Package: emulator (via r-universe)

Type Package

Title Bayesian Emulation of Computer Programs

Version 1.2-24

VignetteBuilder knitr

Depends R (>= 3.0.1), mvtnorm

Suggests knitr,rmarkdown

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Description Allows one to estimate the output of a computer program,
as a function of the input parameters, without actually running
it. The computer program is assumed to be a Gaussian process,
whose parameters are estimated using Bayesian techniques that
give a PDF of expected program output. This PDF is conditional
on a training set of runs, each consisting of a point in
parameter space and the model output at that point. The
emphasis is on complex codes that take weeks or months to run,
and that have a large number of undetermined input parameters;
many climate prediction models fall into this class. The
emulator essentially determines Bayesian posterior estimates of
the PDF of the output of a model, conditioned on results from
previous runs and a user-specified prior linear model. The
package includes functionality to evaluate quadratic forms
efficiently.

License GPL

URL https://github.com/RobinHankin/emulator

BugReports https://github.com/RobinHankin/emulator/issues

Repository https://robinhankin.r-universe.dev

RemoteUrl https://github.com/robinhankin/emulator

RemoteRef HEAD

RemoteSha eaf0ebae961d392d30e5eda41ae35100e3f799cd

Contents

---

emulator-package   *Bayesian Emulation of Computer Programs*

Description

Allows one to estimate the output of a computer program, as a function of the input parameters, without actually running it. The computer program is assumed to be a Gaussian process, whose parameters are estimated using Bayesian techniques that give a PDF of expected program output. This PDF is conditional on a training set of runs, each consisting of a point in parameter space and the model output at that point. The emphasis is on complex codes that take weeks or months to run, and that have a large number of undetermined input parameters; many climate prediction models fall into this class. The emulator essentially determines Bayesian posterior estimates of the PDF of the output of a model, conditioned on results from previous runs and a user-specified prior linear model. The package includes functionality to evaluate quadratic forms efficiently.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

References

- J. Oakley 1999. "Bayesian uncertainty analysis for complex computer codes", PhD thesis, University of Sheffield.

- R. K. S. Hankin 2005. "Introducing BACCO, an R bundle for Bayesian analysis of computer code output", Journal of Statistical Software, 14(16)

Examples

```
## More detail given in optimal.scales.Rd
scales_true  <- c(1,1,1,1,1,4)

## and a real (linear) relation:
real.relation <- function(x){sum( (1:6)*x )}

## Now a design matrix:
val  <- latin.hypercube(100,6)

## apply the real relation:
d <- apply(val,1,real.relation)

## and add some suitably correlated Gaussian noise:
A <- corr.matrix(val,scales=scales_true)
d.noisy <-  as.vector(rmvnorm(n=1,mean=apply(val,1,real.relation), 0.3*A))

## Now try to predict the values at points x:

x <- latin.hypercube(20,6)
predicted <- int.qq(x,d.noisy,xold=val, Ainv=solve(A),pos.def.matrix=diag(scales_true))
observed <- apply(x,1,real.relation)

par(pty='s')
plot(predicted,observed,xlim=c(4,18),ylim=c(4,18))
abline(0,1)
```

---

betahat.fun	*Calculates MLE coefficients of linear fit*

Description

Determines the maximum likelihood regression coeffients for the specified regression basis and correlation matrix A.

The ".A" form needs only A (and not Ainv), thus removing the need to calculate a matrix inverse. Note that this form is *slower* than the other if Ainv is known in advance, as solve(.,.) is slow.

If Ainv is not known in advance, the two forms seem to perform similarly in the cases considered here and in the goldstein package.

Usage

```
betahat.fun(xold, Ainv, d, give.variance=FALSE, func)
betahat.fun.A(xold, A, d, give.variance=FALSE, func)
```

Arguments

xold	Data frame, each line being the parameters of one run
A	Correlation matrix, typically provided by corr.matrix()
Ainv	Inverse of the correlation matrix A
d	Vector of results at the points specified in xold
give.variance	Boolean, with TRUE meaning to return information on the variance of $\hat{\beta}$ and default FALSE meaning to return just the estimator
func	Function to generate regression basis; defaults to regressor.basis

Note

Here, the strategy of using two separate functions, eg foo() and foo.A(), one of which inverts A and one of which uses notionally more efficient means. Compare the other strategy in which a Boolean flag, use.Ainv, has the same effect. An example would be scales.likelihood().

Author(s)

Robin K. S. Hankin

References

- J. Oakley and A. O'Hagan, 2002. *Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs*, Biometrika 89(4), pp769-784
- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

Examples

```
data(toy)
val <- toy
H <- regressor.multi(val)
d <- apply(H,1,function(x){sum((0:6)*x)})


fish <- rep(2,6)
A <- corr.matrix(val,scales=fish)
Ainv <- solve(A)
```

```
# now add suitably correlated Gaussian noise:
d <-  as.vector(rmvnorm(n=1,mean=d, 0.1*A))

betahat.fun(val , Ainv , d)            # should be close to c(0,1:6)


# Now look at the variances:
betahat.fun(val,Ainv,give.variance=TRUE, d)


    # now find the value of the prior expectation (ie the regression
    # plane) at an unknown point:
x.unknown <- rep(0.5 , 6)
regressor.basis(x.unknown) %*% betahat.fun(val, Ainv, d)

    # compare the prior with the posterior
interpolant(x.unknown, d, val, Ainv,scales=fish)
    # Heh, it's the same!  (of course it is, there is no error here!)


    # OK, put some error on the old observations:
d.noisy <- as.vector(rmvnorm(n=1,mean=d,0.1*A))

    # now compute the regression point:
regressor.basis(x.unknown) %*% betahat.fun(val, Ainv, d.noisy)

    # and compare with the output of interpolant():
interpolant(x.unknown, d.noisy, val, Ainv, scales=fish)
    # there is a difference!



    # now try a basis function that has superfluous degrees of freedom.
    # we need a bigger dataset.  Try 100:
val <- latin.hypercube(100,6)
colnames(val) <- letters[1:6]
d <- apply(val,1,function(x){sum((1:6)*x)})
A <- corr.matrix(val,scales=rep(1,6))
Ainv <- solve(A)


betahat.fun(val, Ainv, d, func=function(x){c(1,x,x^2)})
    # should be c(0:6 ,rep(0,6).  The zeroes should be zero exactly
    # because the original function didn't include any squares.


## And finally a sanity check:
f <- function(x){c(1,x,x^2)}
jj1 <- betahat.fun(val, Ainv, d, func=f)
jj2 <- betahat.fun.A(val, A, d, func=f)

abs(jj1-jj2)  # should be small
```

---

corr   *correlation function for calculating A*

Description

Calculates the correlation function between two points in parameter space, thus determining the correlation matrix A.

Usage

```
corr(x1, x2, scales=NULL , pos.def.matrix=NULL,
coords="cartesian", spherical.distance.function=NULL)
corr.matrix(xold, yold=NULL, method=1, distance.function=corr, ...)
```

Arguments

x1	First point
x2	Second point
scales	Vector specifying the diagonal elements of B (see below)
pos.def.matrix	Positive definite matrix to be used by corr.matrix() for B. Exactly one of scales and pos.definite.matrix should be specified. Supplying scales specifies the diagonal elements of B (off diagonal elements are set to zero); supply pos.definite.matrix in the general case. A single value is recycled. Note that neither corr() nor corr.matrix() test for positive definiteness
coords	In function corr(), a character string, with default "cartesian" meaning to interpret the elements of x1 (and x2) as coordinates in Cartesian space. The only other acceptable value is currently "spherical", which means to interpret the first element of x1 as row number, and the second element as column number, on a spherical computational grid (such as used by climate model Goldstein; see package goldstein for an example of this option in use). Spherical geometry is then used to calculate the geotetic (great circle) distance between point x1 and x2, with function gcd()
method	An integer with values 1, 2, or 3. If 1, then use a fast matrix calculation that returns $e^{-(x-x')^T B(x-x')}$. If 2 or 3, return the appropriate output from corr(), noting that ellipsis arguments are passed to corr() (for example, scales). The difference between 2 and 3 is a marginal difference in numerical efficiency; the main difference is computational elegance. **Warning 1:** The code for method=2 (formerly the default), has a bug. If yold has only one row, then corr.matrix(xold,yold,scales,method=2) returns the transpose of what one would expect. Methods 1 and 3 return the correct matrix. **Warning 2:** If argument distance.function is not the default, and method is the default (ie 1), then method will be silently changed to 2 on the grounds that method=1 is meaningless unless the distance function is corr()

`distance.function`

> Function to be used to calculate distances in `corr.matrix()`. Defaults to `corr()`

`xold` Matrix, each row of which is an evaluated point

`yold` (optional) matrix, each row of which is an evaluated point. If missing, use `xold`

`spherical.distance.function`

> In `corr`, a function to determine the distance between two points; used if `coords`="spherical". A good one to choose is `gcd()` (that is, Great Circle Distance) of the goldstein library

`...` In function `corr.matrix()`, extra arguments that are passed on to the distance function. In the default case in which the distance.function is `corr()`, one *must* pass `scales`

Details

Function `corr()` calculates the correlation between two points x1 and x2 in the parameter space. Function `corr.matrix()` calculates the correlation matrix between each row of `xold` and `yold`. If `yold=NULL` then the correlation matrix between `xold` and itself is returned, which should be positive definite.

