE`.

Value

Depending on `detail`. If `FALSE`, it returns a list of mean squared errors for each model. If `TRUE`, it returns a list with the mean squared errors for each model, the true data, and the predictions for each model.

`compare_4_emo` *Compare estimated model with true model for 4-emotion model*

Description

This function compares the estimated model with the true model for the 4-emotion model. It prints out the estimated coefficients and the true coefficients for the main effects and interaction effects.

Usage

```
compare_4_emo(model, silent = FALSE)
```

Arguments

`model`	The estimated model, using data simulated from `sim_4_emo()`, and model estimated using `quadVAR()`.
`silent`	Whether to print out the results.

Value

Silently return data frame with the estimated coefficients and the true coefficients for the main effects and interaction effects, while printing out the results rounded to two digits if `silent = FALSE`.

---

find_index *Find index of data that satisfies certain conditions*

Description

Find index of data that satisfies certain conditions

Usage

```
find_index(data, dayvar, beepvar)
```

Arguments

`data`	A data frame.
`dayvar`	String indicating assessment day. Adding this argument makes sure that the first measurement of a day is not regressed on the last measurement of the previous day. IMPORTANT: only add this if the data has multiple observations per day.
`beepvar`	Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing!

Value

A list of two vectors of indices.

---

get_adj_mat *Extract the adjacency matrix from a quadVAR object.*

Description

Extract the adjacency matrix from a quadVAR object.

Usage

```
get_adj_mat(model, value)
```

Arguments

`model`	A quadVAR object.
`value`	The `actual_value` in the output of `linear_quadVAR_network()`.

Value

An adjacency matrix.

`linear_quadVAR_network`
Linearize a quadVAR object to produce a network.

Description

A quadVAR object is nonlinear, which means that the relationship between variables are not the same across different values of the variables. This function linearizes a quadVAR object by specifying the values of the variables that the linearized model will be based on, to facilitate interpretation. The linearized model is then expressed in an adjacency matrix, which can be used to produce a network.

Usage

```
linear_quadVAR_network(model, value = NULL, value_standardized = TRUE)

## S3 method for class 'linear_quadVAR_network'
plot(x, interactive = FALSE, ...)
```

Arguments

`model`	A quadVAR object.
`value`	A numeric vector of length 1 or the same as the number of nodes, that specifies the values of the variables that the linearized model will be based on. If the length is 1, the same value will be used for all variables. The default value is `NULL`, in which case the value will be set to 0 in calculation, which means (if `value_standardized = TRUE`) the linearized model will be based on the mean values of all variables.
`value_standardized`	A logical value that specifies whether the input value is standardized or not. If TRUE, the input value will be regarded as standardized value, i.e., mean + `value` * sd (e.g., 0 is the mean, 1 is mean + sd, ...). If FALSE, the input value will regarded as in the raw scale of the input data. If the raw dataset was already standardized, this parameter does not have an effect. The default value is `TRUE`.
`x`	A linear_quadVAR_network object.
`interactive`	Whether to produce an interactive plot using `shiny` (in which the user can change the values of variables interactively) or a static plot using `qgraph::qgraph()`. Default is `FALSE`.
`...`	Other arguments passed to `qgraph::qgraph()`.

Value

A linear_quadVAR_network with the following elements:

- `adj_mat`: the adjacency matrix of the linearized network.
- `standardized_value`: the standardized value that the linearized model is based on.
- `actual_value`: the value in the raw scale of the input data.
- `model`: the input quadVAR object.
- `value_standardized`: the same as the input.

Methods (by generic)

- `plot(linear_quadVAR_network)`: Produce a plot for the linearized quadVAR model. If `interactive = FALSE`, the output will be a qgraph object, which can be further used to calculate centrality measures using, e.g., `qgraph::centrality()` and `qgraph::centralityPlot()`.

References

The idea of this linearization function is inspired by Kroc, E., & Olvera Astivia, O. L. (2023). The case for the curve: Parametric regression with second- and third-order polynomial functions of predictors should be routine. Psychological Methods. https://doi.org/10.1037/met0000629

`partial_plot` *Make a partial plot of a variable in a model This function takes a quadVAR model as input, and returns a plot of the partial effect of a variable on the dependent variable (controlling all other variables and the intercept), for higher and lower levels of the moderator variable split by the median.*

Description

Make a partial plot of a variable in a model This function takes a quadVAR model as input, and returns a plot of the partial effect of a variable on the dependent variable (controlling all other variables and the intercept), for higher and lower levels of the moderator variable split by the median.

