Package 'CaliCo'

July 24, 2018

Type Package

Title Code Calibration in a Bayesian Framework

Version 0.1.1

Date 2018-07-23

Description Calibration of every computational code. It uses a Bayesian framework to rule the estimation. With a new data set, the prediction will create a prevision set taking into account the new calibrated parameters. The choices between several models is also available. The methods are described in the paper Carmassi et al. (2018) <arXiv:1801.01810>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Imports R6, ggplot2, DiceKriging, DiceDesign, MASS, coda, parallel, gridExtra, gtools

RoxygenNote 6.0.1

Collate 'functions.R' 'models.R' 'forecast.R' 'prior.R' 'RcppExports.R' 'CaliCo-package.R' 'Kernel.R' 'calibration.R' 'sequentialDesign.R'

LinkingTo Rcpp, RcppArmadillo, Matrix

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Mathieu Carmassi [aut, cre]

Maintainer Mathieu Carmassi <mathieu.carmassi@gmail.com>

Repository CRAN

Date/Publication 2018-07-24 09:00:20 UTC

R topics documented:

`calibrate` *Generates* `calibrate.class` *objects*

Description

`calibrate` is a function that allows to generate a `calibrate.class` class in which the estimation is done for a defined `model.class` and `prior.class` objects.

Usage

```
calibrate(md, pr, opt.estim, opt.valid = NULL)
```

Arguments

`md`	a `model.class` object
`pr`	a `prior.class` object
`opt.estim`	estimation options • Ngibbs Number of iteration of the algorithm Metropolis within Gibbs • Nmh Number of iteration of the Metropolis Hastings algorithm • thetaInit Initial point • r regulation percentage in the modification of the k in the Metropolis Hastings

- sig Covariance matrix for the proposition distribution ($k * sig$)
- Nchains Number of MCMC chains to run (if Nchain>1 an output is created called mcmc which is a coda object codamenu)
- burnIn Number of iteration to withdraw

`opt.valid` list of cross validation options (default value opt.valid=NULL)

- nCV Number of iterations for the cross validation
- type.valid Type of cross validation selected. "loo" (leave one out) is the only method emplemented so far.

Value

`calibrate` returns a `calibrate.class` object. Two main methods are available:

- plot(mdfit, x, graph) displays a series of graphs (ACF, MCMC, density a priori vs a posteriori , correlation between parameters, results on the quantify of interest, etc..) or return a list with all the graphs:
  - mdfit The calibrated model (a `calibrate.class` object)
  - x The x-axis
  - graph Allows to select the wanted display. By default all the layout pannel graphs are displayed and `graph="all"`. If `graph="chains"`, only the layout of the autocorrelation, chains points and densities a priori and a posteriori is produced. If `graph="corr"`, only the layout of the correlation graph between each parameter is displayed. If `graph="result"`, only the result on the quantity of interest is given. If `graph=NULL`, no graphs are produced automatically.
- print(mdfit) returns the main information concerning the `calibrate.class` object

Author(s)

M. Carmassi

See Also

`prior`, `calibrate`, `forecast`, `sequentialDesign`

Examples

```
## Not run:
####################### The code to calibrate
X <- cbind(seq(0,1,length.out=10),seq(0,1,length.out=10))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(10,0,0.1)

############### For the first model
###### Definition of the model
md <- model(code,X,Yexp,"model1")
###### Definition of the prior densities
```

```
pr <- prior(type.prior=c("gaussian","gaussian","gaussian","gamma"),opt.prior=
list(c(1,0.01),c(1,0.01),c(11,3),c(2,0.1)))
###### Definition of the calibration options
opt.estim=list(Ngibbs=200,Nmh=400,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=1,burnIn=100)
###### Run the calibration
mdfit <- calibrate(md,pr,opt.estim)
####### The plot generated is a list of ggplot
p <- plot(mdfit,X[,1])
p$out
print(mdfit)

## End(Not run)
```

`calibrate.class` *A Reference Class to generates different* `calibrate.class` *objects*

Description

See the function `calibrate` which produces an instance of this class This class comes with a set of methods, some of them being useful for the user: See the documentation for `calibrate`... Other methods should not be called as they are designed to be used during the calibration process.

Fields should not be changed or manipulated by the user as they are updated internally during the estimation process.

Usage

```
calibrate.class
```

Format

An object of class `R6ClassGenerator` of length 24.