Evaluates Oakley's equation 2.12 for the correlation between $\eta(x)$ and $\eta(x')$: $e^{-(x-x')^T B(x-x')}$.

Value

Returns the correlation function

Note

It is worth reemphasising that supplying `scales` makes matrix B diagonal.

Thus, if `scales` is supplied, `B=diag(scales)` and

$$c(x, x') = \exp\left[-(x - x')^T B(x - x')\right] = \exp\left[\Sigma_i s_i(x_i - x'_i)^2\right]$$

Thus if x has units $[X]$, the units of `scales` are $[X^{-2}]$.

So if `scales[i]` is big, even small displacements in `x[i]` (that is, moving a small distance in parameter space, in the i-th dimension) will result in small correlations. If `scales[i]` is small, even large displacements in `x[1]` will have large correlations

Author(s)

Robin K. S. Hankin

References

- J. Oakley 1999. *Bayesian uncertainty analysis for complex computer codes*, PhD thesis, University of Sheffield.

- J. Oakley and A. O'Hagan, 2002. *Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs*, Biometrika 89(4), pp769-784

- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

Examples

```
jj <- latin.hypercube(2,10)
x1 <- jj[1,]
x2 <- jj[2,]

corr(x1,x2,scales=rep(1,10))            # correlation between 2 points
corr(x1,x2,pos.def.matrix=0.1+diag(10)) # see effect of offdiagonal elements

x <- latin.hypercube(4,7)               # 4 points in 7-dimensional space
rownames(x) <- letters[1:4]             # name the points

corr.matrix(x,scales=rep(1,7))

x[1,1] <- 100                           # make the first point far away
corr.matrix(x,scales=rep(1,7))

# note that all the first row and first column apart from element [1,1]
# is zero (or very nearly so) because the first point is now very far
# from the other points and has zero correlation with them.

# To use just a single dimension, remember to use the drop=FALSE argument:
corr.matrix(x[,1,drop=FALSE],scales=rep(1,1))


# For problems in 1D, coerce the independent variable to a matrix:
m <- c(0.2, 0.4, 0.403, 0.9)
corr.matrix(cbind(m),scales=1)


# now use a non-default value for distance.function.
# Function f() below taken from Jeremy Oakley's thesis page 12,
# equation 2.10:

f <- function(x,y,theta){
  d <- sum(abs(x-y))
  if(d >= theta){
    return(0)
  }else{
    return(1-d/theta)
  }
}


corr.matrix(xold=x, distance.function=f, method=2, theta=4)

 # Note the first row and first column is a single 1 and 3 zeros
 # (because the first point, viz x[1,], is "far" from the other points).
 # Also note the method=2 argument here; method=1 is the fast slick
 # matrix method suggested by Doug and Jeremy, but this only works
 # for distance.function=corr.
```

---

estimator   *Estimates each known datapoint using the others as datapoints*

Description

Uses Bayesian techniques to estimate a model's prediction at each of n datapoints. To estimate the ith point, conditioning variables of $1, \ldots, i-1$ and $i+1, \ldots, n$ inclusive are used (ie, all points except point i).

This routine is useful when finding optimal coefficients for the correlation using boot methods.

Usage

```
estimator(val, A, d, scales=NULL, pos.def.matrix=NULL,
func=regressor.basis)
```

Arguments

val	Design matrix with rows corresponding to points at which the function is known
A	Correlation matrix (note that this is **not** the inverse of the correlation matrix)
d	Vector of observations
scales	Scales to be used to calculate t(x). Note that scales has no default value because estimator() is most often used in the context of assessing the appropriateness of a given value of scales. If the desired distance matrix (called B in Oakley) is not diagonal, pass this matrix to estimator() via the pos.def.matrix argument.
pos.def.matrix	Positive definite matrix B
func	Function used to determine basis vectors, defaulting to regressor.basis if not given.

Details

Given a matrix of observation points and a vector of observations, estimator() returns a vector of predictions. Each prediction is made in a three step process. For each index i:

- Observation d[i] is discarded, and row i and column i deleted from A (giving A[-i,-i]). Thus d and A are the observation vector and correlation matrix that would have been obtained had observation i not been available.
- The value of d[i] is estimated on the basis of the shortened observation vector and the comatrix of A.

It is then possible to make a scatterplot of d vs dhat where dhat=estimator(val,A,d). If the scales used are "good", then the points of this scatterplot will be close to abline(0,1). The third step is to optimize the goodness of fit of this scatterplot.

Value

A vector of observations of the same length as d.

Author(s)

Robin K. S. Hankin

References

- J. Oakley and A. O'Hagan, 2002. *Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs*, Biometrika 89(4), pp769-784

- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

See Also

optimal.scales

Examples

```
# example has 11 observations on 6 dimensions.
# function is just sum( (1:6)*x) where x=c(x_1, ... , x_2)

val <- latin.hypercube(11,6)
colnames(val) <- letters[1:6]
d <- apply(val,1,function(x){sum((1:6)*x)})

#pick some scales:
fish <- rep(1,ncol(val))
A <- corr.matrix(val,scales=fish)

#add some suitably correlated noise:
d <- as.vector(rmvnorm(n=1, mean=d, 0.1*A))

# estimate d using the leave-out-one technique in estimator():
d.est <- estimator(val, A, d, scales=fish)

#and plot the result:
lims <- range(c(d,d.est))
par(pty="s")
plot(d, d.est, xaxs="r", yaxs="r", xlim=lims, ylim=lims)
abline(0,1)
```

---

expert.estimates   *Expert estimates for Goldstein input parameters*

Description

A dataframe consisting of expert judgements of low, best, and high values for each of 19 variables that are used in the creation of the QWERTYgoin.* files by makeinputfiles().

Usage

```
data(expert.estimates)
```

Format

A data frame with 19 observations on the following 3 variables.

low a numeric vector: low estimate

best a numeric vector: best estimate

high a numeric vector: high estimate

Details

The rows correspond to the column names of results.table.

Examples

```
data(expert.estimates)
```

---

interpolant   *Interpolates between known points using Bayesian estimation*

Description

Calculates the posterior distribution of results at a point using the techniques outlined by Oakley. Function interpolant() is the primary function of the package. Function interpolant.quick() gives the expectation of the emulator at a set of points, and function interpolant() gives the expectation and other information (such as the variance) at a single point. Function int.qq() gives a quick-quick vectorized interpolant using certain timesaving assumptions.