Usage

```
partial_plot(model, y, x, moderator)
```

Arguments

`model`	A quadVAR model
`y`	The dependent variable
`x`	The variable for which the partial effect is plotted
`moderator`	The moderator variable

Value

A ggplot object

`predict.quadVAR` *Predict the values of the dependent variables using the quadVAR model*

Description

Predict the values of the dependent variables using the quadVAR model

Usage

```
## S3 method for class 'quadVAR'
predict(
  object,
  newdata = NULL,
  donotpredict = NULL,
  lowerbound = -Inf,
  upperbound = Inf,
  with_const = FALSE,
  ...
)
```

Arguments

`object`	A quadVAR object.
`newdata`	A data frame or tibble containing at least the values of the independent variables, dayvar, and beepvar (if used in model estimation). If NULL, the original data used to fit the model will be used.
`donotpredict`	NOT IMPLEMENTED YET! A character vector of the model names that are not used for prediction. Possible options include "AR", "VAR", "VAR_full", "quadVAR_full", "all_others", with NULL as the default. If set "all_others", then only a `quadVAR` model will be estimated. For datasets with large number of variables, you may set this parameter to "quadVAR_full" to save time.
`lowerbound`	A numeric value or a vector with the same length as the number of variables that specifies the lower bound of the predicted values. If the predicted value is less than this value, it will be replaced by this value. The default value is -Inf.
`upperbound`	A numeric value or a vector with the same length as the number of variables that specifies the upper bound of the predicted values. If the predicted value is greater than this value, it will be replaced by this value. The default value is Inf.

`with_const` A logical value indicating whether to include the constant variables in the prediction. Those variables were automatically excluded in the estimation procedure. The default value is FALSE. When set to TRUE, the lowerbound and upperbound should be a vector with the same length as the number of variables in the model, including the constant variables. The values of the constant variables will be ignored though because their predicted values are always the same, which is the constant value in the input data.

`...` Other arguments passed to the `RAMP::predict.RAMP()` function.

Value

A data frame or tibble containing the predicted values of the dependent variables. If a value cannot be predicted (e.g., because the corresponding previous time point is not in the data), it will be NA.

`quadVAR` *Estimate lag-1 quadratic vector autoregression models*

Description

This function estimate regularized nonlinear quadratic vector autoregression models with strong hierarchy using the `RAMP::RAMP()` algorithm, and also compare it with the linear AR, regularized VAR, and unregularized (full) VAR and quadratic VAR models.

Usage

```
quadVAR(
  data,
  vars,
  dayvar = NULL,
  beepvar = NULL,
  penalty = "LASSO",
  tune = "EBIC",
  donotestimate = NULL,
  SIS_options = list(),
  RAMP_options = list()
)

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

## S3 method for class 'quadVAR'
summary(object, ...)

## S3 method for class 'quadVAR'
coef(object, ...)
```

```
## S3 method for class 'coef_quadVAR'
print(
  x,
  use_actual_names = TRUE,
  abbr = FALSE,
  minlength = 3,
  omit_zero = TRUE,
  digits = 2,
  row.names = FALSE,
  ...
)

## S3 method for class 'quadVAR'
plot(x, value = NULL, value_standardized = TRUE, interactive = FALSE, ...)
```