Fields

`md` a `model.class` object generated by `model`

`pr` a `prior.class` object generated by `prior`

`opt.estim` list of estimation options (see `calibrate` for details)

`opt.valid` list of cross validation options (see `calibrate` for details)

`logPost` the log posterior

`mcmc` a coda variable of the generated chains

`output` list of several chains and acceptation ratios

`onlyCV` activates only the cross validation

`errorCV` output of the CV errors

`n.cores` number cores available

`binf` the lower bound of the parameters for the DOE

`bsup` the upper bound of the parameters for the DOE

CaliCo *Bayesian calibration for computational codes*

Description

The CaliCo package provides five main functions: `model`, `prior`, `calibrate`, `forecast` and `sequentialDesign`.

Details

Package: CaliCo

Type: Package

Version: 0.1.1

Date: 2018-04-13

License: GPL-2 | GPL-3

Author(s)

Mathieu Carmassi

Maintainer: <mathieu.carmassi@gmail.com>

References

Bachoc, F., Blois, G., Garnier, J., and Martinez, J.-M. (2014). Calibration and improved prediction of computer models by universal kriging. Computational Statistics and Data Analysis, pages 81–97

Bayarri, M., Berger, J., Sacks, P. R., Cafeo, J. A., Cavendish, J., Lin, C. H., and Tu, J. (2007 b). A framework for validation of computer models. Technometrics.

Carmassi, M., Barbillon ,P., Chiodetti, M., Keller, M., Parent, E. (2018). Bayesian calibration of a numerical code for prediction, arXiv preprint arXiv:1801.01810.

Cox, D., Park, J. S., and Singer, C. (2001). A statistical method for tuning a computer code to a data base. Computational Statistics and Data Analysis.

Damblin, G. (2015). Contributions statistiques au calage et à la validation des codes de calculs. PhD thesis, University Paris-Saclay

Hastings, W. K. (1970). Mont carlo sampling methods using markov chains and their applications. Biometrika.

Higdon, D., Kennedy, M. C., Cavendish, J., Cafeo, J., and Ryne, R. (2004). Combining field data and computer simulations for calibration and prediction. SIAM Journal on Scientific Computing.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. Journal of the Royal Statistical Society, serie B, Methodological.

Kennedy, M. C. and O'Hagan, A. (2001b). Supplementary details on bayesian calibration of computer models. Journal of the Royal Statistical Society, serie B, Methodological.

Liu, F., Bayarri, S., and Berger, J. (2009). Modularization in bayesian analysis, with emphasis on analysis of computer models. Bayesian Analysis, pages 119–150.

Robert, C. (1996). Méthodes de monte carlo par chaines de markov. economica.

Roustant, O., Ginsbourger, D., and Devills, Y. (2012). Dicekriging, diceoptim : Two r packages for the analysis of computer experiments by kriging-based metamodeling and optimization. Journal of Statistical Software.

Sacks, J., Welch, W. J., and Toby J. Mitchell, H. P. W. (1989). Design and analysis of computer experiments. Statistical science, pages 409–423.

Santner, T., Williams, B., and Notz, W. (2003). The Design and Analysis of Computer Experiments. Springer-Verlab.

Examples

```
# Introduction to CaliCo
## Not run: vignette("CaliCo-introduction")
```

`chain` *Return the MCMC chain in a* `data.frame`

Description

`chain` is a function that returns a `data.frame` of calibration run without the burn-in

Usage

```
chain(modelfit, coda = TRUE)
```

Arguments

`modelfit`	a `calibrate.class` object
`coda`	if TRUE returns a coda object (if Nchains in opt.estim is higher than 1 a coda object is automatically returned see `codamenu`)

Value

return a `data.frame` or a coda object of the MCMC chain(s) generated.

