Package 'GPC'

February 19, 2015

Type Package

Title Generalized Polynomial Chaos

Version 0.1

Depends R (>= 2.7.0), randtoolbox, orthopolynom, ks, lars

Date 2013-02-01

Author Miguel Munoz Zuniga and Jordan Ko

Maintainer Miguel Munoz Zuniga<miguel.munoz-zuniga@ifpen.fr>

Description

A generalized polynomial chaos expansion of a model taking as input independent random variables is achieved. A statistical and a global sensitivity analysis of the model are also carried out.

License GPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2014-12-18 21:08:58

R topics documented:

GPC-package *Generalized Polynomial Chaos*

Description

Generalized polynomial chaos expansion of a model taking independent random input distribution. Three different methods are available for the expansion: the Gauss quadrature method, the adaptive sparse basis method and the least angle regression based adaptive sparse basis method. A statistical and a global sensitivity analysis of the model is then carried out.

Details

The PCE package implements a polynomial chaos expansion of a model. The main option is the choice between three strategy to expand the model into a polynomial chaos basis.

- The Gauss quadrature method where the the Gauss quadrature points are used to estimate the integrales giving the coefficients of the expansion.
- The adaptive sparse method where the sampled design is iteratively enriched until the expansion coefficients estimation, by regression, is accurate. Meanwile a relevant selection of the polynomial basis to keep in the expansion is carried out.
- The adaptive sparse method based on the least angle regression.

Model managing

The PCE package has been designed to work with either models written in R than external models such as heavy computational codes. This is achieved with the input argument model present in all functions of this package. The argument model is expected to be either a funtion or NULL.

If model = m where m is a function, it will be invoked once by y <- m(X).

X is the design of experiments, i.e. a data.frame with p columns (the input factors) and n lines (each, an experiment), and y is the vector of length n of the model responses. The model in invoked once for the whole design of experiment. The argument model can be left to NULL. This is refered to as the decoupled approach and used with external computational codes that rarely run on the statistician's computer. See decoupling.

References

G. Blatman and B. Sudret, 2010, *An adapive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis*, Probabilistic Engineering Mechanics, 25, 183–197.

G. Blatman and B. Sudret, 2011, *Adapive sparse polynomial chaos expansion based on least angle regression*, Journal of Computational Physics, 230, 2345–2367.

J. Ko, D. Lucor and P. Sagaut, 2008, *On Sensitivity of Two-Dimensional Spatially Developing Mixing Layer With Respect to Uncertain Inflow Conditions*, Physics of Fluids, 20(7), 07710201-07710220.

M. Munoz Zuniga ang S. Kucherenko, 2013, *Metamodelling with independent and dependent inputs*, Computer Physics Communications,xx ,xx.

See Also

The model management of the PCE package has some similarity with the one used in the package "sensitivity"

`CreateQuadrature` *Create a full or sparse numerical quadrature.*

Description

Creates a full-tensor multivariate quadrature based on one-dimensional quadrature rules or a Smolyak sparse-tensor multivariate quadrature based on nested one-dimensional quadrature rules.

Usage

```
CreateQuadrature(N,L,QuadPoly,ExpPoly,QuadType,ParamDistrib)
```

Arguments

`N`	Number of random variables
`L`	Level of quadrature used in the approximation.
`QuadPoly`	The type of one-dimensional quadrature rule to be used. For `SPARSE`, one can use `ClenshawCurtis` or `Fejer`. For `FULL`, one could choose `Hermite`,`Legendre`,`Jacobi` or `Laguerre`
`ExpPoly`	The polynomial used in the expansion. Options are `Hermite`, `Legendre`, `Jacobi`, `Laguerre`
`QuadType`	Type of quadrature. Options are `SPARSE` or `FULL`
`ParamDistrib`	Shape parameters used for the definition of random variables.

Value

QuadSize	Number of quadrature points
QuadNodes	A (QuadSize x n) matrix containing the QuadSize d-dimensional quadrature points.
QuadWeights	A n-tuple vector containing the quadrature weights associated with each quadrature point
PolyNodes	A (m x n) matrix containing the m d-dimensional multivariate polynomial evaluated at the n quadrature points.

Author(s)

Jordan Ko

References

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulationof stochastic shear flows*, Doctoral thesis, University of Paris VI.

See Also

tell.GPCE.quad

Examples

```
rm(list = setdiff(ls(), lsf.str()))

# random variable definitgeion
d <- 3              # number of random variables
L <- 4              # quadrature level in each dimention.
                    # could be anisotropic eg c(3,4,5) for full quadrature

# PCE definition
QuadType <- "FULL"              # type of quadrature
QuadPoly <- rep("LEGENDRE",d)   # polynomial to use
ExpPoly <- rep("LEGENDRE",d)    # polynomial to use
ParamDistrib <- NULL

# QuadType <- "SPARSE"                                  # type of quadrature
# QuadPoly <- 'ClenshawCurtis'                          # polynomial to use
# ExpPoly <- rep("LEGENDRE",d)    # polynomial to use

Quadrature = CreateQuadrature(d,L,QuadPoly,ExpPoly,QuadType,ParamDistrib) # quadrature
```

`generatePCEcoeff` *Determine the PCE coefficients using a numerical quadrature approach*

Description

The inner product used to determine the PCE coefficient is approximated using a numerical quadrature where the integral is approximated with a weighted sum of the function evaluations on some quadrature points.

Usage

```
generatePCEcoeff(m,n,y,PhiZn,weights)
```

Arguments

`m`	Number of PCE terms
`n`	Number of quadrature points
`y`	A n-tuple vector containing the solution of the function evaluated at the n quadrature points
`PhiZn`	A (m x n) matrix containing the m d-dimensional multivariate polynomial evaluated at the n quadrature points.
`weights`	A n-tuple vector containing the quadrature weights associated with each quadrature point

Value

`ym`	A m-tuple vector containing the PCE coefficients, ordered according to the canonical sequence of the multivariate polynomial expansion

Author(s)

Jordan Ko

References

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulationof stochastic shear flows*, Doctoral thesis, University of Paris VI.