Arguments

`data`	A `tibble`, data.frame, or matrix that represents a time series of vectors, with each row as a time step.
`vars`	A character vector of the variable names used in the model.
`dayvar`	String indicating assessment day. Adding this argument makes sure that the first measurement of a day is not regressed on the last measurement of the previous day. IMPORTANT: only add this if the data has multiple observations per day.
`beepvar`	Optional string indicating assessment beep per day. Adding this argument will cause non-consecutive beeps to be treated as missing!
`penalty`	The penalty used for the linear and regularized VAR models. Possible options include "LASSO", "SCAD", "MCP", with "LASSO" as the default.
`tune`	Tuning parameter selection method. Possible options include "AIC", "BIC", "EBIC", with "EBIC" as the default.
`donotestimate`	A character vector of the model names that are not estimated. Possible options include, "NULL_model", "AR", "VAR", "VAR_full", "quadVAR_full", "all_others", with NULL as the default. If set "all_others", then only a `quadVAR` model will be estimated. For datasets with large number of variables, you may set this parameter to "quadVAR_full" to save time.
`SIS_options`	A list of other parameters for the `SIS::tune.fit()` function. This is used for the regularized VAR models.
`RAMP_options`	A list of other parameters for the `RAMP::RAMP()` function. This is used for the nonlinear quadratic VAR model.
`...`	For `print.quadVAR`, additional arguments passed to `print.coef_quadVAR()`. For `print.coef_quadVAR`, additional arguments passed to `print.data.frame()`.
`object, x`	An `quadVAR` object. (For `print.coef_quadVAR`, an `coef_quadVAR` object returned by `coef.quadVAR()`.)

use_actual_names
: Logical. If TRUE, the actual variable names are used in the output. If FALSE, the names "X1", "X2", etc., are used in the output. Default is TRUE.

abbr
: Logical. If TRUE, the output is abbreviated. Default is FALSE.

minlength
: the minimum length of the abbreviations.

omit_zero
: Logical. If TRUE, the coefficients that are zero are omitted. Default is FALSE.

digits
: the minimum number of significant digits to be used: see print.default.

row.names
: logical (or character vector), indicating whether (or what) row names should be printed.

value
: A numeric vector of length 1 or the same as the number of nodes, that specifies the values of the variables that the linearized model will be based on. If the length is 1, the same value will be used for all variables. The default value is NULL, in which case the value will be set to 0 in calculation, which means (if value_standardized = TRUE) the linearized model will be based on the mean values of all variables.

value_standardized
: A logical value that specifies whether the input value is standardized or not. If TRUE, the input value will be regarded as standardized value, i.e., mean + value * sd (e.g., 0 is the mean, 1 is mean + sd, ...). If FALSE, the input value will regarded as in the raw scale of the input data. If the raw dataset was already standardized, this parameter does not have an effect. The default value is TRUE.

interactive
: Whether to produce an interactive plot using shiny (in which the user can change the values of variables interactively) or a static plot using qgraph::qgraph(). Default is FALSE.

Value

An quadVAR object that contains the following elements:

- NULL_model: A list of NULL models for each variable.
- AR_model: A list of linear AR models for each variable.
- VAR_model: A list of regularized VAR models for each variable.
- VAR_full_model: A list of unregularized (full) VAR models for each variable.
- quadVAR_model: A list of regularized nonlinear quadratic VAR models for each variable.
- quadVAR_full_model: A list of unregularized (full) nonlinear quadratic VAR models for each variable.
- data,vars,penalty,tune,SIS_options,RAMP_options: The input arguments.
- data_x,data_y: The data directly used for modeling.

Methods (by generic)

- `print(quadVAR)`: Print the coefficients for a quadVAR object. See `coef.quadVAR()` and `print.coef_quadVAR()` for details.
- `summary(quadVAR)`: Summary of a quadVAR object. Different IC definitions used by different packages (which differ by a constant) are unified to make them comparable to each other.
- `coef(quadVAR)`: Extract the coefficients from a quadVAR object.
- `plot(quadVAR)`: Produce a plot for the linearized quadVAR model. Equivalent to first produce a linear quadVAR network using `linear_quadVAR_network()`, then use `plot.linear_quadVAR_network()`.

Functions

- `print(coef_quadVAR)`: Print the coefficients from a quadVAR object.

See Also

`linear_quadVAR_network()`

Examples

```
set.seed(1614)
data <- sim_4_emo(time = 200, sd = 1)
plot(data[, "x1"])
qV1 <- quadVAR(data, vars = c("x1", "x2", "x3", "x4"))
summary(qV1)
coef(qV1)
plot(qV1)
# Compare the estimation with the true model
plot(true_model_4_emo())
plot(qV1, value = 0, value_standardized = FALSE, layout = plot(true_model_4_emo())$layout)
```

---

quadVAR_to_dyn_eqns *Transform a quadVAR object to a list of dynamic equations.*

Description

Transform a quadVAR object to a list of dynamic equations.

