Package: cmce (via r-universe)

September 15, 2024

Type Package

Title Computer Model Calibration for Deterministic and Stochastic Simulators

Version 0.1.0

Date 2018-05-12

Author Matthew T. Pratola <mpratola@stat.osu.edu> [aut, cre, cph]

Maintainer Matthew T. Pratola <mpratola@stat.osu.edu>

Description Implements the Bayesian calibration model described in Pratola and Chkrebtii (2018) <DOI:10.5705/ss.202016.0403> for stochastic and deterministic simulators. Additive and multiplicative discrepancy models are currently supported. See <http://www.matthewpratola.com/software> for more information and examples.

Depends R (>= 3.2.3)

Imports stats

License AGPL-3

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2018-05-24 12:31:59 UTC

Contents

cmce-package — cmce *is an open-source R package implementing Bayesian calibration for deterministic and stochastic simulation models.*

Description

cmce implements a Bayesian calibration model based on a dimension-reduction approach via empirical orthogonal functions. The model generalizes the popular SVD-based approach (e.g. Higdon et al. (2008)) to a tensor-variate decomposition, enabling the calibration of both deterministic simulators and stochastic simulators. Currently, cmce is a proof of concept implementation based entirely on R code, leaving open the possibility of future performance improvements if the codebase is moved to C at a later point. It supports both the popular additive discrepancy as well as multiplicative discrepancy. The model is fit using a Metropolis-within-Gibbs MCMC algorithm.

Details

The main model fitting function in the package is calibrate(). This function will return posterior samples of the calibration parameters as well as the predicted field process, predicted discrepancies and predicted calibrated simulator. The model currently does not support predicting off the input grid and assumes that field data and simulator outputs are observed on the same input grid.

Author(s)

Matthew T. Pratola <mpratola@stat.osu.edu> [aut, cre, cph]

References

Pratola, Matthew T. and Chrebtii, Oksana. (2018) Bayesian Calibration of Multistate Stochastic Simulators. *Statistica Sinica*, **28**, 693–720. doi: 10.5705/ss.202016.0403.

Higdon, Dave, Gattiker, James, Williams, Brian and Rightley, Maria. (2008) Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, **103**, 570–583. doi: 10.1198/016214507000000888.

See Also

calibrate

calibrate — *Fit the calibration model.*

Description

calibrate() runs the MCMC algorithm to calibrate simulator outputs to field observations while estimating unknown settings of calibration parameters and quantifying model discrepancy. Currently, two forms of discrepancy are supported: additve stationary Gaussian Process discrepancy and scalar multiplicative discrepancy (with a Normal prior).

Usage

```
calibrate(yf, phi, N, pinfo, mh, last = 1000)
```

Arguments

yf	A vector of field observations.
phi	A matrix or list of matrices representing simulator outputs
N	The number of MCMC iterations to run.
pinfo	A list of prior parameter settings. See details below.
mh	A list of settings for Metropolis-Hastings proposals. See details below.
last	The number of MCMC steps to report (default is 1000). The first (N-last) steps are discarded as burn-in.

Details

The method can calibrate both stochastic simulator outputs and deterministic simulator outputs, and samples from the posterior distribution of unknowns conditional on the observed field data and simulator outputs. The method makes use of empirical orthogonal functions to reduce the dimension of the data to make computations feasible. In addition, there is support for multi-state simulators, and calibration can be performed when all states or only a subset of states are observed in the field.

For more details on the model, see Pratola and Chkrebtii (2018). Detailed examples demonstrating the method are available at http://www.matthewpratola.com/software.

Value

A list with last samples drawn from the posterior.

References

Pratola, Matthew T. and Chrebtii, Oksana. (2018) Bayesian Calibration of Multistate Stochastic Simulators. *Statistica Sinica*, **28**, 693–720. doi: 10.5705/ss.202016.0403.

