Package ‘plspolychaos’

March 1, 2017

Version 1.1-1
Date 2017-03-01
Title Sensitivity Indexes from Polynomial Chaos Expansions and PLS
Author A. Bouvier [aut], J.-P. Gauchi [cre], A. Bensadoun [ctb]
Maintainer Annie Bouvier <annie.bouvier@inra.fr>
Depends methods
Imports utils, lhs, MASS, graphics, grDevices, stats
Description Computation of sensitivity indexes by using a method based on a truncated Polynomial Chaos Expansion of the response and regression PLS, for computer models with correlated continuous inputs, whatever the input distribution. The truncated Polynomial Chaos Expansion is built from the multivariate Legendre orthogonal polynomials.

The number of runs (rows) can be smaller than the number of monomials. It is possible to select only the most significant monomials.

Of course, this package can also be used if the inputs are independent. Note that, when they are independent and uniformly distributed, the package 'polychaosbasics' is more appropriate.

License GPL (>= 2)
URL https://cran.r-project.org/package=plspolychaos,
http://genome.jouy.inra.fr/logiciels/plspolychaos

NeedsCompilation yes
Repository CRAN
Date/Publication 2017-03-01 12:29:23

R topics documented:

plspolychaos-package ........................................ 2
analyticsPolyLeg ........................................ 3
calcPLSPCE .................................................. 5
desrdata .................................................... 6
Description

Computation of sensitivity indexes by using a method based on a truncated Polynomial Chaos Expansion of the response and regression PLS, for computer models with correlated continuous inputs, whatever the input distribution. The truncated Polynomial Chaos Expansion is built from the multivariate Legendre orthogonal polynomials. The number of runs (rows) can be smaller than the number of monomials. It is possible to select only the most significant monomials. Of course, this package can also be used if the inputs are independent. Note that, when they are independent and uniformly distributed, the package 'polychaosbasics' is more appropriate.

Details

The Legendre chaos polynomials are calculated, either on a user provided dataset by function `polyleg`, or on a simulated LHS by function `analyticsPolyLeg`. Then, function `calcPLSPCE` calculates PLS-regression coefficients, PLS-PCE sensitivity indexes and some other results.

Author(s)

A. Bouvier [aut], J.-P. Gauchi [cre], A. Bensadoun [ctb]

Maintainer: Annie Bouvier <annie.bouvier@inra.fr>

References


See Also

`polychaosbasics` package.
Examples

```r
### First example: the dataset is simulated
nlhs <- 200 # number of rows
degree <- 6 # polynomial degree
set.seed(42)# fix the seed for reproducible results
# Generate data and calculate Legendre polynomials
# Independent inputs; response calculated by the Ishigami function
pce <- analyticsPolyLeg(nlhs, degree, 'ishigami')
# Compute the PLS-PCE sensitivity indexes for ten components
ret <- calcPLSPCE(pce, nc=10)
# Plot the result
## Not run: plot(ret, pce)

### Second example: the dataset is provided and the
### most significant monomials are selected
# Load the dataset
load(system.file("extdata", "ishigami200.Rda", package="plspolychaos"))
X <- ishi200[, -ncol(ishi200)] # inputs
Y <- ishi200[, ncol(ishi200)] # output
# Build Legendre polynomial with the 50 most significant monomials
pce <- polyLeg(X, Y, degree=6, forward=50)
# Compute the PLS-PCE sensitivity indexes
ret <- calcPLSPCE(pce, nc=10)
print(ret, all=TRUE)
```

analyticsPolyLeg

Simulate a Dataset and Calculate Legendre Polynomials

Description

This function simulates a LHS and calculates the Legendre polynomials, optionally reducted to the most significant monomials.

The inputs are generated by the function `randomLHS` (from package `lhs`). Note that they are uniformly and independently sampled. The output is calculated by using the Ishigami [Saltelli, 2000, Chap. 2] or Sobol [Sobol’, 2003] functions. Legendre polynomials are then computed after calibration within the bounds [-1, +1].

