Package ‘ARGOS’

January 18, 2024

Type Package

Title Automatic Regression for Governing Equations (ARGOS)

Version 0.1.1

Maintainer Kevin Egan <kevin.egan@durham.ac.uk>

Description Comprehensive set of tools for performing system identification of both linear and nonlinear dynamical systems directly from data. The Automatic Regression for Governing Equations (ARGOS) simplifies the complex task of constructing mathematical models of dynamical systems from observed input and output data, supporting various types of systems, including those described by ordinary differential equations. It employs optimal numerical derivatives for enhanced accuracy and employs formal variable selection techniques to help identify the most relevant variables, thereby enabling the development of predictive models for system behavior analysis.

Depends R (>= 3.6.0)

Imports Matrix, glmnet, Metrics, boot, tidyverse, magrittr, tidyr,
signal, parallel, deSolve

Suggests testthat, knitr, rmarkdown, devtools

License GPL-3

URL <https://github.com/kevinegan31/ARGOS-Package>

Contact Please report bugs and other issues to

<kevin.egan@durham.ac.uk>.

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Kevin Egan [aut, cre],
Weizhen Li [aut],
Rui Carvalho [aut],
Yuzheng Zhang [aut]

Repository CRAN

Date/Publication 2024-01-18 16:00:02 UTC

R topics documented:

`alasso` *Adaptive Lasso*

Description

This function performs adaptive lasso regression using the cv.glmnet function, then refits the model using ordinary least squares.

Usage

```
alasso(data, index, weights_method = c("ols", "ridge"), ols_ps = TRUE)
```

Arguments

`data`	A data frame or matrix containing the predictors and response. The response must be in the first column.
`index`	A numeric vector of indices indicating the rows of 'data' to use for the adaptive lasso regression.
`weights_method`	A character string specifying the method to calculate the weights. Can be either "ols" or "ridge". Default is "ols".
`ols_ps`	A logical scalar. If TRUE (default), the function returns the coefficients from the OLS fit. If FALSE, it returns the coefficients from the lasso fit.

Value

A numeric vector of coefficients. If 'ols_ps' is TRUE, these are the coefficients from the OLS fit. If 'ols_ps' is FALSE, these are the coefficients from the lasso fit. If an error occurs during the lasso or OLS fit, the function returns a vector of NAs.

argos *Automatic Regression for Governing Equations (ARGOS)*

Description

This function performs sparse regression on a data set to identify the governing equations of the system. It takes a list of data from 'build_design_matrix' then applies the Lasso or Adaptive Lasso for variable selection.

Usage

```
argos(
  design_matrix,
  library_type = c("poly", "four", "poly_four"),
  state_var_deriv = 1,
  alpha_level = 0.05,
  num_samples = 2000,
  sr_method = c("lasso", "alasso"),
  weights_method = NULL,
  ols_ps = TRUE,
  parallel = c("no", "multicore", "snow"),
  ncpus = NULL
)
```

Arguments

`design_matrix`	A list containing data frame, vector of predictor variable orders for 'theta', and derivative matrix.
`library_type`	A character vector (default: c("poly", "four", "poly_four")) specifying the type of library being used.
`state_var_deriv`	An integer. The index of the state variable for which the derivative is calculated. Default is 1.
`alpha_level`	A numeric scalar. The level of significance for confidence intervals. Default is 0.05.
`num_samples`	An integer. The number of bootstrap samples. Default is 2000.
`sr_method`	A character string. The sparse regression method to be used, either "lasso" or "alasso". Default is "lasso".
`weights_method`	A string or NULL. The method for calculating weights in the Adaptive Lasso. If NULL, ridge regression pilot estimates are used. Default is NULL.
`ols_ps`	A logical. If TRUE, post-selection OLS is performed after the Lasso or Adaptive Lasso. Default is TRUE.
`parallel`	A character string. The type of parallel computation to be used, either "no", "multicore" or "snow". Default is "no".
`ncpus`	An integer or NULL. The number of cores to be used in parallel computation. If NULL, the function will try to detect the number of cores. Default is NULL.

Value

A list with three elements: - point_estimates: a vector of point estimates for the coefficients. - ci: a matrix where each column represents the lower and upper bounds of the confidence interval for a coefficient. - identified_model: a matrix of coefficients of the identified model.