See Also

cmce-package getranges scaledesign unscalemat makedistlist gp.realize

Examples

```
library(cmce)

set.seed(7)

# n is the number of observation locations on the spatial-temporal grid
# m is the number of simulation model runs at parameter settings theta_1,...,theta_m
# ns is the number of simulator states.
# sd.field is the std. deviation of the iid normal measurement error for the field data yf.
n=20
m=5
```

```
ns=1
sd.field=0.1

# Just a 1D example:
x=seq(0,1,length=n)

# the nxm model output matrix:
phi=matrix(0,nrow=n,ncol=m+1)
calisettings=matrix(runif(m+1)*5,ncol=1)
r=getranges(calisettings)
k=ncol(calisettings)

# Our "unknown" theta on original scale and transformed to [0,1]:
theta.orig=2.1
calisettings[m+1]=theta.orig
design=scaledesign(calisettings,r)
theta=design[m+1]

# Generate reality - bang!
X=expand.grid(x,design)
l.gen=makedistlist(X)
rho=c(0.2,0.01)
muv=rep(0,(m+1)*n)
lambdav=1/2
surface=gp.realize(l.gen,muv,lambdav,rho)


# phi matrix
phi=matrix(surface,ncol=m+1,nrow=n)

## Not run:
# When phi is a matrix as above, the code performs deterministic calibration.
# To perform calibration for a stochastic simulator with, say, M available realizations,
# replace phi with a list of matrices:
phi=vector("list",M)
for(i in 1:M) phi[[i]]=matrix(realization[[i]],ncol=m+1,nrow=n)

## End(Not run)



# Do some plots:
plot(x,phi[,1],pch=20,ylim=c(-4,4),xlab="x",ylab="response")
for(i in 2:(m+1)) points(x,phi[,i],pch=20)
points(x,phi[,m+1],col="green",pch=20)


# setup
nc=2 # dimension reduction by only retaining nc components of the svd decomposition.
     # Must have nc>=2.
th.init=rep(0,k)
# matrix with all the calibration parameter settings and the last row will be filled
# in with the estimate of theta during MCMC:
```

```
design.w=matrix(c(design[1:m],th.init),ncol=1)


# These matrices are (m+1)x(m+1).  The upper-left mxm matrix is the one used for
# fitting gp's to the weights V.
l.v=makedistlist(design.w)


# we have the true theta stored, we no longer need it in calisettings
calisettings=calisettings[1:m,,drop=FALSE]


# Calibration parameter priors
thetaprior=NULL # use default uniform priors automatically constructed in cal.r


# Fake field data:
yf=phi[,m+1]+rnorm(n,sd=sd.field)


# For additive discrepancy:
l.d=makedistlist(x)
q=1
p=1
inidelta=rep(0,n)


# Specify Normal priors for the multiplicative discrepancies
inikap1=1
lamkap1=Inf


# State indices, since we only have 1 state this is trivial
is1=1:n


# setup pinfo and mh:
pinfo=list(l.v=l.v,l.d=l.d,n=n,m=m,q=k,p=p,nc=nc,ranges=r,thetaprior=thetaprior,
      ns=ns,six=list(is1=is1),inidelta=inidelta,
      lambdav=list(a=rep(10,nc),b=rep(0.1,nc)),
      lambdad=list(a=c(10),b=c(0.1)),
      mukap=c(inikap1),
      lambdakap=c(lamkap1),
      lambdaf=list(a=c(10),b=c(.5)),
      rho=list(a=5,b=1),
      psis=list(a=rep(2,p),b=rep(10,p)),
      delta.corrmodel="gaussian", eps=1e-10)
mh=list(rr=0.1, rp=0.1, rth=0.2)



# Run
N=2000   # Number of iterations, first 50% are used for adaptation
```

```
last=499 # Save these last draws as samples.  The N*50%-last are discarded burn-in.
fit=calibrate(yf,phi,N,pinfo,mh,last=last)



# Plot result
par(mfrow=c(1,2),pty="s")
# plot theta's
plot(density(unscalemat(fit$theta,r)),xlim=c(0,5),cex.lab=1.2,cex.axis=0.8,
    xlab=expression(theta),main="")
abline(v=theta.orig,lwd=2,lty=3)
# Next, the response and discrepancy:
plot(x,yf,col="pink",pch=20,xlim=c(0,1),ylim=c(-3,2),cex.lab=1.2,cex.axis=0.8,main="",
    ylab="response")
ix=seq(1,last,length=100)
for(i in 1:m) lines(x,phi[,i],col="grey",lwd=2)
for(i in ix) lines(x,fit$delta[i,],col="green")
for(i in ix) lines(x,fit$phi[[i]][,m+1])
for(i in ix) lines(x,fit$phi[[i]][,m+1]+fit$delta[i,],col="red",lty=3)
points(x,yf,col="pink",pch=20)
abline(h=0,lty=3)
legend(0.0,2,cex=.5,lwd=c(1,1,1,1),legend=c("emulator","discrepancy","predictive",
    "outputs"),col=c("black","green","red","grey"))
```

`getranges` *Get variable ranges from a design matrix.*

Description

`getranges()` is a helper function to get the lower/upper bounds of variables in a design matrix, used for rescaling the inputs to the `[0,1])` hypercube.