Usage

```
interpolant(x, d, xold, Ainv=NULL, A=NULL, use.Ainv=TRUE,
      scales=NULL, pos.def.matrix=NULL, func=regressor.basis,
      give.full.list = FALSE, distance.function=corr, ...)
interpolant.quick(x, d, xold, Ainv=NULL, scales=NULL,
pos.def.matrix=NULL, func=regressor.basis, give.Z = FALSE,
distance.function=corr, ...)
int.qq(x, d, xold, Ainv, pos.def.matrix, func=regressor.basis)
```

Arguments

x	Point(s) at which estimation is desired. For `interpolant.quick()`, argument `x` is a matrix and an expectation is given for each row
d	vector of observations, one for each row of `xold`
xold	Matrix with rows corresponding to points at which the function is known
A	Correlation matrix `A`. If not given, it is calculated
Ainv	Inverse of correlation matrix `A`. Required by `int.qq()`. In `interpolant()` and `interpolant.quick()` using the default value of `NULL` results in `Ainv` being calculated explicitly (which may be slow: see next argument for more details)
use.Ainv	Boolean, with default `TRUE` meaning to use the inverse matrix `Ainv` (and, if necessary, calculate it using `solve(.)`). This requires the not inconsiderable overhead of inverting a matrix. If, however, `Ainv` is available, using the default option is *much* faster than setting `use.Ainv=FALSE`; see below.
	If `FALSE`, function `interpolant()` does not use `Ainv`, but makes extensive use of `solve(A,x)`, mostly in the form of `quad.form.inv()` calls. This option avoids the overhead of inverting a matrix, but has non-negligible marginal costs.
	If `Ainv` is not available, there is little to choose, in terms of execution time, between calculating it explicitly (that is, setting `use.Ainv=TRUE`), and using `solve(A,x)` (ie `use.Ainv=TRUE`).
	Note: if `Ainv` is given to the function, but `use.Ainv` is FALSE, the code will do as requested and use the slow `solve(A,x)`, which is probably not what you want
func	Function used to determine basis vectors, defaulting to `regressor.basis` if not given
give.full.list	In `interpolant()`, Boolean variable with `TRUE` meaning to return the whole list of posterior parameters as detailed on pp12-15 of Oakley, and default `FALSE` meaning to return just the best estimate
scales	Vector of "roughness" lengths used to calculate `t(x)`, the correlations between `x` and the points in the design matrix `xold`.
	Note that `scales` is needed twice overall: once to calculate `Ainv`, and once to calculate `t(x)` inside `interpolant()` (`t(x)` is determined by calling `corr()` inside an `apply()` loop). A good place to start might be `scales=rep(1,ncol(xold))`.
	It's probably worth restating here that the elements of `scales` correspond to the diagonal elements of the B matrix (see `?corr`) and so have the dimensions of $[D]^{-2}$ where D is the dimensions of `xold`
pos.def.matrix	A positive definite matrix that is used if `scales` is not supplied. Note that precisely one of `scales` and `pos.def.matrix` must be supplied
give.Z	In function `interpolant.quick()`, Boolean variable with `TRUE` meaning to return the best estimate and the error, and default `FALSE` meaning to return just the best estimate
distance.function	
	Function to compute distances between points, defaulting to `corr()`. See `corr.Rd` for details. Note that `method=2` or `method=3` is required if a non-standard distance function is used
...	Further arguments passed to the distance function, usually `corr()`

Value

In function `interpolant()`, if `give.full.list` is `TRUE`, a list is returned with components

betahat	Standard MLE of the (linear) fit, given the observations
prior	Estimate for the prior
sigmahat.square	
	Posterior estimate for variance
mstar.star	Posterior expectation
cstar	Prior correlation of a point with itself
cstar.star	Posterior correlation of a point with itself
Z	Standard deviation (although the distribution is actually a t-distribution with $n - q$ degrees of freedom)

Author(s)

Robin K. S. Hankin

References

- J. Oakley 2004. "Estimating percentiles of uncertain computer code outputs". Applied Statistics, 53(1), pp89-93.

- J. Oakley 1999. "Bayesian uncertainty analysis for complex computer codes", PhD thesis, University of Sheffield.

- J. Oakley and A. O'Hagan, 2002. "Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs", Biometrika 89(4), pp769-784

- R. K. S. Hankin 2005. "Introducing BACCO, an R bundle for Bayesian analysis of computer code output", Journal of Statistical Software, 14(16)

See Also

makeinputfiles,corr

Examples

```
# example has 10 observations on 6 dimensions.
# function is just sum( (1:6)*x) where x=c(x_1, ... , x_2)

data(toy)
val <- toy
real.relation <- function(x){sum( (0:6)*x )}
H <- regressor.multi(val)
d <- apply(H,1,real.relation)
d <- jitter(d,amount=1e-5)    # to prevent numerical problems

fish <- rep(1,6)
fish[6] <- 4
```

```
A <- corr.matrix(val,scales=fish)
Ainv <- solve(A)

# now add some suitably correlated noise to d:
d.noisy <-  as.vector(rmvnorm(n=1, mean=d, 0.1*A))
names(d.noisy) <- names(d)

# First try a value at which we know the answer (the first row of val):
x.known <- as.vector(val[1,])
bayes.known <- interpolant(x.known, d, val, Ainv=Ainv, scales=fish, g=FALSE)
print("error:")
print(d[1]-bayes.known)


# Now try the same value, but with noisy data:
print("error:")
print(d.noisy[1]-interpolant(x.known, d.noisy, val, Ainv=Ainv, scales=fish, g=FALSE))

#And now one we don't know:
x.unknown <- rep(0.5 , 6)
bayes.unknown <- interpolant(x.unknown, d.noisy, val, scales=fish, Ainv=Ainv,g=TRUE)

## [   compare with the "true" value of sum(0.5*0:6) = 10.5   ]



# Just a quickie for int.qq():
int.qq(x=rbind(x.unknown,x.unknown+0.1),d.noisy,val,Ainv,pos.def.matrix=diag(fish))


## (To find the best correlation lengths, use optimal.scales())

 # Now we use the SAME dataset but a different set of basis functions.
 # Here, we use the functional dependence of
 # "A+B*(x[1]>0.5)+C*(x[2]>0.5)+...+F*(x[6]>0.5)".
 # Thus the basis functions will be c(1,x>0.5).
 # The coefficients will again be 1:6.

      # Basis functions:
f <- function(x){c(1,x>0.5)}
      # (other examples might be
      # something like  "f <- function(x){c(1,x>0.5,x[1]^2)}"

      # now create the data
real.relation2 <- function(x){sum( (0:6)*f(x) )}
d2 <- apply(val,1,real.relation2)

      # Define a point at which the function's behaviour is not known:
x.unknown2 <- rep(1,6)
      # Thus real.relation2(x.unknown2) is sum(1:6)=21

      # Now try the emulator:
interpolant(x.unknown2, d2, val, Ainv=Ainv, scales=fish, g=TRUE)$mstar.star
```

```
        # Heh, it got it wrong!  (we know that it should be 21)


        # Now try it with the correct basis functions:
interpolant(x.unknown2, d2, val, Ainv=Ainv,scales=fish, func=f,g=TRUE)$mstar.star
        # That's more like it.

        # We can tell that the coefficients are right by:
betahat.fun(val,Ainv,d2,func=f)
        # Giving c(0:6), as expected.

        # It's interesting to note that using the *wrong* basis functions
        # gives the *correct* answer when evaluated at a known point:
interpolant(val[1,], d2, val, Ainv=Ainv,scales=fish, g=TRUE)$mstar.star
real.relation2(val[1,])
        # Which should agree.


        # Now look at Z.  Define a function Z() which determines the
        # standard deviation at a point near a known point.
Z <- function(o) {
   x <- x.known
   x[1] <- x[1]+ o
   interpolant(x, d.noisy, val, Ainv=Ainv, scales=fish, g=TRUE)$Z
  }

Z(0)      #should be zero because we know the answer (this is just Z at x.known)
Z(0.1)    #nonzero error.


  ## interpolant.quick() should  give the same results faster, but one
  ##   needs a matrix:
u <- rbind(x.known,x.unknown)
interpolant.quick(u, d.noisy, val, scales=fish, Ainv=Ainv,g=TRUE)




# Now an example from climate science.  "results.table" is a dataframe
# of goldstein (a climate model) results.  Each of its 100 rows shows a
# point in parameter space together with certain key outputs from the
# goldstein program.  The following R code shows how we can set up an
# emulator based on the first 27 goldstein runs, and use the emulator to
# predict the output for the remaining 73 goldstein runs.  The results
# of the emulator are then plotted on a scattergraph showing that the
# emulator is producing estimates that are close to the "real" goldstein
# runs.


data(results.table)
data(expert.estimates)

        # Decide which column we are interested in:
```

```
output.col <- 26

      # extract the "important" columns:
wanted.cols <- c(2:9,12:19)

      # Decide how many to keep;
      # 30-40 is about the most we can handle:
wanted.row <- 1:27

      # Values to use are the ones that appear in goin.test2.comments:
val <- results.table[wanted.row , wanted.cols]

      # Now normalize val so that 0<results.table[,i]<1 is
      # approximately true for all i:

normalize <- function(x){(x-mins)/(maxes-mins)}
unnormalize <- function(x){mins + (maxes-mins)*x}

mins  <- expert.estimates$low
maxes <- expert.estimates$high
jj <- t(apply(val,1,normalize))

jj <- as.data.frame(jj)
names(jj) <- names(val)
val <- jj


      ## The value we are interested in  is the 19th (or 20th or ... or 26th) column.
d  <- results.table[wanted.row ,  output.col]

      ##  Now some scales, estimated earlier from the data using
      ##   optimal.scales():

scales.optim <- exp(c( -2.917, -4.954, -3.354, 2.377, -2.457, -1.934, -3.395,
-0.444, -1.448, -3.075, -0.052, -2.890, -2.832, -2.322, -3.092, -1.786))

A <- corr.matrix(val,scales=scales.optim, method=2)
Ainv <-  solve(A)

print("and plot points used in optimization:")
d.observed <- results.table[ , output.col]

A <- corr.matrix(val,scales=scales.optim, method=2)
Ainv <- solve(A)

print("now plot all points:")
design.normalized <- as.matrix(t(apply(results.table[,wanted.cols],1,normalize)))
d.predicted <- interpolant.quick(design.normalized , d , val , Ainv=Ainv,
scales=scales.optim)
jj <- range(c(d.observed,d.predicted))
par(pty="s")
plot(d.observed, d.predicted, pch=16, asp=1,
xlim=jj,ylim=jj,
```

```
xlab=expression(paste(temperature," (",{}^o,C,"), model"   )),
ylab=expression(paste(temperature," (",{}^o,C,"), emulator"))
)
abline(0,1)
```