Author(s)

M. Carmassi

See Also

`model`, `prior`, `calibrate`, `sequentialDesign`

Examples

```
## Not run:
###################### The code to calibrate
X <- cbind(seq(0,1,length.out=10),seq(0,1,length.out=10))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(10,0,0.1)

############### For the first model
###### Definition of the model
md <- model(code,X,Yexp,"model1")
###### Definition of the prior densities
pr <- prior(type.prior=c("gaussian","gaussian","gaussian","gamma"),opt.prior=
list(c(1,0.01),c(1,0.01),c(11,3),c(2,0.1)))
###### Definition of the calibration options
opt.estim=list(Ngibbs=200,Nmh=600,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=1,burnIn=100)
###### Run the calibration
mdfit <- calibrate(md,pr,opt.estim)

mcmc <- chain(mdfit)
### Check coda object
is.mcmc(mcmc)
### get all the chain
mcmc <- chain(mdfit,coda=FALSE)
head(mcmc)

### Multi chains
opt.estim=list(Ngibbs=200,Nmh=600,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=2,burnIn=100)
###### Run the calibration
mdfit2 <- calibrate(md,pr,opt.estim)

mcmc <- chain(mdfit2)
is.mcmc.list(mcmc)

## End(Not run)
```

`DefPos` *Function that deals with negative eigen values in a matrix not positive definite*

Description

Function that deals with negative eigen values in a matrix not positive definite

Usage

```
DefPos(X)
```

Arguments

X	the matrix or the vector

Value

the new matrix X

`estimators` *Return Maximum A Posteriori (MAP) and Mean A Posteriori estimation of a calibration*

Description

`estimators` is a function that returns a list of two elements which are the MAP and the Mean A Posteriori

Usage

```
estimators(modelfit)
```

Arguments

`modelfit`	a `calibrate.class` object

Value

return a `list`:

- MAP The Maximum A Posteriori
- MEAN The Mean A Posteriori

Author(s)

M. Carmassi

See Also

`model`, `prior`, `calibrate`, `sequentialDesign`

Examples

```
## Not run:
###################### The code to calibrate
X <- cbind(seq(0,1,length.out=10),seq(0,1,length.out=10))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(10,0,0.1)

############### For the first model
###### Definition of the model
md <- model(code,X,Yexp,"model1")
###### Definition of the prior densities
pr <- prior(type.prior=c("gaussian","gaussian","gaussian","gamma"),opt.prior=
list(c(1,0.01),c(1,0.01),c(11,3),c(2,0.1)))
###### Definition of the calibration options
opt.estim=list(Ngibbs=200,Nmh=600,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=1,burnIn=100)
###### Run the calibration
mdfit <- calibrate(md,pr,opt.estim)
estimators(mdfit)

## End(Not run)
```

`forecast` *Generates a forecast base on calibration run with* `calibrate`

Description

`forecast` is a function that allows to generate a new `model.class` in which the prediction is done with the Maximum A Posteriori

Usage

```
forecast(modelfit, x.new)
```

Arguments

`modelfit`	a `calibrate.class` object
`x.new`	newdata for the prediction

Details

Note that all the methods for a `model.class` object are availble. Be careful with the `x` in the plot function. It needs to be the x-axis of calibrated data and predicted data.

Value

return a model.class (see model.class for more details)

Author(s)

M. Carmassi

See Also

model, prior, calibrate, sequentialDesign

Examples

```
## Not run:
###################### The code to calibrate
X <- cbind(seq(0,1,length.out=10),seq(0,1,length.out=10))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(10,0,0.1)

############### For the first model
###### Definition of the model
md <- model(code,X,Yexp,"model1")
###### Definition of the prior densities
pr <- prior(type.prior=c("gaussian","gaussian","gaussian","gamma"),opt.prior=
list(c(1,0.01),c(1,0.01),c(11,3),c(2,0.1)))
###### Definition of the calibration options
opt.estim=list(Ngibbs=200,Nmh=600,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=1,burnIn=100)
###### Run the calibration
mdfit <- calibrate(md,pr,opt.estim)
###### Prediction between 1 and 1.2
X.new <- cbind(seq(1,1.2,length.out=10),seq(1,1.2,length.out=10))
fr <- forecast(mdfit,X.new)
print(fr)
plot(fr,c(X[,1],X.new[,1]))

## End(Not run)
```

---

forecast.class *A Reference Class to generates differents* forecast.class *objects*

Description

See the function forecast which produces an instance of this class This class comes with a set of methods, some of them being useful for the user: See the documentation for forecast... Other methods should not be called as they are designed to be used during the calibration process.

Fields should not be changed or manipulated by the user as they are updated internally during the estimation process.

Usage

```
forecast.class
```

Format

An object of class `R6ClassGenerator` of length 24.