Usage

```
getranges(design)
```

Arguments

`design`	An `n` x `p` matrix of input settings

Value

A `p` x `2` matrix with the lower and upper bounds (rounded to nearest integer value) of all `p` variables in the design matrix.

Examples

```
library(cmce)

design=matrix(runif(10,1,5),ncol=2,nrow=5)
getranges(design)
```

---

gp.realize *Draw an unconditional GP realization.*

Description

The list `l.v` is created by `makedistlist()` from a n x p design matrix on which to generate the unconditional GP realization. The correlation parameters take on values in `(0,1)` and use the Gaussian correlation model, calculated as `rho^d(xi,xj)^2`. This helper function is used to create the demonstration data used in the example for `calibrate()`.

Usage

```
gp.realize(l.v,mu,lambda,rhos,lambdaf=Inf,eps=1e-10,from="")
```

Arguments

`l.v`	A list of difference matrices for the design as calculated by `makedistlist()`
`mu`	An n x 1 mean vector for the realization
`lambda`	A scalar quantity for the marginal precision of the drawn realization
`rhos`	A p x 1 vector of correlation parameters
`lambdaf`	A scalar quantity denoting the precision of the error (i.i.d.) component of the realization
`eps`	A fudge factor to help with inverting large correlation matrices
`from`	Internal use only

Value

An n x 1 vector of the GP realization calculated over the finite locations of the original design matrix.

See Also

`getranges` `scaledesign` `makedistlist`

Examples

```
library(cmce)

design=matrix(runif(10,1,5),ncol=2,nrow=5)
r=getranges(design)
design=scaledesign(design,r)
l.v=makedistlist(design)
rho=c(0.2,0.01)
muv=rep(0,nrow(design))
lambdav=1
surface=gp.realize(l.v,muv,lambdav,rho)
```

`makedistlist` *Make list of distance matrices for calculating GP correlation matrices.*

Description

`makedistlist()` is a helper function used to setup the difference matrices that are used in `calibrate()`'s separable GP model.

Usage

```
makedistlist(design)
```

Arguments

`design`	An `n` x `p` matrix of input settings

Value

A list of `p` matrices, each of dimension `n` x `n` that contain the outer subtractions of each variable in the design matrix.

See Also

`getranges` `scaledesign` `gp.realize`

Examples

```
library(cmce)

design=matrix(runif(10,1,5),ncol=2,nrow=5)
r=getranges(design)
design=scaledesign(design,r)
l.v=makedistlist(design)
```

scaledesign *Rescale a design matrix to the [0,1] hypercube.*

Description

scaledesign() is a helper function to rescale a design to the [0,1]) hypercube using variable ranges previously extracted by a call to getranges().

Usage

```
scaledesign(design,r)
```

Arguments

design	An n x p matrix of input settings
r	An p x 2 matrix of variable ranges extracted from getranges()

Value

A n x p design matrix rescaled to the [0,1] hypercube.

See Also

unscalemat

Examples

```
library(cmce)

design=matrix(runif(10,1,5),ncol=2,nrow=5)
r=getranges(design)
scaledesign(design,r)
```

unscalemat *Unscale a matrix back to its original ranges.*

Description

unscalemat() is a helper function to rescale a matrix back to its original ranges. Typically this is used to rescale the posterior samples of the parameters back to their original scale.

Usage

```
unscalemat(mat,r)
```

Arguments

mat	An n x p matrix of numbers scaled to the [0,1] hypercube
r	An p x 2 matrix of the original ranges of the variables

Value

A n x p matrix with variables rescaled back to their original ranges, as specified by ranges.

See Also

getranges scaledesign

Examples

```
library(cmce)

design=matrix(runif(10,1,5),ncol=2,nrow=5)
r=getranges(design)
design=scaledesign(design,r)
unscalemat(design,r)
```

Index