---

latin.hypercube	*Latin hypercube design matrix*

Description

Gives a Latin hypercube design matrix with an arbitrary number of points in an arbitrary number of dimensions. The toy dataset was generated using latin.hypercube().

Usage

```
latin.hypercube(n, d, names=NULL, normalize=FALSE, complex=FALSE)
```

Arguments

n	Number of points
d	Number of dimensions
names	Character vector for column names (optional)
normalize	Boolean variable with TRUE meaning to normalize each column so the minimum is zero and the maximum is one. If it takes its default FALSE, the points represent midpoints of n equispaced intervals; the points thus have a minimum of $0.5/n$ and a maximum of $1 - 0.5/n$
complex	Boolean with default FALSE meaning to return a complex latin hypercube in which real and imaginary components separately form a latin hypercube

Author(s)

Robin K. S. Hankin

Examples

```
#10 points, 6 dimensions:
(latin.hypercube(10,6) -> x)
plot(as.data.frame(x))

latin.hypercube(10,2,complex=TRUE)
```

| makeinputfiles | *Makes input files for condor runs of goldstein* |

Description

Wrapper to create arbitrary numbers of condor-compatible goldstein runnable input files. Function `sample.from.exp.est()` samples from the appropriate distribution.

This function is not designed for the general user: it is tailored for use in the environment of the National Oceanographic Centre, with a particular version of the specialist model "goldstein".

Usage

```
makeinputfiles(number.of.runs = 100, gaussian = TRUE,
   directoryname="~/goldstein/genie-cgoldstein/", filename="QWERTYgoin",
   expert.estimates, area.outside=0.05)
sample.from.exp.est(number.of.runs, expert.estimates,
   gaussian=TRUE, area.outside=0.05)
```

Arguments

number.of.runs	Number of condor runs to generate
gaussian	Boolean variable with default TRUE meaning use a lognormal distribution, and FALSE meaning a uniform distribution. In the case of a Gaussian distribution, only the upper and lower columns are used: here these values are interpreted as the 2.5%ile and 97.5%ile respectively and a lognormal distribution with the appropriate parameters is used.
	Note that this approach discards the "best" value, but OTOH it seemed to me that my expert chose his "best" value as an arithmetic (sic) mean of his high and low values, and thus has limited information content.
directoryname	Name of directory to which input files are written
filename	Basename of input files
expert.estimates	
	Dataframe holding expert estimates (supplied by a climate scientist). Use `data(expert.estimates)` to load a sample dataset that was supplied by Bob Marsh
area.outside	Area of tails of the lognormal distribution (on a log scale) that fall outside the expert ranges. Default value of 0.05 means interpret a and b as the 2.5%ile and 97.5%ile respectively.

Details

This function creates condor-compatible goldstein runnable input files that are placed in directory `/working/jrd/sat/rksh/goldstein`. The database `results.table` is made using the shell scripts currently in `/users/sat/rksh/goldstein/emulator`.

Note that `makeinputfiles(number.of.runs=n)` creates files numbered from 0 to $n-1$: so be careful of off-by-one errors. It's probably best to avoid reference to the "first", "second" file etc.

Instead, refer to files using their suffix number. Note that the suffix number is not padded with zeros due to the requirements of Condor.

The suffix number of a file matches the name of its `tmp` file (so, for example, file with suffix number 15 writes output to files `tmp/tmp.15` and `tmp/tmp.avg.15`).

Value

Returns zero on successful completion. The function is used for its side-effect of creating a bunch of Goldstein input files.

Author(s)

Robin K. S. Hankin

See Also

expert.estimates, results.table

Examples

```
## Not run:
  data(expert.estimates) system("mkdir /users/sat/rksh/tmp",ignore=TRUE)
  makeinputfiles(number.of.runs = 100, gaussian = TRUE,
  directoryname="~/tmp/", expert.estimate=expert.estimates)

## End(Not run)
data(results.table)
data(expert.estimates)


output.col <- 25
wanted.row <- 1:27
wanted.cols <- c(2:9,12:19)

val <- results.table[wanted.row , wanted.cols]

mins  <- expert.estimates$low
maxes <- expert.estimates$high

normalize <- function(x){(x-mins)/(maxes-mins)}
unnormalize <- function(x){mins + (maxes-mins)*x}

jj <- t(apply(val,1,normalize))

jj <- as.data.frame(jj)
names(jj) <- names(val)
val <- as.matrix(jj)

scales.optim <- exp(c( -2.63, -3.03, -2.24, 2.61,
-1.65, -3.13, -3.52, 3.16, -3.32, -2.53, -0.25,  -2.55, -4.98, -1.59,
-4.40, -0.81))
```

```
d  <- results.table[wanted.row ,  output.col]
A <- corr.matrix(val, scales=scales.optim)
Ainv <- solve(A)

x <- sample.from.exp.est(1000,exp=expert.estimates)
x <- t(apply(x,1,normalize))
ensemble <- interpolant.quick(x , d , val , Ainv, scales=scales.optim)
hist(ensemble)
```

---

model	*Simple model for concept checking*

Description

Oakley's simple model, used as an example for proof-of-concept

Usage

```
model(x)
```

Arguments

x   Input argument

Author(s)

Robin K. S. Hankin

Examples

```
model(seq(0,0.1,10))
```

OO2002	*Implementation of the ideas of Oakley and O'Hagan 2002*

Description

Implementation of the ideas of Oakley and O'Hagan 2002: var.conditional() calculates the conditional variance-covariance matrix, and cond.sample() samples from the appropriate multivariate t distribution.

Usage

```
cond.sample(n = 1, x, xold, d, A, Ainv, scales = NULL, pos.def.matrix =
NULL, func = regressor.basis, ...)
var.conditional(x, xold, d, A, Ainv, scales = NULL, pos.def.matrix = NULL,
    func = regressor.basis, distance.function = corr, ...)
```

Arguments

n	In function `cond.sample()`, the number of observations to take, defaulting to 1
x	Simulation design points
xold	Design points
d	Data vector
A	Correlation matrix
Ainv	Inverse of correlation matrix A
scales	Roughness lengths
pos.def.matrix	Positive definite matrix of correlations
func	Function to calculate H
distance.function	
	Distance function (defaulting to `corr()`)
...	Further arguments passed to the distance function, usually `corr()`

Details

We wish to generate the distribution for the process at uncertain point x; uncertainty in x is captured by assuming it to be drawn from a pdf X.