See Also

`getM`, `CreateQuadrature`

Examples

```
rm(list = setdiff(ls(), lsf.str()))

### CASE 1:
### Ishigami function definition: y= 1 + Phi_1(x1)*Phi_1(x2) + Phi_3(x2)
Model <- function(x){
  a <- 3
  b <- 7
  res <- sin(pi*x[1,]) + a*sin(pi*x[2,])^2 + b*(pi*x[3,])^4*sin(pi*x[1,])
  return(res)
}

# determine the exact solutions for the Ishigami function
a <- 3
b <- 7
MeanExact <- a/2
VarExact <- a^2/8 + b*pi^4/5 + b^2*pi^8/18 + 1/2
SobolExact <- c(b*pi^4/5 + b^2*pi^8/50 + 1/2, a^2/8, 0, 0, 8*b^2*pi^8/225, 0, 0)

# random variable definitgeion
d <- 3              # number of random variables
L <- 4              # quadrature level in each dimention.
                    # could be anisotropic eg c(3,4,5) for full quadrature
P <- L-1            # maximum polynomial expansion (cardinal order)
M <- getM(d,P)      # number of PCE termrs
ParamDistrib <- NULL
# ParamDistrib<- list(beta=rep(0.0,d),alpha=rep(0.0,d))

# PCE definition
QuadType <- "FULL"              # type of quadrature
QuadPoly <- rep("LEGENDRE",d)   # polynomial to use
ExpPoly <- rep("LEGENDRE",d)    # polynomial to use

# QuadType <- "SPARSE"                                  # type of quadrature
# QuadPoly <- 'ClenshawCurtis'                          # polynomial to use
# ExpPoly <- rep("LEGENDRE",d)    # polynomial to use

Quadrature = CreateQuadrature(d,L,QuadPoly,ExpPoly,QuadType,ParamDistrib) # quadrature

# function sampling
y <- Model(Quadrature$QuadNodes) # Ishigami function d=3

# generate PCE coefficients
PCE = generatePCEcoeff(M,Quadrature$QuadSize,y,Quadrature$PolyNodes,Quadrature$QuadWeights) # PCE


# getting mean and variance
PCEMean = PCE$PCEcoeff[1]
PCEVar = sum(PCE$PCEcoeff[2:M]^2*PCE$PhiIJ[2:M])
#
# Sobol' sensitivity analysis
Index <- indexCardinal(d,P)
```

```
Sobol = getSobol(d,Index,PCE$PCEcoeff,PCE$PhiIJ)

cat('Mean and variance normalized absolute errors are',
     abs(PCEMean-MeanExact)/MeanExact, 'and', abs(PCEVar-VarExact)/VarExact,'\n')
cat('PCE Sobol indices are ',Sobol$Values,'\n')
cat('Sobol absolte errors are ',abs(Sobol$Values-SobolExact),'\n')
```

---

getM *Get canonical polynomial chaos expansion size*

Description

The getM is a very simple function that determines the size of the canonical polynomial chaos expansion for d random variables up to and including degree p in the expansion. The size is determined from choose(d+p,d) or factorial(d+p)/factorial(d)/factorial(p).

Usage

```
getM(d,p)
```

Arguments

d	Number of input random variables
p	Order of the polynomial chaos expansion

Value

M	Number of terms in the canonical d-dimensional polynomial chaos expansion up to and including degree p

Author(s)

Jordan Ko

References

R. Ghanem and P. Spanos, 1991, *Stochastic Finite Elements: A Spectral Approach*. Berlin: Springer.

Examples

```
d <- 10
p <- 15
m <- getM(d,p)
```

getSobol *Determine Sobol indices from PCE solutions*

Description

Computes the Sobol sensitivity indices from the coefficient of the polynomial chaos expansion.

Usage

```
getSobol(d,Index,Coeff,PhiIJ)
```

Arguments

d	Number of random inputs
Index	A (m x d) matrix that donates the polynomial order for each random variables in the canonical construction of PCE
Coeff	A m-tuple vector containing the PCE coefficients, ordered according to the canonical sequence of the multivariate polynomial expansion
PhiIJ	A m-tuple vector containing the variance of each.

Value

Names	The
Values	

Author(s)

Jordan Ko

References

I. M. Sobol', 1993, *Sensitivity estimate for non-linear mathematical models*. Mathematical modelling and computational experiments 1, 407-414.

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulation of stochastic shear flows*, Doctoral thesis, University of Paris VI.

See Also

getM,CreateQuadrature

Examples

```
### Ishigami function definition
Model <- function(x){
  a <- 3
  b <- 7
  res <- sin(pi*x[1,]) + a*sin(pi*x[2,])^2 + b*(pi*x[3,])^4*sin(pi*x[1,])
  return(res)
}

# Find exact solution for the Ishigami test function
a <- 3
b <- 7
MeanExact <- a/2
VarExact <- a^2/8 + b*pi^4/5 + b^2*pi^8/18 + 1/2
SobolExact <- c(b*pi^4/5 + b^2*pi^8/50 + 1/2, a^2/8, 0, 0, 8*b^2*pi^8/225, 0, 0)

# random variable definition
d <- 3              # number of random variables
L <- 4              # quadrature level in each dimension.
                    # could be anisotropic eg c(3,4,5) for full quadrature
P <- L-1            # maximum polynomial expansion (cardinal order)
M <- getM(d,P)      # number of PCE terms
ParamDistrib <- NULL
# ParamDistrib<- list(beta=rep(0.0,d),alpha=rep(0.0,d))

# PCE definition
QuadType <- "FULL"              # type of quadrature
QuadPoly <- rep("LEGENDRE",d)   # polynomial to use
ExpPoly <- rep("LEGENDRE",d)    # polynomial to use

# QuadType <- "SPARSE"                                  # type of quadrature
# QuadPoly <- 'ClenshawCurtis'                          # polynomial to use
# ExpPoly <- rep("LEGENDRE",d)    # polynomial to use

Quadrature = CreateQuadrature(d,L,QuadPoly,ExpPoly,QuadType,ParamDistrib) # quadrature

# function sampling
y <- Model(Quadrature$QuadNodes) # Ishigami function d=3

# generate PCE coefficients
PCE = generatePCEcoeff(M,Quadrature$QuadSize,y,Quadrature$PolyNodes,Quadrature$QuadWeights) # PCE

# Sobol' sensitivity analysis
Index <- indexCardinal(d,P)
Sobol = getSobol(d,Index,PCE$PCEcoeff,PCE$PhiIJ)

cat('PCE Sobol indices are ',Sobol$Values,'\n')
cat('Sobol absolte errors are ',abs(Sobol$Values-SobolExact),'\n')
```