Usage

```
quadVAR_to_dyn_eqns(model, minus_self = TRUE)
```

Arguments

`model`	A quadVAR object.
`minus_self`	Whether to subtract the term itself from the equation. If `TRUE`, the equation will be in the form of (0 =) `... - X1`; if `FALSE`, the equation will be in the form of (X1 =)`...`.

Value

A list of dynamic equations in characters. You can also use `rlang::parse_expr()` to parse them into expressions.

`sim_4_emo` *Simulate a 4-emotion model*

Description

This function simulates a 4-emotion model which is nonlinear, bistable, discrete, and (almost) centered to zero. Adapted from the model described by van de Leemput et al. (2014).

Usage

```
sim_4_emo(time = 200, init = c(1.36, 1.36, 4.89, 4.89), sd = 1)
```

Arguments

`time`	The number of time steps to simulate.
`init`	A vector of initial values for the four variables. Default is c(1.36, 1.36, 4.89, 4.89), which is one of the stable states of the model.
`sd`	The standard deviation of the noise.

Value

A matrix with the simulated data.

References

van de Leemput, I. A., Wichers, M., Cramer, A. O., Borsboom, D., Tuerlinckx, F., Kuppens, P., ... & Scheffer, M. (2014). Critical slowing down as early warning for the onset and termination of depression. Proceedings of the National Academy of Sciences, 111(1), 87-92.

See Also

`true_model_4_emo()`, `compare_4_emo()`, `quadVAR()`

`true_model_4_emo` *True model for 4-emotion model*

Description

This function generate the true model for the 4-emotion model. It can used to compare the estimated model with the true model, or to plot the true model.

Usage

```
true_model_4_emo(...)

## S3 method for class 'true_model_4_emo'
coef(object, ...)

## S3 method for class 'true_model_4_emo'
print(x, which = NULL, ...)
```

Arguments

`...`	Not in use.
`object`	A true_model_4_emo object.
`x`	A true_model_4_emo object.
`which`	Which model to print out. There are four models in total, corresponding to the four variables.

Value

A true_model_4_emo object.

NULL, but prints out the true model.

Methods (by generic)

- `coef(true_model_4_emo)`: This function returns the coefficients for the 4-emotion model. It is also used in other functions to generate the linearized version of the true model and to make plots. It returns a list of coefficients for the 4-emotion model, in the same format as `coef.quadVAR()`
- `print(true_model_4_emo)`: This function prints out the true model for the 4-emotion model in the same format as `RAMP::RAMP()`, to help users to compare the true model and the estimated model.

See Also

`true_model_4_emo()`, `compare_4_emo()`, `quadVAR()`

Examples

```
coef(true_model_4_emo())
plot(true_model_4_emo())
## Not run:
plot(true_model_4_emo(), interactive = TRUE)

## End(Not run)
```

`tune.fit` *Using the* **glmnet** *and* **ncvreg** *packages, fits a Generalized Linear Model or Cox Proportional Hazards Model using various methods for choosing the regularization parameter*

Description

This function is modified from `SIS::tune.fit()`. It is used to tune the regularization parameter for the regularized VAR models. This wrapper is used because of the following reasons.

1. The original `SIS::tune.fit()` function does not return the value of the information criteria that we would like to use.
2. We use the ncvreg package exclusively (so we removed the code using the glmnet package). This is to make the result more consistent, and also because the ncvreg package has better support for the calculation of information criteria.
3. We also removed the generalized linear model (GLM) option, and the cross-validation option because we do not use them.
4. We use stats::AIC() and stats::BIC() instead of the ones using SIS:::loglik() to make the calculation methods more consistent.
5. We added `...` to allow the user to pass additional arguments to the ncvreg::ncvreg() function.