The basic idea is to estimate m^* at *simulated* design points using `cond.sample()`, which samples from the multivariate t distribution conditional on the data d at the design points. The random datavector of estimates m^* is called ddash.

We repeat this process many times, each time estimating $\eta(\cdot)$ using the augmented dataset `c(d,ddash)` as a training set.

For each estimated $\eta(\cdot)$, we have a complete emulator that can be used to build up an ensemble of estimates.

Value

Function `cond.sample()` returns a $n \times p$ matrix whose rows are independent samples from the appropriate multivariate t distribution. Here, p is the number of rows of x (ie the number of simulated design points). Consider a case where there are just two simulated design points, close to each other but far from any point of the original design points. Then function `cond.sample(n=4, ...)` will give four numbers which are close to one another but have high (between-instantiation) variance.

Function `var.conditional()` calculates the denominator of equation 3 of Oakley and OHagan 2002. This function is intended to be called by `cond.sample()` but might be interesting *per se* when debugging or comparing different choices of simulated design points.

Note

Function `cond.sample()` and `var.conditional()` together are a superset of function `interpolant()` because it accounts for covariance between multiple observations. It is, however, much slower.

Also note that these functions are used to good effect in the examples section of `oo2002.Rd`.

Author(s)

Robin K. S. Hankin

References

- J. Oakley 2002. *Bayesian inference for the uncertainty distribution of computer model outputs*. Biometrika, 89(4):769–784

- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

See Also

regressor.basis, for a more visually informative example of cond.sample() et seq; and interpolant for more examples

Examples

```
    # Now we use the functions.  First we set up some design points:
    # Suppose we are given the toy dataset and want to know the PDF of
    # fourth power of the response at point x, where uncertainty in x
    # may be represented as it being drawn from a normnl distribution
    # with mean c(0.5,0.5,...,0.5) and a variance of 0.001.

data(toy)
val <- toy
real.relation <- function(x){sum( (0:6)*x )}
H <- regressor.multi(val)
d <- apply(H,1,real.relation)

    # and some scales (which are assumed to be known):
fish <- rep(1,6)
fish[6] <- 4

    # And determine A and Ainv:
A <- corr.matrix(val,scales=fish)
Ainv <- solve(A)

    # and add some suitably correlated Gaussian noise:
d.noisy <-  as.vector(rmvnorm(n=1, mean=d, 0.1*A))
names(d.noisy) <- names(d)

    # Now some simulation design points.  Choose n'=6:
xdash <-  matrix(runif(36),ncol=6)

    # And just for fun, we insert a near-useless seventh simulation
    # design point (it is nearly useless because it is a near copy of
    # the sixth).  We do this in order to test the coding:
xdash <- rbind(xdash,xdash[6,] + 1e-4)
colnames(xdash) <- colnames(val)
rownames(xdash) <- c("alpha","beta","gamma","delta","epsilon","zeta","zeta.copy")
```

```
      # Print the variance matrix:
(vm <- var.conditional(x=xdash,xold=val,d=d.noisy,A=A,Ainv=Ainv,scales=fish))
      # Note that the sixth and seventh columns are almost identical
      # (and so, therefore, are the sixth and seventh rows) as
      # expected.

      # Also, the final eigenvalue of vm should be small:
eigen(vm)$values


      # Now sample from the conditional t-distribution.  Taking n=3 samples:
(cs <- cond.sample(n=3, x=xdash, xold=val, d=d.noisy, A=A, Ainv=Ainv,
                   scales = fish, func = regressor.basis))

      # Note the last two columns are nearly identical, as expected.

      # Just as a test, what is the variance matrix at the design points?
(vc <- var.conditional(x=val,xold=val,d=d.noisy,A=A,Ainv=Ainv,scales=fish))
      # (This should be exactly zero);
max(eigen(vc)$values)
      # should be small

      # Next, we apply the methods of OO2002 using Monte Carlo techniques.
      # We will generate  10 different versions of eta:
number.of.eta <- 10

      # And, for each eta,  we will sample from the posterior t distribution 11 times:
number.of.X <- 11


      # create an augmented design matrix, of the design points plus the
      # simulated design points:
design.augmented <- rbind(val,xdash)
A.augmented <- corr.matrix(design.augmented, scales=fish)
Ainv.augmented <- solve(A.augmented)

out <- NULL
for(i in seq_len(number.of.eta)){
        # Create random data by sampling from the conditional
        # multivariate t at the simulated design points xdash, from
        # the t-distribution given the data d:
   ddash <- cond.sample(n=1, x=xdash, xold=val, d=d.noisy, Ainv=Ainv, scales=fish)

        # Now use the emulator to calculate m^* at points chosen from
        # the PDF of X:
   jj <-
     interpolant.quick(x=rmvnorm(n=number.of.X,rep(0.5,6),diag(6)/1000),
                       d=c(d.noisy,ddash),
                       xold=design.augmented,
                       Ainv=Ainv.augmented,
                       scales=fish)
   out <- c(out,jj)
```

```
}

   # histogram of the fourth power:
hist(out^4, col="gray")
   # See oo2002 for another example of cond.sample() in use
```

---

optimal.scales   *Use optimization techniques to find the optimal scales*

Description

Uses optimization techniques (either Nelder-Mead or simulated annealing) to find the optimal
scales, or roughness lengths. Function `optimal.scale()` (ie singular) finds the optimal scale on
the assumption that the roughness is isotropic so all scales are identical.

Usage

```
optimal.scales(val, scales.start, d, use.like = TRUE,  give.answers =
FALSE, func=regressor.basis, ...)
optimal.scale(val, d, use.like = TRUE,  give.answers =
FALSE, func=regressor.basis, ...)
```

Arguments

`val`	Matrix with rows corresponding to points at which the function is known
`scales.start`	Initial guess for the scales (plural). See details section for explanation
`d`	vector of observations, one for each row of `val`
`use.like`	Boolean, with default `TRUE` meaning to use likelihood for the objective function, and `FALSE` meaning to use a leave-out-one bootstrap estimator
`give.answers`	Boolean, with default `FALSE` meaning to return just the roughness lengths and `TRUE` meaning to return extra information as returned by `optim()`
`func`	Function used to determine basis vectors, defaulting to `regressor.basis` if not given
`...`	Extra parameters passed to `optim()` or `optimize()`. See examples for usage of this argument

Details

Internally, this function works with the logarithms of the roughness lengths, because they are
inherently positive. However, note that the lengths themselves must be supplied to argument
`scales.start`, not their logarithms.

The reason that there are two separate functions is that `optim()` and `optimize()` are very different.

Value

If `give.answers` takes the default value of `FALSE`, a vector of roughness lengths is returned. If `TRUE`, output from `optim()` is returned directly (note that element `par` is the logarithm of the desired roughness length as the optimization routine operates with the logs of the lengths as detailed above)

Note

This function is slow to evaluate because it needs to calculate and invert `A` each time it is called, because the scales change from call to call.

In this package, "scales" means the diagonal elements of the B matrix. See the help page for `corr` for more discussion of this topic.

Note the warning about partial matching under the "dot-dot-dot" argument in both `optim.Rd` [used in `optimal.scales()`] and `optimize.Rd` [used in `optimal.scale()`]: any unmatched arguments will be passed to the objective function. Thus, passing named but unmatched arguments to `optimal.scale[s]()` will cause an error, because those arguments will be passed, by `optim()` or `optimize()`, to the (internal) `objective.fun()`.

In particular, note that passing `control=list(maxit=4)` to `optimal.scale()` will cause an error for this reason [`optimize()` does not take a `control` argument].

Author(s)

Robin K. S. Hankin

References

- J. Oakley 2004. *Estimating percentiles of uncertain computer code outputs*. Applied Statistics, 53(1), pp89-93.

- J. Oakley 1999. *Bayesian uncertainty analysis for complex computer codes*, PhD thesis, University of Sheffield.