GPCE.lar *Adaptive sparse generalized polynomial chaos expansion based on least angle regression*

Description

The GPCE.lar function implements a polynomial chaos expansion of a given model or an external model. The strategy for the expansion of the model into a polynomial chaos basis is the adaptive sparse method based on the least angle regression. A statistical and a global sensitivity analysis of the model is then carried out.

Usage

```
GPCE.lar(Model = NULL, PCSpace = "Uniform",
         InputDim = 3, InputDistrib = c(),
         ParamDistrib = NULL, Q2tgt = 1 - 10^(-6),
         Eps = 10^(-2) * (1 - Q2tgt), EpsForw = Eps,
         EpsBack = Eps, EnrichStep = 50,
         jmax= InputDim, pmaxi = 6,
         DesignLength = 8, SeedSob =sample(1:1000, 1))
```

Arguments

Model	a function defining the model to analyze or NULL if the model is external
PCSpace	The space where the expansion is achieved. Options are Gaussian, Uniform and Physic. Physic use the same distributions as the input ones for the expansion
InputDim	Dimension of the input
InputDistrib	Distribution of the input. Options are Gaussian, Uniform, Beta, Gamma
ParamDistrib	Parameters of the input distributions
Q2tgt	Fix the accuracy of the expansion fitting. By default 1-10(^-6)
Eps	Common epsilon for the selection of the basis. By default set to 10^(-2)*(1-Q2tgt)
EpsForw	Epsilon used for the forward selection of the basis. By default set to Eps
EpsBack	Epsilon used for the forward selection of the basis. By default set to Eps
EnrichStep	Number of samples to add to the experimental design. By default set to 50
jmax	The maximum interaction order between the input variables
pmaxi	The maximum degree of the polynomial basis
DesignLength	The length of the input design. By default set to 8
SeedSob	Seed for the Sobol design generation

Value

Designs	A list containing the Sobol design, the input distributions design, the polynomial chaos design and the design length
Output	Vector of the model output
TruncSet	Matrix of the kept sparse polynomial basis. TruncSet_ij is the jth polynomial degree associated to the ith variable
CoeffPCE	Vector of the expansion coefficients associated to the TruncSet. CoeffPCE_j is the jth coefficient associaed to the jth polynomial basis.
R2	The R2 PCE approximation oerror
Q2	The Q2 PCE approximation error
Moments	A list containing the fourth first moments of the output: mean, variance, standard deviation, skweness and kurtosis
Sensitivity	A list containing the sobol sensitivity indices and the sobol total sensitivity indices
OutputDistrib	A list containing a kernel estimation of the output distribution and the associated bandwidth

Author(s)

Munoz Zuniga Miguel

References

G. Blatman and B. Sudret, 2011, *Adapive sparse polynomial chaos expansion based on least angle regression*, Journal of Computational Physics, 230, 2345–2367.

See Also

tell.GPCE.lar

Examples

```
### CASE 1: model is a R function.
### Model definition: y= 1 + Phi_1(x1)*Phi_1(x2)
Model <- function(x){
  PHerm = hermite.he.polynomials(5, normalized=FALSE)
  y=1+unlist(polynomial.values(PHerm[2],x[,1]))*unlist(polynomial.values(PHerm[2],x[,2]))
  return(y)
}

### Run the algorithm with the lar regression method
ResultObject=GPCE.lar(Model=Model, PCSpace="Gaussian", InputDim=3, InputDistrib=rep("Gaussian",3))
names(ResultObject)
###

### CASE 2: external model (for the example the function Model will be used externaly).

### initialized Output
```

```
Output=c()

### Get a first design
ResultObject=GPCE.lar(PCSpace="Gaussian",InputDim=3,InputDistrib=rep("Gaussian",3))
names(ResultObject)

### Calculate the model output for the given design and concatenate the model output results
### into the output vector
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)

### If the expansion has been calculated the function tell return the full expansion
### paramaters, the moments analysis, the sensitivity analysis and the output distribution
### If not the function tell() return an enriched design.
### In the later case the user calculate the output externally and give them
### to the tell function with the previous ResultObject for further calculation.
### See GPCE.sparse documentation for an example.
```

GPCE.quad *Solve polynomial chaos expansion with numerical quadrature*

Description

The GPCE.quad function implements a polynomial chaos expansion of a given model or an external model. The strategy for the expansion of the model into a polynomial chaos basis is the Gauss quadrature method where the Gauss quadrature points are used to estimate the integrales corresponding to the coefficients of the expansion. A statistical and a global sensitivity analysis of the model is then carried out.

Usage

```
GPCE.quad(InputDim,PCSpace,InputDistrib,ParamDistrib,
Model,ModelParam,Output,DesignInput,p,ExpPoly,QuadType,QuadPoly,QuadLevel)
```

Arguments

InputDim	Number of input random variables
PCSpace	To decide if it should be removed after discussion with Miguel..
InputDistrib	Distribution of the input. Options are Gaussian, Uniform, Beta, Gamma
ParamDistrib	Shape parameters of the random variables.
Model	A function defining the model to analyze or NULL if the model is external
ModelParam	Optional parameters for function evaluation.
Output	The results of the model evaluation at the quadrature points.

DesignInput	To decide if it should be removed after discussion with Miguel.
p	The order of the polynomial chaos expansion. order)
ExpPoly	The polynomial used in the expansion. Options are Hermite, Legendre, Jacobi, Laguerre
QuadType	Type of quadrature. Options are SPARSE or FULL
QuadPoly	The type of one-dimensional quadrature rule to be used. For SPARSE, one can use ClenshawCurtis or Fejer. For FULL, one could choose Hermite,Legendre,Jacobi or Laguerre
QuadLevel	Level of quadrature used in the approximation.