Fields

`modelfit` a `calibrate.class` object generated by `calibrate`

`md.new` the new model created with the MAP estimator

`x.new` a new data set (according to the data set used for calibration)

`Kernel.class` *A Reference Class to generates differents model objects*

Description

See the function `model` which produces an instance of this class

Usage

```
Kernel.class
```

Format

An object of class `R6ClassGenerator` of length 24.

Fields

`code` a function which takes in entry X and theta

`X` the matrix of the forced variables

`var` the variance for the covariance function

`psi` the scale vector of correlation length

`n` number of experiments

`Kernel.type` the chosen form of covariance

`Cov` the covariance

kernel.fun *Generates covariances matrices thanks to* Kernel.class

Description

Kernel.fun is a function that allows us to generate covariances matrices from data

Usage

```
kernel.fun(X, var, psi, kernel.type = "gauss")
```

Arguments

X	data
var	the variance for the covariance function
psi	the parameter vector
kernel.type	the choice of the form of the kernel (with d chosen as an euclidian distance)

- gauss
$$\sigma^2 exp{-1/2(d/\psi)^2}$$
- exp
$$\sigma^2 exp{-1/2d/\psi}$$
- matern3_2
$$\sigma^2(1+\sqrt{3}d^2/\psi)exp{-\sqrt{3}d^2/\psi}$$
- matern5_2
$$\sigma^2(1+\sqrt{5}d^2/\psi+5d^2/(3\psi^2))exp{-\sqrt{5}d^2/\psi}$$

Value

Kernel.fun returns a covariance matrix

Author(s)

M. Carmassi

See Also

model.class, prior.class

Examples

```
## Not run:
X <- cbind(seq(0,10,length.out=10),seq(8,20,length.out=10))
var <- 2
psi <- 0.1
Cov <- kernel.fun(X,var,psi,kernel.type="matern5_2")

## End(Not run)
```

`MetropolisHastingsCpp` *C++ implementation of the algorithm for parameter calibration (without discrepancy)*

Description

Run a Metropolis Hastings within Gibbs algorithm and a Metropolis Hastings algorithm with the covariance matrix estimated on the the sample set generated in the Metropolis within Gibbs. This algorithm is suitable only for models without discrepancy.

Usage

```
MetropolisHastingsCpp(Ngibbs, Nmh, theta_init, r, SIGMA, Yf, binf, bsup,
  LogTest, stream)
```

Arguments

`Ngibbs`	the number of iteration in the Metropolis within Gibbs
`Nmh`	the number of iteration in the Metropolis Hastings
`theta_init`	the starting point
`r`	regulation percentage in the modification of the k in the Metropolis Hastings
`SIGMA`	the covariance of the proposition distribution
`Yf`	the vector of recorded data
`binf`	the lower bound of the parameters to calibrate
`bsup`	the upper bound of the parameters to calibrate
`LogTest`	the log posterior density distribution
`stream`	(default=1) if stream=0 the progress bar is disabled

Value

list of outputs:

- PHIwg the points of the Metropolis within Gibbs algorithm in the transformed space
- PHI the points of the Metropolis Hastings algorithm in the transformed space
- THETAwg the points of the Metropolis within Gibbs algorithm in the real space
- THETA the points of the Metropolis Hastings algorithm in the real space
- AcceptationRatioWg the vector of the acceptance ratio for each parameter in the Metropolis within Gibbs
- AcceptationRatio the acceptance ratio in the Metropolis Hastings
- S the covariance computed after the Metropolis within Gibbs
- LikeliWG the likelihood computed at each iteration of the Metropolis within Gibbs algorithm
- Likeli the likelihood computed at each iteration of the Metropolis Hastings algorithm

`MetropolisHastingsCppD` *C++ implementation of the algorithm for parameter calibration (with discrepancy)*

Description

Run a Metropolis Hastings within Gibbs algorithm and a Metropolis Hastings algorithm with the covariance matrix estimated on the the sample set generated in the Metropolis within Gibbs. This algorithm is suitable only for models with discrepancy.