Examples

```
# Identify the x1 equation of the Duffing Oscillator with ARGOS.
# Output provides point estimates, confidence intervals, and identified model.
x_t <- duffing_oscillator(n=1000, dt = 0.01,
                          init_conditions = c(1, 0),
                          gamma_value = 0.1, kappa_value = 1,
                          epsilon_value = 5, snr = 49)
duffing_design_matrix <-
       build_design_matrix(x_t, dt = 0.01, sg_poly_order = 4,
                           library_degree = 5, library_type = "poly")
design_matrix <- duffing_design_matrix
state_var_deriv = 1 # Denotes first equation/derivative to be identified
alpha_level = 0.05
num_samples = 10
sr_method = "lasso"
weights_method = NULL
ols_ps = TRUE
parallel = "no"
ncpus = NULL
library_type <- "poly"
perform_argos <- argos(design_matrix = design_matrix,
                       library_type = library_type,
                       state_var_deriv = state_var_deriv,
                       alpha_level = alpha_level,
                       num_samples = num_samples,
                       sr_method = "lasso",
                       weights_method = NULL,
                       ols_ps = TRUE,
                       parallel = "no",
                       ncpus = NULL)
perform_argos$point_estimates
perform_argos$ci
perform_argos$identified_model
```

---

build_design_matrix *Build Design Matrix*

Description

This function first smooths the data and approximates the derivative before building the design matrix to include monomial and fourier terms.

Usage

```
build_design_matrix(
  x_t,
  dt = 1,
  sg_poly_order = 4,
  library_degree = 5,
  library_type = c("poly", "four", "poly_four")
)
```

Arguments

x_t	Matrix of observations.
dt	Time step (default is 1).
sg_poly_order	Polynomial order for Savitzky-Golay Filter.
library_degree	Degree of polynomial library (default is 5).
library_type	Type of library to use. Can be one of "poly", "four", or "poly_four".

Value

A list with two elements:

- sorted_theta - A matrix with sorted polynomial/trigonometric terms.
- monomial_orders - A vector indicating the order of each polynomial term.
- xdot_filtered - A matrix with derivative terms (dependent variable).

Examples

```
# Build a design matrix using the Duffing Oscillator as the state-space.
# Output provides matrix, and derivative matrix monomial orders
# (needed for running `argos`).
x_t <- duffing_oscillator(n=5000, dt = 0.01,
                          init_conditions = c(1, 0),
                          gamma_value = 0.1, kappa_value = 1,
                          epsilon_value = 5, snr = 49)
duffing_design_matrix <-
         build_design_matrix(x_t, dt = 0.01, sg_poly_order = 4,
                             library_degree = 5, library_type = "poly")
head(duffing_design_matrix$sorted_theta)
```

---

cubic2d_system *Cubic 2D System*

Description

Simulates a two-dimensional damped oscillator with cubic dynamics and optional noise.

Usage

```
cubic2d_system(n, init_conditions, dt, snr = Inf)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`init_conditions`	Initial conditions as a numeric vector of length 2.
`dt`	Time step between observations.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates a two-dimensional damped oscillator with cubic dynamics. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a 2D cubic system with 100 time points and no noise
data <- cubic2d_system(n = 100, init_conditions = c(1, 2), dt = 0.01, snr = Inf)
```

---

duffing_oscillator *Duffing Oscillator*

Description

Simulates the Duffing oscillator with optional noise.

Usage

```
duffing_oscillator(
  n,
  dt,
  init_conditions,
  gamma_value,
  kappa_value,
  epsilon_value,
  snr = Inf
)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`dt`	Time step between observations.
`init_conditions`	Initial conditions as a numeric vector of length 2.
`gamma_value`	Value of gamma parameter.
`kappa_value`	Value of kappa parameter.
`epsilon_value`	Value of epsilon parameter.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates a Duffing oscillator with the specified parameters. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a Duffing oscillator with 100 time points and no noise
data <- duffing_oscillator(
  n = 100,
  dt = 0.01,
  init_conditions = c(2, 6),
  gamma_value = 0.1,
  kappa_value = 1,
  epsilon_value = 5,
  snr = Inf
)
```

`lasso` *Lasso*

Description

This function performs lasso regression using the cv.glmnet function, then refits the model using ordinary least squares.

Usage

```
lasso(data, index, ols_ps = TRUE)
```

Arguments

- `data` — A data frame or matrix containing the predictors and response. The response must be in the first column.
- `index` — A numeric vector of indices indicating the rows of 'data' to use for the lasso regression.
- `ols_ps` — A logical scalar. If TRUE (default), the function returns the coefficients from the OLS fit. If FALSE, it returns the coefficients from the lasso fit.

Value

A numeric vector of coefficients. If 'ols_ps' is TRUE, these are the coefficients from the OLS fit. If 'ols_ps' is FALSE, these are the coefficients from the lasso fit. If an error occurs during the lasso or OLS fit, the function returns a vector of NAs.

`linear2d_system` *Linear 2D System*

Description

Simulates a two-dimensional damped oscillator with linear dynamics and optional noise.