Value

Designs	A list containing the quadrature size, n, a vector of the n quadrature weights, and a n * d matrix of the quadrature points
Output	Vector of the model output
PCEcoeff	Matrix of the kept sparse polynomial basis. TruncSet_ij is the jth polynomial degree associated to the ith variable
Moments	A list containing the fourth first moments of the output: mean, variance, standard deviation, skweness and kurtosis
Sensitivity	A list containing the sobol sensitivity indices (Values) and the normalized Sobol nominal and total sensitivity indices (S and ST)

Author(s)

Jordan Ko

References

J. Ko, D. Lucor and P. Sagaut, 2008, *On Sensitivity of Two-Dimensional Spatially Developing Mixing Layer With Respect to Uncertain Inflow Conditions*, Physics of Fluids, 20(7), 07710201-07710220.

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulationof stochastic shear flows*, Doctoral thesis, University of Paris VI.

J. Ko, D. Lucor and P. Sagaut, 2011, *Effects of base flow uncertainty on Couette flow stability*, Computers and Fluids, 43(1), 82-89.

See Also

tell.GPCE.quad

Examples

```
# CASE 1: Validating Legendre PCE with Ishigami function definition
rm(list = setdiff(ls(), lsf.str()))
Model <- function(x){
  a <- 3
```

```
  b <- 7
  res <- sin(pi*x[1,]) + a*sin(pi*x[2,])^2 + b*(pi*x[3,])^4*sin(pi*x[1,])
  return(res)
}

# Determine the exact solution for the Ishigami test function
a <- 3
b <- 7
MeanExact <- a/2
VarExact <- a^2/8 + b*pi^4/5 + b^2*pi^8/18 + 1/2
SobolExact <- c(b*pi^4/5 + b^2*pi^8/50 + 1/2, a^2/8, 0, 0, 8*b^2*pi^8/225, 0, 0)

d = 3
l = 4
ResultObject=GPCE.quad(Model=Model,InputDim=d,PCSpace="Uniform",InputDistrib=rep('Uniform',d),
                  DesignInput=NULL,p=c(l-1),ExpPoly=rep("LEGENDRE",d),QuadType=c("FULL"),
                  QuadPoly=rep("LEGENDRE",d),QuadLevel=c(l),ParamDistrib=NULL,Output=NULL)
names(ResultObject)
cat('Mean and variance normalized absolute errors are',
  abs(ResultObject$Moments$PCEMean-MeanExact)/MeanExact,
  'and', abs(ResultObject$Moments$PCEVar-VarExact)/VarExact,'\n')
cat('PCE Sobol indices are ',ResultObject$Sensitivity$Values,'\n')
cat('Sobol absolte errors are ',abs(ResultObject$Sensitivity$Values-SobolExact),'\n')

# CASE 2: Validating Hermite PCE with orthogonal Hermite functions
### Model is a R function as a sum of multivariate Hermite polynomials

Model <- function(x,param){
    d <- param$d
    p <- param$p
    PCETrue <- param$PCETrue

    n <- dim(x)[2]
    index <- indexCardinal(d,p)
    PHerm <- hermite.he.polynomials(p, normalized=FALSE)
    y <- rep(0,n)

    for (nn in seq(1,n)){
        for (mm in seq(1,getM(d,p))){
            tmp <- 1;
            for (dd in seq(1,d))
            {
            tmp = tmp * unlist(polynomial.values(PHerm[index[dd,mm]+1],x[dd,nn]))
            }
            y[nn] = y[nn] + PCETrue[mm]*tmp
        }
    }
    return(y)
}

## Problem definition
d = 2;          # random dimension
l = 3;          # quadrature level
```

```
p = l - 1;      # polynomial order of expansion
m = getM(d,p);  # size of polynomial expansion

## Model definition
ModelParam <- NULL
ModelParam$d <- d
ModelParam$p <- p
ModelParam$PCETrue <- sample(seq(1,m),m,replace = FALSE)

## CASE 1: The model is directly evaluated from the GPCE.quad function
ResultObject=GPCE.quad(InputDim=d,PCSpace="Normal",InputDistrib=rep('Gaussian',d),
                   DesignInput=NULL,p=c(p),ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                  QuadPoly=rep("HERMITE",d),QuadLevel=c(l),ParamDistrib=NULL,Output=NULL,
                      Model=Model,ModelParam=ModelParam)
cat("The exact PCE coefficients are: \n")
cat(ModelParam$PCETrue,"\n")
cat("The estimated PCE coefficients are: \n")
cat(ResultObject$PCEcoeff,"\n")

## CASE 2: Model is evaluated separately from the GPCE.quad function
# First, the quadrature points are determined from the GPCE.quad function
NoModelResult=GPCE.quad(InputDim=d,PCSpace="Normal",InputDistrib=rep('Gaussian',d),
                   DesignInput=NULL,p=c(p),ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                  QuadPoly=rep("HERMITE",d),QuadLevel=c(l),ParamDistrib=NULL,Output=NULL)
cat("The quadrature points can be determined from the Design variable of the output below: \n")
cat(names(NoModelResult),"\n")

# Second, the model is evalauted at the  quadrature points and stored in Output
Output <- Model(NoModelResult$Design$QuadNodes,ModelParam)

# Third, the model output is passed back to GPCE.quad, along with DesignInput and Output
cat("After Design$QuadNodes are evaluated and stored in Output,
the results is passed back to GPCE.quad:\n")
NoModelResult=GPCE.quad(InputDim=d,PCSpace = "Normal",InputDistrib=rep('Gaussian',d),
                        DesignInput=NoModelResult$Design$QuadNodes,p=c(p),
                        ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                        QuadPoly=rep("HERMITE",d),QuadLevel=c(l),
                        ParamDistrib=NULL,Output=Output)
cat("The exact PCE coefficients are:\n")
cat(ModelParam$PCETrue,"\n")
cat("The estimated PCE coefficients are:\n")
cat(NoModelResult$PCEcoeff,"\n")

### Run the algorithm with the lar regression method
# ResultObject=GPCE.lar(Model=Model, PCSpace="Gaussian", InputDim=d, InputDistrib=rep("Gaussian",d))


# CASE 3: Validating Jacobi PCE with orthogonal Jacobi functions
rm(list = setdiff(ls(), lsf.str()))

# Multivariate Jacobi polynomial test
Model <- function(x){
    a = 2;
```

```
    b = 3;
    res <- 2 +
        3*(2*(a+1)+(a+b+2)*(x[1,]-1))/2 +
        5*(2*(a+1)+(a+b+2)*(x[2,]-1))/2 +
        7*(2*(a+1)+(a+b+2)*(x[3,]-1))/2 +
        11.*(4*(a+1)*(a+2)+4*(a+b+3)*(a+2)*(x[1,]-1)+(a+b+3)*(a+b+4)*(x[1,]-1)^2)/8 +
        13*(2*(a+1)+(a+b+2)*(x[1,]-1))*(2*(a+1)+(a+b+2)*(x[2,]-1))/4 +
        17*(2*(a+1)+(a+b+2)*(x[1,]-1))*(2*(a+1)+(a+b+2)*(x[3,]-1))/4 +
        19.*(4*(a+1)*(a+2)+4*(a+b+3)*(a+2)*(x[2,]-1)+(a+b+3)*(a+b+4)*(x[2,]-1)^2)/8 +
        23*(2*(a+1)+(a+b+2)*(x[2,]-1))*(2*(a+1)+(a+b+2)*(x[3,]-1))/4 +
        29.*(4*(a+1)*(a+2)+4*(a+b+3)*(a+2)*(x[3,]-1)+(a+b+3)*(a+b+4)*(x[3,]-1)^2)/8
    return(res)
}

# x is a random variabls following the beta distribution with the pdf
# f(x,a,b) = (1-x)^a(1+x)^b/2^(a+b+1)/Beta(a+1,b+1), where Beta is the beta function

d = 3
l = 3
ParamDistrib <- NULL
ParamDistrib$alpha <- rep(2,d)  # 1st shape parameters for the random variables
ParamDistrib$beta <- rep(3,d)   # 2nd shape parameters for the random variables

ResultObject=GPCE.quad(Model=Model,InputDim=d,PCSpace="Beta",InputDistrib=rep('Beta',d),
                  DesignInput=NULL,p=c(l-1),ExpPoly=rep("JACOBI",d),QuadType=c("FULL"),
                      QuadPoly=rep("JACOBI",d),QuadLevel=c(l),ParamDistrib,Output=NULL)

cat('Estimated PCE coefficients are', ResultObject$PCEcoeff)

# CASE 4: Validating Laguerre PCE with orthogonal Laguerre functions

rm(list = setdiff(ls(), lsf.str()))

# Multivariate Laguerre polynomial test
Model <- function(x){
    a <- 2
    res <- 2 +
      3*(-x[1,]+a+1) +
      5*(-x[2,]+a+1) +
      7*(-x[3,]+a+1) +
      11*(x[1,]^2/2-(a+2)*x[1,]+(a+2)*(a+1)/2) +
      13*(-x[1,]+a+1)*(-x[2,]+a+1) +
      17*(-x[1,]+a+1)*(-x[3,]+a+1) +
      19*(x[2,]^2/2-(a+2)*x[2,]+(a+2)*(a+1)/2) +
      21*(-x[2,]+a+1)*(-x[3,]+a+1) +
      29*(x[3,]^2/2-(a+2)*x[3,]+(a+2)*(a+1)/2)
    return(res)
}

# x is a random variabls following the gamma distribution with scale parameter theta = 1 and shape
# parameter k = a + 1 giving the pdf f(x,a) = x^(k-1)*exp(-x)/Gamma(k) = x^a*exp(-x)/Gamma(a+1),
# where Gamma is the gamma function
```

```
a = 2
d = 3
l = 3

ParamDistrib = NULL
ParamDistrib$alpha = rep(a,d)

ResultObject=GPCE.quad(Model=Model,InputDim=d,PCSpace="Gamma",InputDistrib=rep('Gamma',d),
                  DesignInput=NULL,p=c(l-1),ExpPoly=rep("LAGUERRE",d),QuadType=c("FULL"),
                    QuadPoly=rep("LAGUERRE",d),QuadLevel=c(l),ParamDistrib,Output=NULL)

cat('Estimated PCE coefficients are', ResultObject$PCEcoeff)
```