See Also

[interpolant,corr](interpolant)

Examples

```
##First, define some scales:
fish <- c(1,1,4)

## and a sigmasquared value:
REAL.SIGMASQ <- 0.3

## and a real relation:
real.relation <- function(x){sum( (1:3)*x )}

## Now a design matrix:
val  <- latin.hypercube(7,3)

## apply the real relation:
```

```
d <- apply(val,1,real.relation)

## and add some suitably correlated Gaussian noise:
A <- corr.matrix(val,scales=fish)
d.noisy <-  as.vector(rmvnorm(n=1,mean=apply(val,1,real.relation),REAL.SIGMASQ*A))

##  Now see if we can estimate the roughness lengths well.  Remember that
##  the true values are those held in vector "fish":

optimal.scales(val=val, scales.start=rep(1,3), d=d.noisy,
        method="SANN",control=list(trace=1000,maxit=3),
        give=FALSE)



# Now a test of optimal.scale(), where there is only a single roughness
#  scale to estimate.  This should be more straightforward:


df <- latin.hypercube(7,3)
fish2 <- rep(2,3)
A2 <- corr.matrix(df,scales=fish2)
d.noisy <- as.vector(rmvnorm(n=1, mean=apply(df,1,real.relation), sigma=A2))

jj.T <- optimal.scale(val=df,d=d.noisy,use.like=TRUE)
jj.F <- optimal.scale(val=df,d=d.noisy,use.like=FALSE)
```

---

pad   *Simple pad function*

Description

Places zeros to the left of a string. If the string consists only of digits 0-9, pad() does not change
the value of the string if interpreted as a numeric.

Usage

```
pad(x,len,padchar="0",strict=TRUE)
```

Arguments

x	Input argument (converted to character)
len	Desired length of output
padchar	Character to pad x with, defaulting to "0"
strict	Boolean variable governing the behaviour when length of x is less than len. Under these circumstances, if strict takes the default value of TRUE, then return an error; if FALSE, return a truncated version of x (least significant characters retained)

Author(s)

Robin K. S. Hankin

Examples

```
pad("1234",len=10)
pad("1234",len=3,strict=FALSE)
```

---

prior.b   *Prior linear fits*

Description

Gives the fitted regression coefficients corresponding to the specified regression model.

Usage

```
prior.b(H, Ainv, d, b0 = NULL, B0 = NULL)
prior.B(H , Ainv , B0=NULL)
```

Arguments

H	Regression basis function (eg that returned by `regressor.multi()`)
Ainv	A^{-1} where A is a correlation matrix (eg that returned by `corr.matrix()`)
d	Vector of data points
b0	prior constant
B0	prior coefficients

Author(s)

Robin K. S. Hankin

References

- J. Oakley 2004. *Estimating percentiles of uncertain computer code outputs*. Applied Statistics, 53(1), pp89-93.

- J. Oakley 1999. *Bayesian uncertainty analysis for complex computer codes*, PhD thesis, University of Sheffield.

Examples

```
# example has 10 observations on 6 dimensions.
# function is just sum( (1:6)*x) where x=c(x_1, ... , x_2)

data(toy)
val <- toy
d <- apply(val,1,function(x){sum((1:6)*x)})

#add some noise:
d <- jitter(d)

A <- corr.matrix(val,scales=rep(1,ncol(val)))
Ainv <- solve(A)
H <- regressor.multi(val)

prior.b(H,Ainv,d)
prior.B(H,Ainv)
```

---

quad.form   *Evaluate a quadratic form efficiently*

Description

Given a square matrix M of size $n \times n$, and a matrix x of size $n \times p$ (or a vector of length n), evaluate various quadratic forms.

(in the following, x^T denotes the complex conjugate of the transpose, also known as the Hermitian transpose. This only matters when considering complex numbers).

- Function quad.form(M,x) evaluates $x^T M x$ in an efficient manner
- Function quad.form.inv(M,x) returns $x^T M^{-1} x$ using an efficient method that avoids inverting M
- Function quad.tform(M,x) returns $x M x^T$ using tcrossprod() without taking a transpose
- Function quad.tform.inv(M,x) returns $x M^{-1} x^T$, although a single transpose is needed
- Function quad.3form(M,l,r) returns $l^T M r$ using nested calls to crossprod(). It's no faster than calling crossprod() directly, but makes code neater and less error-prone (IMHO)
- Function quad.3form.inv(M,l,r) returns $l^T M^{-1} r$
- Function quad.3tform(M,l,r) returns $l M r^T$ using nested calls to tcrossprod(). Again, this is to make for neater code
- Function quad.diag(M,x) returns the *diagonal* of the (potentially very large) square matrix quad.form(M,x) without calculating the off diagonal elements
- Function quad.tdiag(M,x) similarly returns the diagonal of quad.tform(M,x)
- Function quad.3diag(M,l,r) returns the diagonal of quad.3form(M,l,r)
- Function quad.3tdiag(M,l,r) returns the diagonal of quad.3tform(M,l,r)

These functions invoke the following lower-level calls:

- Function `ht(x)` returns the Hermitian transpose, that is, the complex conjugate of the transpose, sometimes written x^*
- Function `cprod(x,y)` returns $x^T y$, equivalent to `crossprod(Conj(x),y)`
- Function `tcprod(x,y)` returns xy^T, equivalent to `crossprod(x,Conj(y))`

Note again that in the calls above, "transpose" [that is, x^T] means "Conjugate transpose", or the Hermitian transpose.

Usage

```
quad.form(M, x, chol=FALSE)
quad.form.inv(M, x)
quad.tform(M, x)
quad.3form(M,left,right)
quad.3tform(M,left,right)
quad.tform.inv(M,x)
quad.diag(M,x)
quad.tdiag(M,x)
quad.3diag(M,left,right)
quad.3tdiag(M,left,right)
cprod(x,y)
tcprod(x,y)
ht(x)
```

Arguments

M	Square matrix of size $n \times n$
x, y	Matrix of size $n \times p$, or vector of length n
chol	Boolean, with `TRUE` meaning to interpret argument `M` as the lower triangular Cholesky decomposition of the quadratic form. Remember that `M.lower %*% M.upper == M`, and `chol()` returns the upper triangular matrix, so one needs to use the transpose `t(chol(M))`
left, right	In function `quad.3form()`, matrices with n rows and arbitrary number of columns

Details

The "meat" of `quad.form()` for `chol=FALSE` is just `crossprod(crossprod(M, x), x)`, and that of `quad.form.inv()` is `crossprod(x, solve(M, x))`.

If the Cholesky decomposition of `M` is available, then calling with `chol=TRUE` and supplying `M.upper` should generally be faster (for large matrices) than calling with `chol=FALSE` and using `M` directly. The time saving is negligible for matrices smaller than about 50×50, even if the overhead of computing `M.upper` is ignored.

Note

These functions are used extensively in the emulator and calibrator packages' R code, primarily in the interests of elegant code, but also speed. For the problems I usually consider, the speedup (of `quad.form(M,x)` over `t(x) %*% M %*% x`, say) is marginal at best.

Author(s)

Robin K. S. Hankin

See Also

[optimize](#)

Examples

```
jj <- matrix(rnorm(80),20,4)
M <- crossprod(jj,jj)
M.lower <- t(chol(M))
x <- matrix(rnorm(8),4,2)

jj.1 <- t(x) %*% M %*% x
jj.2 <- quad.form(M,x)
jj.3 <- quad.form(M.lower,x,chol=TRUE)
print(jj.1)
print(jj.2)
print(jj.3)




## Make two Hermitian positive-definite matrices:
L <- matrix(c(1,0.1i,-0.1i,1),2,2)
LL <- diag(11)
LL[2,1] <- -(LL[1,2] <- 0.1i)


z <- t(latin.hypercube(11,2,complex=TRUE))


quad.diag(L,z)     # elements real because L is HPD
quad.tdiag(LL,z)   # ditto




## Now consider accuracy:
quad.form(solve(M),x) - quad.form.inv(M,x)  # should be zero
quad.form(M,x) - quad.tform(M,t(x))         # should be zero
quad.diag(M,x) - diag(quad.form(M,x))       # should be zero
diag(quad.form(L,z))  - quad.diag(L,z)      # should be zero
diag(quad.tform(LL,z)) - quad.tdiag(LL,z)   # should be zero
```

---

regressor.basis   *Regressor basis function*

Description

Creates a regressor basis for a vector.