Usage

```
MetropolisHastingsCppD(Ngibbs, Nmh, theta_init, r, SIGMA, Yf, binf, bsup,
  LogTest, stream)
```

Arguments

`Ngibbs`	the number of iteration in the Metropolis within Gibbs
`Nmh`	the number of iteration in the Metropolis Hastings
`theta_init`	the starting point
`r`	regulation percentage in the modification of the k
`SIGMA`	the covariance of the proposition distribution
`Yf`	the vector of recorded data
`binf`	the lower bound of the parameters to calibrate
`bsup`	the upper bound of the parameters to calibrate
`LogTest`	the log posterior density distribution
`stream`	(default=1) if stream=0 the progress bar is disabled

Value

list of outputs:

- PHIwg the points of the Metropolis within Gibbs algorithm in the transformed space
- PHI the points of the Metropolis Hastings algorithm in the transformed space
- THETAwg the points of the Metropolis within Gibbs algorithm in the real space
- THETA the points of the Metropolis Hastings algorithm in the real space
- AcceptationRatioWg the vector of the acceptance ratio for each parameter in the Metropolis within Gibbs
- AcceptationRatio the acceptance ratio in the Metropolis Hastings
- S the covariance computed after the Metropolis within Gibbs
- LikeliWG the likelihood computed at each iteration of the Metropolis within Gibbs algorithm
- Likeli the likelihood computed at each iteration of the Metropolis Hastings algorithm

`model` *Generates* `model.class` *objects.*

Description

`model` is a function that generates a calibration model and the associated likelihood.

Usage

```
model(code, X, Yexp, model = "model1", ...)
```

Arguments

`code`	the computational code (function of X and theta)
`X`	the matrix of the forced variables
`Yexp`	the vector of the experiments
`model`	string of the model chosen ("model1","model2","model3","model4") by default "model1" is chosen. See details for further clarifications
`...`	additional options (see details section)

Details

The different statistical models are:

- Model1:
$$foriin[1,...,n]Yexp_i = f(x_i,\Theta) + \epsilon(x_i)$$
- Model2:
$$foriin[1,...,n]Yexp_i = F(x_i,\Theta) + \epsilon(x_i)$$
- Model3:
$$foriin[1,...,n]Yexp_i = f(x_i,\Theta) + \delta(x_i) + \epsilon(x_i)$$
- Model4:
$$foriin[1,...,n]Yexp_i = F(x_i,\Theta) + \delta(x_i) + \epsilon(x_i)$$

where $foriin[1,\ldots,n]\epsilon(x_i)\ N(0,\sigma^2)$, $F(.,.)\ PG(m_1(.,.),c_1(.,.),(.,.))$ and $\delta(.)\ PG(m_2(.),c_2(.,.))$. There is four kind of models in calibration. They are properly defined in [1].

To establish a Gaussian process three options are available:

- **opt.gp** is an option list containing the parameters to establish the surrogate (only for model2 and model4).
  - **type** type of the chosen kernel (value by default "matern5_2") from `km` function
  - **DOE** design of experiments for the surrogate (default value NULL). If NULL the DOE is automatically generated with the **opt.emul** option.
- **opt.emul** is an option list containing characteristics about emulation option (only for model2 and model4).

  - **p** the number of parameter in the model (default value 1)
  - **n.emul** the number of points for contituing the Design Of Experiments (DOE) (default value 100)
  - **binf** the lower bound of the parameter vector (default value 0)
  - **bsup** the upper bound of the parameter vector (default value 1)
- **opt.sim** is an option list containing the design and corresponding outputs of the code, in the case where no numerical code is available (only for model2 and model4).
  - **Ysim** Output of the code
  - **DOEsim** DOE corresponding to the output of the code

To add a discrepancy in the model, the option opt.disc must be added:

- **opt.disc** is an option list containing characteristics on the discrepancy (only for model3 and model4)
  - **kernel.type** see `kernel.fun` for further details

Value

`model` returns a `model.class` object. This class contains two main methods:

- plot(model,x) this method generates the plot for a new Θ, Θ_D (for model3 and model4), σ^2. The parameter values need to be added to the model with the pipe `%<%`. The argument `x` represents the x-axis and have to be specified to produce a plot.
- print(model) this method presents several pieces of information about the model.

Author(s)

M. Carmassi

References

[1] CARMASSI, Mathieu, BARBILLON, Pierre, KELLER, Merlin, et al. Bayesian calibration of a numerical code for prediction. arXiv preprint arXiv:1801.01810, 2018.