GPCE.sparse *Adaptive sparse generalized polynomial chaos expansion*

Description

The GPCE.sparse function implements a polynomial chaos expansion of a given model or an external model. The strategy for the expansion of the model into a polynomial chaos basis is the adaptive sparse method where the sampled design is iteratively enriched until the expansion coefficients estimation, by regression, is accurate. Meanwile a relevant selection of the polynomial basis to keep in the expansion is carried out. A statistical and a global sensitivity analysis of the model is then carried out.

Usage

```
GPCE.sparse(Model = NULL, PCSpace = "Uniform", InputDim = 3, InputDistrib = c(),
                 ParamDistrib = NULL, Q2tgt = 1 - 10^(-6), Eps = 10^(-2) * (1 -
                 Q2tgt), EpsForw = Eps, EpsBack = Eps, EnrichStep = 50, jmax
                 = InputDim, pmaxi = 12, DesignLength = 8, SeedSob =
                 sample(1:1000, 1))
```

Arguments

Model	A function defining the model to analyze or NULL if the model is external
PCSpace	The space where the expansion is achieved. Options are Gaussian, Uniform and Physic. Physic use the same distributions as the input ones for the expansion
InputDim	Dimension of the input
InputDistrib	Distribution of the input. Options are Gaussian, Uniform, Beta, Gamma
ParamDistrib	Parameters of the input distributions
Q2tgt	Fix the accuracy of the expansion fitting. By default 1-10(^-6)