Usage

```
regressor.basis(x)
regressor.multi(x.df,func=regressor.basis)
```

Arguments

x	vector of coordinates
x.df	Matrix whose rows are coordinates of points
func	Regressor basis function to use; defaults to `regressor.basis`

Details

The regressor basis specified by `regressor.basis()` is just the addition of a constant term, which is conventionally placed in the first position. This is a very common choice for a set of bases, although it is important to investigate both simpler and more sophisticated alternatives. Tony would recommend simpler functions (perhaps as simple as `function(x){1}`, that is, nothing but a constant), and Jonty would recommend more complicated bespoke functions that reflect prior beliefs.

Function `regressor.multi()` is just a wrapper for `regressor.basis()` that works for matrices. This is used internally and the user should not need to change it.

Note that the user is free to define and use functions other than this one when using, for example, `corr()`.

Value

Returns simple regressor basis for vectors or matrices.

Note

When writing replacements for `regressor.basis()`, it is important to return a vector with at least one **named** element (see the R source code for function `regressor.basis()`, in which the first element is named "const").

Returning a vector all of whose elements are unnamed will cause some of the package functions to fail in various weird places. It is good practice to use named vectors in any case.

Function `regressor.multi()` includes an ugly hack to ensure that the perfectly reasonable choice of `regressor.basis=function(x){1}` works. The hack is needed because `apply()` treats functions that return a length-1 value argument differently from functions that return a vector: if x <- `as.matrix(1:10)`

then

`apply(x,1,function(x){c(1,x)})`

returns a matrix (as desired), but

`apply(x,1,function(x){c(1)})`

returns a vector (of 1s) which is not what is wanted. The best way to deal with this (IMHO) is to confine the ugliness to a single function, here `regressor.multi()`.

Author(s)

Robin K. S. Hankin

Examples

```
regressor.basis(rep(5,6))
m <- matrix(1:27,9,3)
regressor.multi(m)
regressor.multi(m,func=function(x){c(a=88,x,x^2,x[1]^4)})


# and now a little example where we can choose the basis functions
# explicitly and see the effect it has.  Note particularly the poor
# performance of func2() in extrapolation:


func1 <- function(x){
  out <- c(1,cos(x))
  names(out) <- letters[1:length(x)]
  return(out)
}

func2 <- function(x){
  out <- c(1,cos(x),cos(2*x),cos(3*x))
  names(out) <- letters[1:length(x)]
  return(out)
}

func3 <- function(x){out <- c(1,x)
names(out)[1] <- "const"
return(out)
}

func.chosen <- func1


toy <- sort(c(seq(from=0,to=1,len=9),0.2))
toy <- as.matrix(toy)
colnames(toy) <- "a"
rownames(toy) <- paste("obs",1:nrow(toy),sep=".")

d.noisy <- as.vector(toy>0.5)+rnorm(length(toy))/40

fish <- 100
x <- seq(from=-1,to=2,len=200)
A <- corr.matrix(toy,scales=fish)
Ainv <- solve(A)

 ## Now the interpolation.  Change func.chosen() from func1() to func2()
 ## and see the difference!

jj <- interpolant.quick(as.matrix(x), d.noisy, toy, scales=fish,
```

```
                            func=func.chosen,
                             Ainv=Ainv,g=TRUE)

plot(x,jj$mstar.star,xlim=range(x),type="l",col="black",lwd=3)
lines(x,jj$prior,col="green",type="l")
lines(x,jj$mstar.star+jj$Z,type="l",col="red",lty=2)
lines(x,jj$mstar.star-jj$Z,type="l",col="red",lty=2)
points(toy,d.noisy,pch=16,cex=2)
legend("topright",lty=c(1,2,1,0),
    col=c("black","red","green","black"),pch=c(NA,NA,NA,16),
    legend=c("best estimate","+/-1 sd","prior","training set"))



  ## Now we will use O&O 2002.

## First, some simulated design points:
xdash <- as.matrix(c(-0.5, -0.1, -0.2, 1.1, 1.15))

## create an augmented design set:
design.augmented <- rbind(toy,xdash)

## And calculate the correlation matrix of the augmented dataset:
A.augmented <- corr.matrix(design.augmented, scales=fish)
Ainv.augmented <- solve(A.augmented)

## Now, define a function that samples from the
## appropriate posterior t-distribution, adds these random
## variables to the dataset, then calculates a new
## etahat and evaluates and plots it:

f <- function(...){

ddash <- cond.sample(n=1, x=xdash, xold=toy, d=d.noisy, A=A,
                            Ainv=Ainv, scales=fish, func=func.chosen)

        jj.aug <-
           interpolant.quick(x       = as.matrix(x),
                             d       = c(d.noisy,as.vector(ddash)),
                             xold    = design.augmented,
                             Ainv    = Ainv.augmented,
                             scales = fish,func=func.chosen)
points(xdash,ddash,type="p",pch=16,col="gray")
points(x, jj.aug, type="l", col="gray")
}


## Now execute the function a few times to assess the uncertainty in eta:
f()
f()
f()
```

---

results.table   *Results from 100 Goldstein runs*

Description

A dataframe consisting of 100 rows, corresponding to 100 runs of Goldstein. Columns 1-19 are input values; columns 20-27 are outputs gleaned from goout files.

Usage

```
data(results.table)
```

Format

A data frame with 100 observations on the following 27 variables.

filenumber Number of the condor run (ie file appenge of goin.* and goout.*)

windstress a numeric vector (input value)

oc.horiz.diffus a numeric vector (input value)

oc.vert.diffus a numeric vector (input value)

oc.drag a numeric vector (input value)

at.heat.diffus a numeric vector (input value)

at.mois.diffus a numeric vector (input value)

at.width a numeric vector (input value)

at.slope a numeric vector (input value)

advfact.zonalheat a numeric vector (input value)

advfact.meridheat a numeric vector (input value)

advfact.zonalmois a numeric vector (input value)

advfact.meridmois a numeric vector (input value)

co2.scaling a numeric vector (input value)

clim.sens a numeric vector (input value)

thres.humid a numeric vector (input value)

ice.diffus a numeric vector (input value)

fw.scaling a numeric vector (input value)

solar.const a numeric vector (input value)

ominp a numeric vector (output value)

omaxp a numeric vector (output value)

omina a numeric vector (output value)

omaxa a numeric vector (output value)

avn a numeric vector (output value)

rms a numeric vector (output value)

average.SAT a numeric vector (output value)

model.error a numeric vector (output value)

Examples

```
data(results.table)
```

---

s.chi	*Variance estimator*

Description

Returns estimator for a priori σ^2

Usage

```
s.chi(H, Ainv, d, s0 = 0, fast.but.opaque = TRUE)
```

Arguments

H	Regression basis function (eg that returned by `regressor.multi()`)
Ainv	A^{-1} where A is a correlation matrix (eg that returned by `corr.matrix()`)
d	Vector of data points
s0	Optional offset

fast.but.opaque

Boolean, with default `TRUE` meaning to use `quad.form()`, and `FALSE` meaning to use straightforward `%*%`. The first form should be faster, but the code is less intelligible than the second form. Comparing the returned value with this argument on or off should indicate the likely accuracy attained.

Details

See O'Hagan's paper (ref below), equation 12 for details and context.

Author(s)

Robin K. S. Hankin

References

A. O'Hagan 1992. "Some Bayesian Numerical Analysis", pp345-363 of *Bayesian Statistics 4* (ed J. M. Bernardo et al), Oxford University Press

Examples

```
# example has 10 observations on 6 dimensions.
# function is just sum( (1:6)*x) where x=c(x_1, ... , x_2)
data(toy)
val <- toy
colnames(val) <- letters[1:6]
H <- regressor.multi(val)
d <- apply(H,1,function(x){sum((0:6)*x)})


# create A matrix and its inverse:
A <- corr.matrix(val,scales=rep(1,ncol(val)))
Ainv <- solve(A)

# add some suitably correlated noise:
d <- as.vector(rmvnorm(n=1, mean=d, 0.1*A))

# now evaluate s.chi():
s.chi(H, Ainv, d)


# assess accuracy:
s.chi(H, Ainv, d, fast=TRUE) - s.chi(H, Ainv, d, fast=FALSE)
```

---

sample.n.fit	*Sample from a Gaussian process and fit an emulator to the points*

Description

Sample 'n' fit: sample from an appropriate multivariate Gaussian process in one dimension, then fit an emulator to it.