See Also

`prior`, `calibrate`, `forecast`, `sequentialDesign`

Examples

```
## Not run:
###### The code to calibrate
X <- cbind(seq(0,1,length.out=5),seq(0,1,length.out=5))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(5,0,0.1)

###### For the first model
```

```
### Generate the model
model1 <- model(code,X,Yexp,"model1")
### Plot the results with the first column of X
model1 %<% list(theta=c(1,1,11),var=0.01)
plot(model1,X[,1],CI="err")

### Summury of the foo class generated
print(model1)

###### For the second model
### code function is available, no DOE generated upstream
binf <- c(0.9,0.9,10.5)
bsup <- c(1.1,1.1,11.5)
opt.gp <- list(type="matern5_2", DOE=NULL)
opt.emul <- list(p=3,n.emul=150,binf=binf,bsup=bsup,type="maximinLHS")
model2 <- model(code,X,Yexp,"model2",
                opt.gp=opt.gp,
                opt.emul=opt.emul)
model2 %<% list(theta=c(1,1,11),var=0.1)
### Plot the model
plot(model2,X[,1])

### code function is available and use a specific DOE
DOE <- DiceDesign::lhsDesign(20,5)$design
DOE[,3:5] <- unscale(DOE[,3:5],binf,bsup)

opt.gp <- list(type="matern5_2", DOE=DOE)
model2 <- model(code,X,Yexp,"model2",
                opt.gp=opt.gp)
model2 %<% list(theta=c(1,1,11),var=0.1)
plot(model2,X[,1])

### no code function but DOE and code output available
Ysim <- matrix(nr=20,nc=1)
for (i in 1:20)
{
  covariates <- as.matrix(DOE[i,1:2])
  dim(covariates) <- c(1,2)
  Ysim[i] <- code(covariates,DOE[i,3:5])
}

opt.sim <- list(Ysim=Ysim,DOEsim=DOE)
opt.gp <- list(type="matern5_2", DOE=NULL)
model2 <- model(code=NULL,X,Yexp,"model2",
                opt.gp=opt.gp,
                opt.sim=opt.sim)
model2 %<% list(theta=c(1,1,11),var=0.1)
plot(model2,X[,1])

###### For the third model
model3 <- model(code,X,Yexp,"model3",opt.disc=list(kernel.type="gauss"))
model3 %<% list(theta=c(1,1,11),thetaD=c(20,0.5),var=0.1)
plot(model3,X[,1],CI="err")
```

```
print(model3)


## End(Not run)
```

`model.class` *A Reference Class to generates differents model objects*

Description

See the function `model` which produces an instance of this class This class comes with a set of methods, some of them being useful for the user: See the documentation for `model`... Other methods should not be called as they are designed to be used during the calibration process.

Fields should not be changed or manipulated by the user as they are updated internally during the estimation process.

Usage

```
model.class
```

Format

An object of class `R6ClassGenerator` of length 24.

Fields

`code` a function which takes in entry X and theta

`X` the matrix of the forced variables

`Yexp` the experimental output

`n` the number of experiments

`d` the number of forced variables

`binf` the lower bound of the parameters for the DOE

`bsup` the upper bound of the parameters for the DOE

`opt.gp` a list of parameter for the surrogate (default NULL)

- **type** type of the chosen kernel (value by default "matern5_2") from `km` function
- **DOE** design of experiments for the surrogate (default value NULL)

`opt.emul` a list of parameter to establish the DOE (default NULL)

- **p** the number of parameter in the model (default value 1)
- **n.emul** the number of points for constituting the Design Of Experiments (DOE) (default value 100)
- **binf** the lower bound of the parameter vector (default value 0)
- **bsup** the upper bound of the parameter vector (default value 1)

`opt.sim` a list of parameter containing output of the code and corresponding DOE

- **Ysim** Output of the code
- **DOEsim** DOE corresponding to the output of the code

`model` the model choice (see `model` for more specification).

`opt.disc` a list of parameter for the discrepancy

- **kernel.type** the kernel chosen for the Gaussian process

`multivariate` *Simulate from a Multivariate Normal Distribution*

Description

The matrix decomposition is done via eigen; although a Choleski decomposition might be faster, the eigen decomposition is stabler.

Usage

```
multivariate(n = 1, mu, Sigma, tol = 1e-06, empirical = FALSE,
  EISPACK = FALSE)
```

Arguments

`n`	the number of samples required.
`mu`	a vector giving the means of the variables.
`Sigma`	a positive-definite symmetric matrix specifying the covariance matrix of the variables.
`tol`	tolerance (relative to largest variance) for numerical lack of positive-definiteness in Sigma.
`empirical`	logical. If true, mu and Sigma specify the empirical not population mean and covariance matrix.
`EISPACK`	logical: values other than FALSE are an error.