Eps	Common epsilon for the selection of the basis. By default set to 10^(-2)*(1-Q2tgt)
EpsForw	Epsilon used for the forward selection of the basis. By default set to Eps
EpsBack	Epsilon used for the forward selection of the basis. By default set to Eps
EnrichStep	Number of samples to add to the experimental design. By default set to 50
DesignLength	The length of the input design. By default set to 8
jmax	The maximum interaction order between the input variables
pmaxi	The maximum degree of the polynomial basis
SeedSob	Seed for the Sobol design generation

Value

Designs	A list containing the Sobol design, the input distributions design, the polynomial chaos design and the design length
Output	Vector of the model output
TruncSet	Matrix of the kept sparse polynomial basis. TruncSet_ij is the jth polynomial degree associated to the ith variable
CoeffPCE	Vector of the expansion coefficients associated to the TruncSet. CoeffPCE_j is the jth coefficient associaed to the jth polynomial basis.
R2	The R2 PCE approximation error
Q2	The Q2 PCE approximation error
Moments	A list containing the fourth first moments of the output: mean, variance, standard deviation, skweness and kurtosis
Sensitivity	A list containing the sobol sensitivity indices and the sobol total sensitivity indices
OutputDistrib	A list containing a kernel estimation of the output distribution and the associated bandwidth

Author(s)

Munoz Zuniga Miguel

References

G. Blatman and B. Sudret, 2010, *An adapive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis*, Probabilistic Engineering Mechanics, 25, 183–197.

See Also

tell.GPCE.sparse

Examples

```
### CASE 1: model is a R function.
### Model definition: y= 1 + Phi_1(x1)*Phi_1(x2) + Phi_3(x2)
Model <- function(x){
  PHerm = hermite.he.polynomials(7, normalized=FALSE)
  y=1+unlist(polynomial.values(PHerm[2],x[,1]))*unlist(polynomial.values(PHerm[2],x[,2]))+
   unlist(polynomial.values(PHerm[4],x[,2]))/sqrt(factorial(3))
   return(y)
}

### Run the algorithm with the sparse regression method
ResultObject=GPCE.sparse(Model=Model, PCSpace="Gaussian", InputDim=3,
                         InputDistrib=rep("Gaussian",3))
names(ResultObject)
###

### CASE 2: external model (for the example the function Model will be used externaly).

### initialized Output
Output=c()

### Get a first design
ResultObject=GPCE.sparse(PCSpace="Gaussian",InputDim=3,InputDistrib=rep("Gaussian",3))
names(ResultObject)

### Calculate the model output for the given design and concatenate the model output results
### into the output vector
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)

### If the expansion has been calculated the function tell return the full expansion
### paramaters, the moments analysis, the sensitivity analysis and the output distribution
### If not the function tell() return an enriched design.
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)
###
```

`indexCardinal` *Get the order of canonical polynomial chaos expansion*

Description

For a d-dimensional polynomial chaos expansion up to order p, there are a total of M polynomials, determine from get(d,p). Each polynomial $phi_m(x)$ is expressed as the product of the polynomial

from each random variables, i.e. $phi_m(x) = phi_m,1(x_1)phi_m,2(x_2)...phi_m,d(x_d)$, each with a different polynomial order. We can thus very succintly denote phi_m(x) with a n-tuple vector containing the polynomial order from each dimension and the entire canonical PCE expansion can be express as a (m x n) matrix.

Usage

```
indexCardinal(d,p,m,index)
```

Arguments

d	Number of input random variables
p	Order of the polynomial chaos expansion
m	Pointer to the current n-tuple vector being calculated
index	A dummy variable that stores the polynoial order calculated so far

Value

Index	A (m x d) matrix that donates the polynomial order for each random variables in the canonical construction of PCE

Author(s)

Jordan Ko

References

R. Ghanem and P. Spanos, 1991, *Stochastic Finite Elements: A Spectral Approach*. Berlin: Springer.

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulationof stochastic shear flows*, Doctoral thesis, Universit\'e Paris VI.

See Also

getM

Examples

```
d <- 3
p <- 5
m <- getM(d,p)
index <- indexCardinal(d,p)
print(t(index))
```

`plot.GPCE.lar` *Plot results of adaptive sparse generalized polynomial chaos expansion based on least angle regression*

Description

Plot results of adaptive sparse generalized polynomial chaos expansion based on least angle regression

Usage

```
## S3 method for class 'GPCE.lar'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`x`	A list of arguments returned by the `PCE` function
`ylim`	y-coordinate plotting limits
`...`	Supplmeentary arguments passed to nodeplot

See Also

`GPCE.lar`

`plot.GPCE.quad` *Plot results of generalized polynomial chaos expansion based on the quadrature solution*

Description

Plot results of generalized polynomial chaos expansion based on the quadrature solution

Usage

```
## S3 method for class 'GPCE.quad'
plot(x,...)
```

Arguments

`x`	A list of arguments returned by the `PCE` function
`...`	Supplmeentary arguments passed to nodeplot

See Also

`GPCE.quad`

`plot.GPCE.sparse` *Plot results of adaptive sparse generalized polynomial chaos expansion.*

Description

Plot results of adaptive sparse generalized polynomial chaos expansion.

Usage

```
## S3 method for class 'GPCE.sparse'
plot(x, ylim = c(0, 1), ...)
```

Arguments

`x`	A list of arguments returned by the `PCE` function
`ylim`	y-coordinate plotting limits
`...`	Supplementary arguments passed to nodeplot

See Also

`GPCE.sparse`

`polyNorm` .

Description

expansion.

Usage

```
polyNorm(degree,x,polynomType,alpha,beta)
```

Arguments

`degree`	Number of random inputs
`x`	A (m x d) matrix that donates the polynomial order for each random variables in the canonical construction of PCE
`polynomType`	A m-tuple vector containing the PCE coefficients, ordered according to the canonical sequence of the multivariate polynomial expansion
`alpha`	A m-tuple vector containing the variance of each.
`beta`	A m-tuple vector containing the variance of each.

Value

y

Author(s)

Jordan Ko

Examples