Usage

```
tune.fit(
  x,
  y,
  family = "gaussian",
  penalty = c("SCAD", "MCP", "lasso"),
  concavity.parameter = switch(penalty, SCAD = 3.7, 3),
  tune = c("aic", "bic", "ebic"),
  type.measure = c("deviance", "class", "auc", "mse", "mae"),
  gamma.ebic = 1,
  ...
)
```

Arguments

`x`	The design matrix, of dimensions n * p, without an intercept. Each row is an observation vector.
`y`	The response vector of dimension n * 1. Quantitative for `family='gaussian'`, non-negative counts for `family='poisson'`, binary (0-1) for `family='binomial'`. For `family='cox'`, `y` should be an object of class `Surv`, as provided by the function `Surv()` in the package **survival**.
`family`	Response type (see above).
`penalty`	The penalty to be applied in the regularized likelihood subproblems. 'SCAD' (the default), 'MCP', or 'lasso' are provided.
`concavity.parameter`	The tuning parameter used to adjust the concavity of the SCAD/MCP penalty. Default is 3.7 for SCAD and 3 for MCP.
`tune`	Method for selecting the regularization parameter along the solution path of the penalized likelihood problem. Options to provide a final model include `tune='cv'`, `tune='aic'`, `tune='bic'`, and `tune='ebic'`. See references at the end for details.
`type.measure`	Loss to use for cross-validation. Currently five options, not all available for all models. The default is `type.measure='deviance'`, which uses squared-error for gaussian models (also equivalent to `type.measure='mse'` in this case), deviance for logistic and poisson regression, and partial-likelihood for the Cox model. Both `type.measure='class'` and `type.measure='auc'` apply only to logistic regression and give misclassification error and area under the ROC curve, respectively. `type.measure='mse'` or `type.measure='mae'` (mean absolute error) can be used by all models except the `'cox'`; they measure the deviation from the fitted mean to the response. For `penalty='SCAD'` and `penalty='MCP'`, only `type.measure='deviance'` is available.
`gamma.ebic`	Specifies the parameter in the Extended BIC criterion penalizing the size of the corresponding model space. The default is `gamma.ebic=1`. See references at the end for details.
`...`	additional arguments to be passed to the ncvreg::ncvreg() function.

Details

Original description from `SIS::tune.fit()`:

This function fits a generalized linear model or a Cox proportional hazards model via penalized maximum likelihood, with available penalties as indicated in the **glmnet** and **ncvreg** packages. Instead of providing the whole regularization solution path, the function returns the solution at a unique value of , the one optimizing the criterion specified in tune.

Value

Returns an object with

`ix`	The vector of indices of the nonzero coefficients selected by the maximum penalized likelihood procedure with `tune` as the method for choosing the regularization parameter.

`a0`	The intercept of the final model selected by `tune`.
`beta`	The vector of coefficients of the final model selected by `tune`.
`fit`	The fitted penalized regression object.
`lambda`	The corresponding lambda in the final model.
`lambda.ind`	The index on the solution path for the final model.

Author(s)

Jianqing Fan, Yang Feng, Diego Franco Saldana, Richard Samworth, and Yichao Wu

References

Jerome Friedman and Trevor Hastie and Rob Tibshirani (2010) Regularization Paths for Generalized Linear Models Via Coordinate Descent. *Journal of Statistical Software*, **33**(1), 1-22.

Noah Simon and Jerome Friedman and Trevor Hastie and Rob Tibshirani (2011) Regularization Paths for Cox's Proportional Hazards Model Via Coordinate Descent. *Journal of Statistical Software*, **39**(5), 1-13.

Patrick Breheny and Jian Huang (2011) Coordiante Descent Algorithms for Nonconvex Penalized Regression, with Applications to Biological Feature Selection. *The Annals of Applied Statistics*, **5**, 232-253.

Hirotogu Akaike (1973) Information Theory and an Extension of the Maximum Likelihood Principle. In *Proceedings of the 2nd International Symposium on Information Theory*, BN Petrov and F Csaki (eds.), 267-281.

Gideon Schwarz (1978) Estimating the Dimension of a Model. *The Annals of Statistics*, **6**, 461-464.

Jiahua Chen and Zehua Chen (2008) Extended Bayesian Information Criteria for Model Selection with Large Model Spaces. *Biometrika*, **95**, 759-771.

Examples

```
## Not run:
set.seed(0)
data("leukemia.train", package = "SIS")
y.train <- leukemia.train[, dim(leukemia.train)[2]]
x.train <- as.matrix(leukemia.train[, -dim(leukemia.train)[2]])
x.train <- standardize(x.train)
model <- tune.fit(x.train[, 1:3500], y.train, family = "binomial", tune = "bic")
model$ix
model$a0
model$beta

## End(Not run)
```

Index