Value

If n = 1 a vector of the same length as mu, otherwise an n by length(mu) matrix with one sample in each row.

prior *Generates* prior.class *objects.*

Description

prior is a function that generates a prior.class containing information about one or several priors. When several priors are selected, the function prior returns a list of prior.class.

Usage

```
prior(type.prior, opt.prior)
```

Arguments

type.prior	the vector of the prior types selected. For example type.prior=c("gaussian","gamma")
opt.prior	list of the hyperparameters relatives to the prior selected. If the first prior selected is Gaussian, the hyperparameters would be the mean and the standard deviation. See Details for further clarifications.

Details

The densities implemented are defined as follow

- The Gaussian density:

$$f(x) = 1/(\sigma * \sqrt(2\pi))exp{-}1/2 * ((x-\mu)/\sigma)^2$$

  where μ and σ (the mean and the standard deviation) are the two hyperparameters. The vector $c(\mu, \sigma^2)$ is the one looked for in opt.prior.

- The Gamma density:

$$f(x) = 1/(k^a * \Gamma(a)) * x^(a-1) * exp(-(x/k))$$

  where a and k (the shape and the scale) are the two hyperparameters. The vector $c(a, k)$ is the one looked for in opt.prior.

- The Uniform density:

$$f(x) = 1/(b-a)$$

  where a and b (the upper and the lower bound) are the two hyperparameters. The vector $c(a, b)$ is the one looked for in opt.prior.

Value

prior returns a prior.class object. Two main methods are available:

- plot() display the probability density of the prior
- print() returns the main information concerning the prior distribution

Author(s)

M. Carmassi

See Also

model, calibrate, forecast, sequentialDesign

Examples

```
## Not run:
#### Only one prior is wanted
## For a Gaussian Prior
gaussian <- prior(type.prior="gaussian",opt.prior=list(c(0.5,0.001)))
plot(gaussian)

#### For several priors
priors <- prior(type.prior=c("gaussian","gamma"),opt.prior=list(c(0.5,0.001),c(5,1)))
plot(priors$Prior1)
plot(priors$Prior2)

## End(Not run)
```

---

prior.class *A Reference Class to generate differents* prior.class *objects*

Description

See the function prior which produces an instance of this class This class comes with a set of methods, some of them being useful for the user: See the documentation prior. Other methods should not be called as they are designed to be used during the optimization process.

Fields should not be changed or manipulated by the user as they are updated internally during the estimation process.

Usage

```
prior.class
```

Format

An object of class R6ClassGenerator of length 24.

Fields

type.prior of the selected prior

opt.prior the characteristics of the selected prior

seqDesign.class *A Reference Class to generate a better Model2 or Model4* model.class *objects*

Description

This class generates a new model.class for Model4 and Model2. Based on the previous estimation of the Gaussian process in the function model, the design of experiments previously used is improved according to [Damblin et al. 2018]. The aim is to reduce the error produced by the initial estimation of the Gaussian process by fortifying the initial DOE. The method consists in proposing new points based on the expectancy improvement criterion.

Fields should not be changed or manipulated by the user as they are updated internally during the estimation process.

Usage

```
seqDesign.class
```

Format

An object of class R6ClassGenerator of length 24.

Fields

doe.init the initial DOE used to fit the first Gaussian process

GP.init the initial Gaussian process generated in model function

GP.new the new Gaussian process fortified with the new design points

p the number of parameter

md the initial model

md.new the new model

mdfit the initial calibrated model

mdfit.new the new calibrated model

X the data set

m minimum of the sum of squares used in the algorithm

References

DAMBLIN, Guillaume, BARBILLON, Pierre, KELLER, Merlin, et al. Adaptive numerical designs for the calibration of computer codes. SIAM/ASA Journal on Uncertainty Quantification, 2018, vol. 6, no 1, p. 151-179.