```
d <- 3
p <- 2
Index <- indexCardinal(d,p)
```

`predict.GPCE.lar` *predict results of adaptive lar generalized polynomial chaos expansion.*

Description

predict results of adaptive lar generalized polynomial chaos expansion.

Usage

```
## S3 method for class 'GPCE.lar'
predict(object,x,Selection=1:nrow(object$TruncSet),...)
```

Arguments

`object`	A list of arguments returned by the `PCE` function
`x`	A matrix with the points where the polynomial chaos expansion needs to be evaluated
`Selection`	A matrix indicating which polynoms of the expansion will be use for the prediction
`...`	additional parameters

See Also

`GPCE.lar`

`predict.GPCE.quad` *predict results of adaptive quad generalized polynomial chaos expansion.*

Description

predict results of adaptive quad generalized polynomial chaos expansion.

Usage

```
## S3 method for class 'GPCE.quad'
predict(object,n,x,...)
```

Arguments

`object`	A list of arguments returned by the `PCE` function
`n`	Number of points to evaluate
`x`	A matrix with the points where the polynomial chaos expansion needs to be evaluated
`...`	additional parameters

See Also

`GPCE.quad`

`predict.GPCE.sparse` *predict results of adaptive sparse generalized polynomial chaos expansion.*

Description

predict results of adaptive sparse generalized polynomial chaos expansion.

Usage

```
## S3 method for class 'GPCE.sparse'
predict(object,x,Selection=1:nrow(object$TruncSet),...)
```

Arguments

`object`	A list of arguments returned by the `PCE` function
`x`	A matrix with the points where the polynomial chaos expansion needs to be evaluated
`Selection`	A matrix indicating which polynoms of the expansion will be use for the prediction
`...`	additional parameters

See Also

GPCE.sparse

`print.GPCE.lar` *Print results of adaptive lar generalized polynomial chaos expansion based on least angle regression*

Description

Print results of adaptive lar generalized polynomial chaos expansion based on least angle regression

Usage

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

Arguments

x	A list of arguments returned by the PCE function
...	Supplementary arguments passed to print()

See Also

GPCE.lar

`print.GPCE.quad` *Print results of adaptive quad generalized polynomial chaos expansion.*

Description

Print results of adaptive quad generalized polynomial chaos expansion.

Usage

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

Arguments

x	A list of arguments returned by the PCE function
...	Supplementary arguments passed to print()

See Also

GPCE.quad

`print.GPCE.sparse` *Print results of adaptive sparse generalized polynomial chaos expansion.*

Description

Print results of adaptive sparse generalized polynomial chaos expansion.

Usage

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

Arguments

`x`	A list of arguments returned by the `PCE` function
`...`	Supplementary arguments passed to print()

See Also

`GPCE.sparse`

`tell.GPCE.lar` *Adaptive sparse generalized polynomial chaos expansion based on least angle regression*

Description

The function `tell.GPCE.lar` is used for generalized polynomial chaos expansion of external models. When a functional model is given to the function `GPCE.lar`, this latter automatically call the function `tell.GPCE.lar`. When in the function `GPCE.lar` the `Model` option is `NULL` then this latter returns a list of `Designs` and a list of Arguments, `Args`, the ones entered for the previous `GPCE.lar` run. Then the `Designs`, the `Args` and the vector of manually calculated outputs need to be given to the function `tell.GPCE.lar` to run the algorithm.

Usage

```
## S3 method for class 'GPCE.lar'
tell(x,Output,...)
```

Arguments

`x`	ResultObjectDesign2EvalArgs: A list of arguments returned by the `PCE` function
`Output`	The vector of the manually calculated model outputs
`...`	additional parameters

Value

The same as the GPCE.lar function

Author(s)

Munoz Zuniga Miguel

References

G. Blatman and B. Sudret, 2011, *Adapive sparse polynomial chaos expansion based on least angle regression*, Journal of Computational Physics, 230, 2345–2367.

See Also

GPCE.lar

Examples

```
### External model (for the example the function Model defined below will be used externaly)
### Model definition: y= 1 + Phi_1(x1)*Phi_1(x2) + Phi_3(x2)
Model <- function(x){
  PHerm = hermite.he.polynomials(5, normalized=FALSE)
  y=1+unlist(polynomial.values(PHerm[2],x[,1]))*unlist(polynomial.values(PHerm[2],x[,2]))+
   unlist(polynomial.values(PHerm[4],x[,2]))/sqrt(factorial(3))
   return(y)
}

### initialized Output
Output=c()

### Get a first design
ResultObject=GPCE.lar(PCSpace="Gaussian",InputDim=3,InputDistrib=rep("Gaussian",3))
names(ResultObject)

### Calculate the model output for the given design and concatenate the model output results
### into the output vector
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)

### If the expansion has been calculated the function tell return the full expansion
### paramaters, the moments analysis, the sensitivity analysis and the output distribution
### If not the function tell() return an enriched design.
### In the later case the user calculate the output
### externally and give them to the tell function
### with the previous ResultObject for further calculation.
### See GPCE.sparse documentation for an example.
```

`tell.GPCE.quad` *Generalized polynomial chaos expansion based on numerical quadrature*

Description

The function `tell.GPCE.quad` is used for generalized polynomial chaos expansion of external models. When a functional model is given to the function `GPCE.quad`, this latter automatically call the function `tell.GPCE.quad`.

When in the function `GPCE.quad` the `Model` option is `NULL` then this latter returns a list of `Designs` and a list of Arguments, `Args`, the ones entered for the previous `GPCE.quad` run.

Then the `Designs`, the `Args` and the vector of manually calculated outputs need to be given to the function `tell.GPCE.quad` to run the algorithm.

Usage