Usage

```r
analyticsPolyLeg(nlhs, degree, model.fun, forward=NULL)
```

Arguments

- `nlhs` integer equal to the number of rows of the dataset.
- `degree` integer equal to the degree of the polynomial. Should be greater than 1 and less than 11.
- `model.fun` string equal to the required model. Valid values are 'ishigami' and 'sobol'.

```r
```
forward NULL or an integer equal to the required number of monomials. A null value (the default), or a value less than the number of inputs or greater than the total number of monomials, means that all the monomials are kept. See details.

Details

- The Ishigami function has three inputs that are linked to the output $Y$ according to:

$$Y = \sin(X_1) + 7 \times (\sin(X_2))^2 + 0.1 \times (X_3)^4 \times \sin(X_1)$$

Each $X_j$ is a uniform random variable on the interval $[-\pi, +\pi]$.

- The Sobol function has height inputs. The four first ones only are generated by using the function randomLHS. The four last are set to 0.5 (see Gauchi, 2017). The output $Y$ is then the product of:

$$(4 \times X_j - 2 + A_j)/(1 + A_j)$$

for $j$ in 1 to 8, and $A = (1, 2, 5, 10, 20, 50, 100, 500)$

- When the value of the argument forward is non NULL, it should be an integer equal to the required number of the monomials (let say $q$). The $q$ monomials are selected, among all the monomials of the full polynomial, by all the linear simple regressions of the output versus all the monomials. Those associated with the $q$ largest $R^2$ values are kept.

Value

An objet of class PCEpoly.

Note

The returned values are dependent on the random seed.

References


See Also

- Function polyLeg calculates Legendre polynomials on a user dataset.
- Function calcPLSPCE calculates PLS-PCE sensivity indexes from the returned object.
calcPLSPCE

Examples

nlhs <- 200 # number of rows in the dataset
degree <- 6 # polynomial degree
set.seed(42) # fix the seed for reproducible results
### Data simulation and creation of the full polynomials
pce <- analyticsPolyLeg(nlhs, degree, 'ishigami')
print(pce)
### Selection of the 50 most significant monomials
pcef <- analyticsPolyLeg(nlhs, degree, 'ishigami', forward=50)
print(pcef)

Description

Compute the optimal number of components, the PLS-PCE sensitivity indexes and related results.

Usage

calcPLSPCE(pce, nc = 2)

Arguments

pce 
an object of class PCEpoly. Design to analyze.
nc 
integer. Required number of components.

Value

An object of class PLSPCE.

Examples

### Load the dataset
load(system.file("extdata", "ishigami200.Rda", package="plspolychaos"))
X <- ishi200[, -ncol(ishi200)] # inputs
Y <- ishi200[, ncol(ishi200)] # output
### Creation of the full polynomials
degree <- 6 # polynomial degree
pce <- polyLeg(X, Y, degree)
### Compute the PLS-PCE sensitivity indexes for 25 components
ret <- calcPLSPCE(pce, nc=25)
descrdata  

**Main Characteristics of the Dataset**

**Description**

Display the number of rows, the mean, standard deviation, range and correlations of the inputs and output.

**Usage**

descrdata(X, Y)

**Arguments**

- **X**  
  matrix with as many columns as inputs. Dataset of inputs.

- **Y**  
  vector of length equal to the number of rows in X. Model outputs.

**Value**

Nothing. It is a display function.

**Examples**

```r
### Load the dataset
load(system.file("extdata", "ishigami200.Rda", package="plspolychaos"))
X <- ishi200[, -ncol(ishi200)] # inputs
Y <- ishi200[, ncol(ishi200)] # output
### Data characteristics
descrdata(X, Y)
```

getNames  

**Display Structure of a Class**

**Description**

Display the names, class and length of all the slots of a PCEpoly object.

**Usage**

getNames(object)

**Arguments**

- **object**  
  object from class PCEpoly.
Details

It is a generic function. Its method is defined in classe \texttt{PCEpoly}.

Value

Nothing. It is a display function.

See Also

\texttt{Classe PCEpoly}.