Usage

```
linear2d_system(n, init_conditions, dt, snr = Inf)
```

Arguments

- `n` — Number of time points (rounded to the nearest integer).
- `init_conditions` — Initial conditions as a numeric vector of length 2.
- `dt` — Time step between observations.
- `snr` — Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates a two-dimensional damped oscillator with linear dynamics. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a 2D linear system with 100 time points and no noise
data <- linear2d_system(n = 100, init_conditions = c(-1, 1), dt = 0.01, snr = Inf)
```

`linear3d_system` *Linear 3D System*

Description

Simulates a three-dimensional linear dynamical system with optional noise.

Usage

```
linear3d_system(n, init_conditions, dt, snr = Inf)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`init_conditions`	Initial conditions as a numeric vector of length 3.
`dt`	Time step between observations.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates a three-dimensional linear dynamical system. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a 3D linear system with 100 time points and no noise
data <- linear3d_system(n = 100, init_conditions = c(1, 2, 3), dt = 0.01, snr = Inf)
```

`lorenz_system` *Lorenz Chaotic System*

Description

Simulates the Lorenz chaotic system with optional noise.

Usage

```
lorenz_system(n, init_conditions, dt, snr = Inf)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`init_conditions`	Initial conditions as a numeric vector of length 3 (X, Y, Z).
`dt`	Time step between observations.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates the Lorenz chaotic system with the specified parameters. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable (X, Y, Z).

Examples

```
# Simulate the Lorenz system with 1000 time points and no noise
data <- lorenz_system(
  n = 1000,
  dt = 0.01,
  init_conditions = c(-8, 7, 27),
  snr = Inf
)
```

lotka_volterra *Lotka-Volterra System*

Description

Simulates the Lotka-Volterra predator-prey system with optional noise.

Usage

```
lotka_volterra(n, init_conditions, dt, snr = Inf)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`init_conditions`	Initial conditions as a numeric vector of length 2.
`dt`	Time step between observations.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates the Lotka-Volterra predator-prey system with the specified parameters. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a Lotka-Volterra system with 100 time points and no noise
data <- lotka_volterra(
  n = 100,
  dt = 0.01,
  init_conditions = c(2, 1),
  snr = Inf
)
```

rossler_system *Rossler Chaotic System*

Description

Simulates the Rossler chaotic system with optional noise.

Usage

```
rossler_system(n, dt, init_conditions, a, b, c, snr = Inf)
```

Arguments

n	Number of time points (rounded to the nearest integer).
dt	Time step between observations.
init_conditions	Initial conditions as a numeric vector of length 3 (X, Y, Z).
a	Rossler parameter 1
b	Rossler parameter 2
c	Rossler parameter 3
snr	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates the Rossler chaotic system with the specified parameters. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable (X, Y, Z).

Examples

```
# Simulate the Rossler system with 1000 time points and no noise
data <- rossler_system(
  n = 1000,
  dt = 0.01,
  init_conditions = c(0, 2, 0),
  a = 0.2, b = 0.2, c = 5.7,
  snr = Inf
)
```

sg_optimal_combination

Optimal Savitzky-Golay Filter Parameters Finder

Description

This function finds the optimal parameters for the Savitzky-Golay filter by evaluating combinations of polynomial orders and window lengths.

Usage

```
sg_optimal_combination(x_t, dt = 1, polyorder)
```

Arguments

`x_t`	A numeric vector or one-column matrix. The data to be smoothed.
`dt`	A numeric scalar. The time-step interval of the data. Default is 1.
`polyorder`	A numeric scalar. The order of the polynomial to be used in the Savitzky-Golay filter. If not specified, 4 will be used by default.

Value

A list with three elements: - sg_combinations: a matrix where each row represents a combination of polynomial order and window length tried. - sg_order_wl: a vector of length 2 with the optimal polynomial order and window length. - f_dist: a data frame with the mean squared error of the differences between the original data and the smoothed data for each combination.

vdp_oscillator

Van der Pol Oscillator

Description

Simulates the Van der Pol oscillator with optional noise.

Usage

```
vdp_oscillator(n, dt, init_conditions, mu, snr = Inf)
```

Arguments

`n`	Number of time points (rounded to the nearest integer).
`dt`	Time step between observations.
`init_conditions`	Initial conditions as a numeric vector of length 2.
`mu`	Parameter controlling the nonlinear damping level of the system.
`snr`	Signal-to-noise ratio (in dB). Use Inf for no noise.

Details

This function simulates a Van der Pol oscillator with the specified parameters. It uses the specified time step and initial conditions to compute the system's state over time. If a non-Infinite SNR is provided, Gaussian noise is added to the system.

Value

A numeric matrix representing the system's state over time. Each row corresponds to a time point, and each column represents a variable.

Examples

```
# Simulate a Van der Pol oscillator with 100 time points and no noise
data <- vdp_oscillator(
  n = 100,
  dt = 0.01,
  init_conditions = c(-1, 1),
  mu = 1.2,
  snr = Inf
)
```

Index