```
## S3 method for class 'GPCE.quad'
tell(x,Output,...)
```

Arguments

	The same as the `GPCE.quad` function
	ResultObjectDesign2EvalArgs: This is the argument defined by the user.
`Output`	The vector of the manually calculated model outputs
`...`	additional parameters

Value

The same as the `GPCE.quad` function

Author(s)

Jordan Ko

References

J. Ko, D. Lucor and P. Sagaut, 2008, *On Sensitivity of Two-Dimensional Spatially Developing Mixing Layer With Respect to Uncertain Inflow Conditions*, Physics of Fluids, 20(7), 07710201-07710220.

J. Ko, 2009, *Applications of the generalized polynomial chaos to the numerical simulationof stochastic shear flows*, Doctoral thesis, University of Paris VI.

J. Ko, D. Lucor and P. Sagaut, 2011, *Effects of base flow uncertainty on Couette flow stability*, Computers and Fluids, 43(1), 82-89.

See Also

GPCE.quad

Examples

```
### Model is a R function as a sum of multivariate Hermite polynomials

Model <- function(x,param){
    d <- param$d
    p <- param$p
    PCETrue <- param$PCETrue

    n <- dim(x)[2]
    index <- indexCardinal(d,p)
    PHerm <- hermite.he.polynomials(p, normalized=FALSE)
    y <- rep(0,n)

    for (nn in seq(1,n)){
        for (mm in seq(1,getM(d,p))){
            tmp <- 1;
            for (dd in seq(1,d))
            {
            tmp = tmp * unlist(polynomial.values(PHerm[index[dd,mm]+1],x[dd,nn]))
            }
            y[nn] = y[nn] + PCETrue[mm]*tmp
        }
    }
    return(y)
}

## Problem definition
d = 2;          # random dimension
l = 3;          # quadrature level
p = l - 1;      # polynomial order of expansion
m = getM(d,p);  # size of polynomial expansion

## Model definition
ModelParam <- NULL
ModelParam$d <- d
ModelParam$p <- p
ModelParam$PCETrue <- sample(seq(1,m),m,replace = FALSE)

## CASE 1: The model is directly evaluated from the GPCE.quad function
ResultObject=GPCE.quad(InputDim=d,PCSpace="Normal",InputDistrib=rep('Gaussian',d),
                   DesignInput=NULL,p=c(p),ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                  QuadPoly=rep("HERMITE",d),QuadLevel=c(l),ParamDistrib=NULL,Output=NULL,
                      Model=Model,ModelParam=ModelParam)
cat("The exact PCE coefficients are: \n")
cat(ModelParam$PCETrue,"\n")
cat("The estimated PCE coefficients are: \n")
cat(ResultObject$PCEcoeff,"\n")
```

```
## CASE 2: Model is evaluated separately from the GPCE.quad function
# First, the quadrature points are determined from the GPCE.quad function
NoModelResult=GPCE.quad(InputDim=d,PCSpace="Normal",InputDistrib=rep('Gaussian',d),
                    DesignInput=NULL,p=c(p),ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                   QuadPoly=rep("HERMITE",d),QuadLevel=c(l),ParamDistrib=NULL,Output=NULL)
cat("The quadrature points can be determined from the Design variable of the output below: \n")
cat(names(NoModelResult),"\n")

# Second, the model is evalauted at the  quadrature points and stored in Output
Output <- Model(NoModelResult$Design$QuadNodes,ModelParam)

# Third, the model output is passed back to GPCE.quad, along with DesignInput and Output
cat("After Design$QuadNodes are evalauted and stored in Output,
the results is passed back to GPCE.quad:\n")
NoModelResult=GPCE.quad(InputDim=d,PCSpace = "Normal",InputDistrib=rep('Gaussian',d),
                        DesignInput=NoModelResult$Design$QuadNodes,p=c(p),
                        ExpPoly=rep("HERMITE",d),QuadType=c("FULL"),
                        QuadPoly=rep("HERMITE",d),QuadLevel=c(l),
                        ParamDistrib=NULL,Output=Output)
cat("The exact PCE coefficients are:\n")
cat(ModelParam$PCETrue,"\n")
cat("The estimated PCE coefficients are:\n")
cat(NoModelResult$PCEcoeff,"\n")
```

`tell.GPCE.sparse` *Adaptive sparse generalized polynomial chaos expansion*

Description

The function `tell.GPCE.sparse` is used for generalized polynomial chaos expansion of external models. When a functional model is given to the function `GPCE.sparse`, this latter automatically call the function `tell.GPCE.sparse`. When in the function `GPCE.sparse` the `Model` option is `NULL` then this latter returns a list of `Designs` and a list of Arguments, `Args`, the ones entered for previous the `GPCE.sparse` run. Then the `Designs`, the `Args` and the vector of manually calculated outputs need to be given to the function `tell.GPCE.sparse` to run the algorithm.

Usage

```
## S3 method for class 'GPCE.sparse'
tell(x,Output,...)
```

Arguments

`x`	ResultObjectDesign2EvalArgs: a list of arguments returned by the `GPCE.sparse` function
`Output`	the vector of the manually calculated model outputs
`...`	additional parameters

Value

The same as the GPCE.sparse function

Author(s)

Munoz Zuniga Miguel

References

G. Blatman and B. Sudret, 2010, *An adapive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis*, Probabilistic Engineering Mechanics, 25, 183–197.

See Also

GPCE.sparse

Examples

```
### External model (for the example the function Model defined below will be used externaly)
### Model definition: y= 1 + Phi_1(x1)*Phi_1(x2) + Phi_3(x2)
Model <- function(x){
  PHerm = hermite.he.polynomials(5, normalized=FALSE)
  y=1+unlist(polynomial.values(PHerm[2],x[,1]))*unlist(polynomial.values(PHerm[2],x[,2]))+
  unlist(polynomial.values(PHerm[4],x[,2]))/sqrt(factorial(3))
  return(y)
}

### initialized Output
Output=c()

### Get a first design
ResultObject=GPCE.sparse(PCSpace="Gaussian",InputDim=3,InputDistrib=rep("Gaussian",3))
names(ResultObject)

### Calculate the model output for the given design and concatenate the model output results
### into the output vector
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)

### If the expansion has been calculated the function tell return the full expansion
### paramaters, the moments analysis, the sensitivity analysis and the output distribution
### If not the function tell() return an enriched design.
Output=c(Output,Model(ResultObject$Design2Eval))

### Give the design and the calculated ouput to the tell function
ResultObject=tell(ResultObject,Output)
names(ResultObject)
###
```

Index