Usage

```
sample.n.fit(n = 10, scales.generate = 100, scales.fit = 100, func = regressor.basis, ...)
```

Arguments

n	Number of observations to make
scales.generate	
	Scales to generate the data with: small values give uncorrelated observations, large values give correlated observations (hence the points fall on a smooth line)
scales.fit	Scales to use to fit the emulator. Small values give an emulator that is the prior with short, sharp excursions to make the emulator go through the points; large values give smooth emulators that exhibit overshoots resembling Gibbs's phenomenon

func	Function used to determine basis vectors, defaulting to `regressor.basis` if not given.
...	Further arguments passed to `plot()`.

Details

The point of this function is to investigate what happens when inappropriate scales are used for the emulator: that is, when `scales.generate` and `scales.fit` are wildly different.

Note that the sampling distribution has a constant expectation (of zero); so the prior should be zero, making it easy to see mispredictions of beta.

Author(s)

Robin K. S. Hankin

Examples

```
sample.n.fit(main="Default: scales match")
sample.n.fit(scales.generate=5,main="generate scale small")

# Now use a quadratic function instead of the default linear:
 f <- function(x){out <- c(1,x,x^2)
 names(out) <- c("const","linear","quadratic")
 out}

sample.n.fit(main="quadratic prior" , func=f)
```

---

scales.likelihood	*Likelihood of roughness parameters*

Description

Gives the a postiori likelihood for the roughness parameters as a function of the observations.

Usage

```
scales.likelihood(pos.def.matrix = NULL, scales = NULL, xold,
use.Ainv = TRUE, d, give_log=TRUE, func = regressor.basis)
```

Arguments

pos.def.matrix	Positive definite matrix used for the distance metric
scales	If the positive definite matrix is diagonal, `scales` specifies the diagonal elements. Specify exactly one of `pos.def.matrix` or `scales` (ie not both)
xold	Points at which code has been run

use.Ainv	Boolean, with default `TRUE` meaning to calculate A^{-1} explicitly and use it. Setting to `FALSE` means to use methods (such as `quad.form.inv()`) which do not require inverting the `A` matrix. Although one should avoid inverting a matrix if possible, in practice there does not appear to be much difference in execution time for the two methods
d	Observations in the form of a vector with entries corresponding to the rows of `xold`
give_log	Boolean, with default `TRUE` meaning to return the logarithm of the likelihood (ie the support) and `FALSE` meaning to return the likelihood itself
func	Function used to determine basis vectors, defaulting to `regressor.basis` if not given

Details

This function returns the likelihood function defined in Oakley's PhD thesis, equation 2.37. Maximizing this likelihood to estimate the roughness parameters is an alternative to the leave-out-one method on the `interpolant()` helppage; both methods perform similarly.

The value returned is

$$(\hat{\sigma})^{-(n-q)/2} |A|^{-1/2} \cdot \left| H^T A^{-1} H \right|^{-1/2} .$$

Value

Returns the likelihood or support.

Note

This function uses a Boolean flag, `use.Ainv`, to determine whether `A` has to be inverted or not. Compare the other strategy in which separate functions, eg `foo()` and `foo.A()`, are written. An example would be `betahat.fun()`.

Author(s)

Robin K. S. Hankin

References

- J. Oakley 1999. *Bayesian uncertainty analysis for complex computer codes*, PhD thesis, University of Sheffield.

- J. Oakley and A. O'Hagan, 2002. *Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs*, Biometrika 89(4), pp769-784

- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

See Also

[optimal.scales](#)

Examples

```
data(toy)
val <- toy

#define a real relation
real.relation <- function(x){sum( (0:6)*x )}

#Some scales:
fish <- rep(1,6)
fish[6] <- 4
A <- corr.matrix(val,scales=fish)
Ainv <- solve(A)

# Gaussian process noise:
H <- regressor.multi(val)
d <- apply(H,1,real.relation)
d.noisy <- as.vector(rmvnorm(n=1,mean=d, 0.1*A))

# Compare likelihoods with true values and another value:
scales.likelihood(scales=rep(1,6),xold=toy,d=d.noisy)
scales.likelihood(scales=fish     ,xold=toy,d=d.noisy)


# Verify that use.Ainv does not affect the numerical result:
u.true  <- scales.likelihood(scales=rep(1,6),xold=toy,d=d.noisy,use.Ainv=TRUE)
u.false <- scales.likelihood(scales=rep(1,6),xold=toy,d=d.noisy,use.Ainv=FALSE)
print(c(u.true, u.false))  # should be identical up to numerical accuracy


# Now use optim():
f <- function(fish){scales.likelihood(scales=exp(fish), xold=toy, d=d.noisy)}
e <-
optim(log(fish),f,method="Nelder-Mead",control=list(trace=0,maxit=10,fnscale=
-1))
best.scales <- exp(e$par)
```

---

sigmahatsquared   *Estimator for sigma squared*

Description

Returns maximum likelihood estimate for sigma squared. The ".A" form does not need Ainv, thus removing the need to invert A. Note that this form is *slower* than the other if Ainv is known in advance, as solve(.,.) is slow.

Usage

```
sigmahatsquared(H, Ainv, d)
sigmahatsquared.A(H, A, d)
```

Arguments

H	Regressor matrix (eg as returned by `regressor.multi()`)
A	Correlation matrix (eg `corr.matrix(val)`)
Ainv	Inverse of the correlation matrix (eg `solve(corr.matrix(val))`)
d	Vector of observations

Details

The formula is

$$\frac{y^T \left(A^{-1} - A^{-1}H(H^T A^{-1}H)^{-1}H^T A^{-1} \right) y}{n - q - 2}$$

where y is the data vector, H the matrix whose rows are the regressor functions of the design matrix, A the correlation matrix, n the number of observations and q the number of elements in the basis function.

Author(s)

Robin K. S. Hankin

References

- J. Oakley and A. O'Hagan, 2002. *Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs*, Biometrika 89(4), pp769-784

- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

Examples

```
## First, set sigmasquared to a value that we will try to estimate at the end:
REAL.SIGMASQ <- 0.3

## First, some data:
val <- latin.hypercube(100,6)
H <- regressor.multi(val,func=regressor.basis)

## now some scales:
 fish <- c(1,1,1,1,1,4)

## A and Ainv
A <- corr.matrix(as.matrix(val),scales=fish)
Ainv <- solve(A)

## a real relation; as used in helppage for interpolant:
real.relation <- function(x){sum( (1:6)*x )}

## use the real relation:
d <- apply(val,1,real.relation)
```

```
## now add some Gaussian process noise:
d.noisy <-  as.vector(rmvnorm(n=1,mean=d, REAL.SIGMASQ*A))

## now estimate REAL.SIGMASQ:
sigmahatsquared(H,Ainv,d.noisy)

## That shouldn't be too far from the real value specified above.

## Finally, a sanity check:
sigmahatsquared(H,Ainv,d.noisy) - sigmahatsquared.A(H,A=A,d.noisy)
```

---

toy   *A toy dataset*

Description

A matrix consisting of 10 rows and 6 columns corresponding to 10 points in a six-dimensional space.

Usage

```
data(toy)
```

Examples

```
data(toy)
real.relation <- function(x){sum( (1:6)*x )}

d <- apply(toy, 1, real.relation)

# Supply some scales:
fish <- rep(2,6)

# Calculate the A matrix:
A <- corr.matrix(toy,scales=fish)
Ainv <- solve(A)

# Now add some suitably correlated noise:
d.noisy <- as.vector(rmvnorm(n=1,mean=d, 0.1*A))

# Choose a point:
x.unknown <- rep(0.5,6)

# Now use interpolant:
interpolant(x.unknown, d.noisy, toy, Ainv, scales=fish, g=FALSE)


# Now verify by checking the first row of toy:
```

```
interpolant(toy[1,], d.noisy, toy, Ainv, scales=fish, g=FALSE)
# Should match d.noisy[1].
```

---

tr	*Trace of a matrix*

Description

Returns the trace of a matrix

Usage

```
tr(a)
```

Arguments

a   Matrix whose trace is desired

Author(s)

Robin K. S. Hankin

Examples

```
tr(matrix(1:9,3,3))
```

Index