Examples

### Load the dataset

```r
load(system.file("extdata", "ishigami200.Rda", package="plspolychaos"))
X <- ishi200[, -ncol(ishi200)] # inputs
Y <- ishi200[, ncol(ishi200)] # output
pce <- polyLeg(X, Y, degree=6)
### Display recursively the names, class and
### length of all the components.
getNames(pce)
```

Description

Container of the polynomial description structure.

Objects from the Class

Objects from this class are created by calls to functions \texttt{polyLeg} or \texttt{analyticsPolyLeg}. They are stored in the slot design in the object of class \texttt{PCEpoly} returned by these functions.

Slots

- \texttt{Data}: matrix with as many columns as inputs and as many rows as monomials plus one. Element \((i,j)\) is an integer equal to the degree of the input \(j\) in the monomial \(i-1\). The first row is equal to zero: it is for the constant term.
- \texttt{degree}: integer equal to the polynomial degree.
- \texttt{total.nmono}: integer equal to the number of monomials in the full polynomial.

Methods

\texttt{print} signature(x = "PCEdesign", all=FALSE, ...): method of function \texttt{print}. If option all is set to TRUE, all the monomials are printed. The additional arguments are passed to the \texttt{print.default} function.

\texttt{show} signature(object = "PCEdesign"): same as function \texttt{print}, without any arguments.
See Also

• Functions `polyLeg` and `analyticsPolyLeg`, creators of objects from this class.
• Class `PCEpoly` in which objects from this class are stored.

---

**PCEpoly-class**

Class "PCEpoly"

---

Description

Container of the PCE design: the computed values of the monomials and the polynomial structure description.

Objects from the Class

Objects from this class are created by calls to functions `polyLeg` or `analyticsPolyLeg`.

Slots

`Ndata`: matrix. The computed values of the monomials of the Legendre polynomial. The number of rows is the number of rows of the dataset. The number of columns is the number of monomials plus one. The first column is equal to one: it is for the constant term.

`struc`: object of class `PCEdesign`. Matrix coding the polynomial structure.

`nvx`: integer equal to the number of inputs.

`call`: expression of class ‘call’. The command which creates the object.

Methods

`getNames` signature(object = "PCEpoly"): display the names, class and length of all the components.

`print` signature(object = "PCEpoly", all=FALSE, ...): method of function `print`. The polynomial expression is printed when option `all` is set to `TRUE`. The additional arguments are passed to the `print.default` function.

`show` signature(object = "PCEpoly"): same as function `print`, without any arguments.

See Also

• Functions `polyLeg` and `analyticsPolyLeg`, creators of objects from this class
• the generic method `getNames`

Examples

`showClass("PCEpoly")`
Description

An S4 class container for the result of the function `calcPLSPCE`.

Slots

indexes: matrix with as many rows as inputs and three columns. Values of the PLS-PCE sensitivity indexes. The column labels are LE, PE, TPE.
   - `indexes[i, "LE"]` is the Linear Effect of the input i.
   - `indexes[i, "PE"]` is the Polynomial Effect. It is the effect of the monomials in which only the input i appears.
   - `indexes[i, "TPE"]` is the Total Polynomial Effect. It is the effect of all the monomials in which the input i appears.

indexes.percent: matrix. Percentages of the PLS-PCE sensitivity indexes, i.e values of `indexes` expressed as percentages of the sums of their columns.

ncopt: number of the optimal component.

R2: matrix with as many rows as components and three columns. The columns are labeled R2, %R2 and %R2cum. They store respectively the R2 values, percentages of their column sums, and cumulated values of their percentages.

Q2: matrix with as many rows as components and two columns. The columns are labeled Q2 and Q2cum. They store respectively the Q2 and the Q2cum values.

rmsep: matrix with as many rows as components and one column. Root mean square error predictions.

COEF: matrix with as many rows as monomials plus one and as many columns as components. PLS-regression coefficients. The first row is the constant term.

betaCR: matrix with as many rows as monomials and as many columns as components. The centered-reduced PLS-regression coefficients, for all the components.

y.hat: vector of length equal to the number of rows of the dataset. Metamodel outputs.