`sequentialDesign` *Calibration with a sequential design*

Description

The aim is to reduce the error produced by the initial estimation of the Gaussian process by fortifying the initial DOE. The method consists in proposing new points based on the expectancy improvement criterion. The method and the algorithm are detailed in [Damblin et al. 2018]

Usage

```
sequentialDesign(md, pr, opt.estim, k)
```

Arguments

`md`	the model to improve (model2 or model4)
`pr`	list of priors to use for calibration
`opt.estim`	estimation options • NgibbsNumber of iteration of the algorithm Metropolis within Gibbs • Nmh Number of iteration of the Metropolis Hastings algorithm • thetaInit Initial point • r regulation percentage in the modification of the k in the Metropolis Hastings • sig Covariance matrix for the proposition distribution ($k * sig$) • Nchains Number of MCMC chains to run (if Nchain>1 an output is created called mcmc which is a coda object `codamenu`) • burnIn Number of iteration to withdraw
`k`	number of iteration in the algorithm

Value

a `seqDesign.class`

References

DAMBLIN, Guillaume, BARBILLON, Pierre, KELLER, Merlin, et al. Adaptive numerical designs for the calibration of computer codes. SIAM/ASA Journal on Uncertainty Quantification, 2018, vol. 6, no 1, p. 151-179.

See Also

`model`, `prior`, `calibrate`, `sequentialDesign`

Examples

```
## Not run:
###### The code to calibrate
X <- cbind(seq(0,1,length.out=5),seq(0,1,length.out=5))
code <- function(X,theta)
{
  return((6*X[,1]*theta[2]-2)^2*theta[1]*sin(theta[3]*X[,2]-4))
}
Yexp <- code(X,c(1,1,11))+rnorm(5,0,0.1)

###### For the second model
### code function is available, no DOE generated upstream
binf <- c(0.9,0.9,10.5)
bsup <- c(1.1,1.1,11.5)
opt.gp <- list(type="matern5_2", DOE=NULL)
opt.emul <- list(p=3,n.emul=150,binf=binf,bsup=bsup,type="maximinLHS")
model2 <- model(code,X,Yexp,"model2",
                opt.gp=opt.gp,
                opt.emul=opt.emul)
model2 %<% list(theta=c(1,1,11),var=0.1)

pr <- prior(type.prior=c("gaussian","gaussian","gaussian","gamma"),opt.prior=
list(c(1,0.01),c(1,0.01),c(11,3),c(2,0.1)))
###### Definition of the calibration options
opt.estim=list(Ngibbs=200,Nmh=400,thetaInit=c(1,1,11,0.1),r=c(0.3,0.3),
sig=diag(4),Nchains=1,burnIn=100)
###### Run the sequential calibration
mdfit <- sequentialDesign(model2,pr,opt.estim,2)
#plot(mdfit,X[,1])

## End(Not run)
```

`unscale` *Function which unscale un matrix or a vector*

Description

Function which unscale un matrix or a vector

Usage

```
unscale(M, binf, bsup, diag = FALSE, sym = FALSE)
```

Arguments

`M`	the matrix or the vector
`binf`	the lower bound
`bsup`	the upper bound
`diag`	default value False if we want to unscale the whole matrix
`sym`	default value False if we do not have a symmetric matrix

Value

the unscaled vector or matrix

Examples

```
## Not run:
X <- diag(3)*runif(3)
Y <- unscale(X,rep(10,3),rep(15,3))
print(Y)

## End(Not run)
```

`unscale.matrix.diag` *Function which unscale only the diagonal component of a matrix*

Description

Function which unscale only the diagonal component of a matrix

Usage

```
unscale.matrix.diag(M, binf, bsup)
```

Arguments

`M`	the matrix
`binf`	the lower bound
`bsup`	the upper bound

Value

the normalized diagonal

Examples

```
## Not run:
X <- diag(3)*runif(3)
Y <-unscale.matrix.diag(X,rep(10,3),rep(15,3))
print(Y)

## End(Not run)
```

unscale.vector *Function which unscale a vector between two bounds*

Description

Function which unscale a vector between two bounds

Usage

```
unscale.vector(x, binf, bsup)
```

Arguments

x	the vector to unscale
binf	the lower bound
bsup	the upper bound

Value

y the vector unscaled

Examples

```
## Not run:
X <- runif(3)
Y <-unscale.vector(X,rep(10,3),rep(15,3))
print(Y)

## End(Not run)
```

%<% *Operator to define active bindings variables*

Description

Operator to define active bindings variables

Usage

```
md %<% param
```

Arguments

md	a statistical model defined by the function model
param	a list of parameter values

Value

a `model.class` parametrized.

Index