STRUC: object of class `PCEdesign` coding the polynomial structure.

Methods

`print` signature(`x = "PLSPCE"`, `all=FALSE`, `...`): method of function `print`. When option `all` is FALSE, some slots are hidden: `RMSEP`, `coef`, `betaCR`, `y.hat`, `STRUC`. When option `all` is set to TRUE, the names of the hidden slots are printed. The additional arguments are passed to the `print.default` function.

`show` signature(`object = "PLSPCE"`): same as function `print`, without any arguments.
plot signature(x = "PLSPCE", pce, options = c("fit", "bar", "compo")): method of function plot. Different plots are drawn, according to options. When it includes “fit”, computer model outputs against metamodel outputs. When it includes “bar”, barplots of the polynomial and total polynomial effects (PE and TPE). When it includes “compo”, TPE against components. This last plot requires the calculation of the TPE for each component, which may take some time.

Note

The COEF and STRUC slots can be used for making predictions for Legendre-coded new input values.

See Also

Function calcPLSPCE, creator of objects from this class.

polyLeg Calculate Legendre Polynomials on a Dataset

Description

This function calculates Legendre polynomials, optionally reducted to the most significant monomials, on a user dataset.

Legendre polynomials are computed after calibration within the bounds [-1, +1].

Usage

polyLeg(lhs, Y, degree, forward=NULL)

Arguments

lhs matrix with as many columns as inputs. Dataset of inputs. Generally, a space filling filling design is used for forming this dataset. Typically, this is a simple LHS (see McKay, 1979) or a modified LHS.

Y vector of length equal to the number of rows in lhs. Model outputs.

degree integer greater than 1 and less than 11. Degree of the polynomial.

forward NULL or an integer equal to the required number of monomials. A null value (the default), or a value less than the number of inputs or greater than the total number of monomials, means that all the monomials are kept. See details.

Details

When the value of the argument forward is non NULL, it should be an integer equal to the required number of the monomials (let say q). The q monomials are selected, among all the monomials of the full polynomial, by all the linear simple regressions of the output versus all the monomials. Those associated with the q largest \( R^2 \) values are kept.
Value

An objet of class `PCEpoly`.

References


See Also

- Function `analyticsPolyLeg` builds Legendre polynomials from a simulated dataset.
- Function `calcPLSPCE` calculates PLS-PCE sensitivity indexes from the returned object.

Examples

```r
### Load the dataset
load(system.file("exdata", "ishigami200.Rda", package="plspolychaos"))
X <- ishi200[, -ncol(ishi200)] # inputs
Y <- ishi200[, ncol(ishi200)] # output
degree <- 6 # polynomial degree
### Creation of the full polynomials
pce <- polyLeg(X, Y, degree)
print(pce)
### Selection of the 50 most significant monomials
pcef <- polyLeg(X, Y, degree, forward=50)
print(pcef)
```
Index

*Topic **classes**
  - PCEdesign-class, 7
  - PCEpoly-class, 8
  - PLSPCE-class, 9

*Topic **package**
  - plspolychaos-package, 2

*Topic **regression**
  - analyticsPolyLeg, 3
  - calcPLSPCE, 5
  - plspolychaos-package, 2
  - polyLeg, 10

analyticsPolyLeg, 2, 3, 7, 8, 11
calcPLSPCE, 2, 4, 5, 9–11
descrdata, 6
getNames, 6, 8
getNames, PCEpoly-method (getNames), 6

lhs, 3
PCEdesign, 8, 9
PCEdesign-class, 7
PCEpoly, 4–8, 11
PCEpoly-class, 8
plot, 10
plot.PLSPCE (PLSPCE-class), 9
PLSPCE, 5
PLSPCE-class, 9
plspolychaos (plspolychaos-package), 2
plspolychaos-package, 2
polychaosbasics, 2
polyLeg, 2, 4, 7, 8, 10
print, 7–9
print.default, 7–9
print.PCEdesign (PCEdesign-class), 7
print.PCEpoly (PCEpoly-class), 8
print.PLSPCE (PLSPCE-class), 9

randomLHS, 3, 4