Package ‘calibrator’

February 19, 2015

Type     Package
Title    Bayesian calibration of complex computer codes
Version  1.2-6
Date     28 June 2009
Author   Robin K. S. Hankin
Depends  R (>= 2.0.0), emulator (>= 1.2-11)
Imports  cubature
Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>
Description Performs Bayesian calibration of computer models as per
Kennedy and O’Hagan 2001. The package includes routines to find the
hyperparameters and parameters; see the help page for stage1() for a
worked example using the toy dataset. A tutorial is provided in the
calex.Rnw vignette; and a suite of especially simple one dimensional
examples appears in inst/doc/one.dim/.
License  GPL
NeedsCompilation no
Repository CRAN
Date/Publication 2013-12-09 08:22:43

R topics documented:

beta1hat.fun .................................................. 2
beta2hat.fun .................................................. 4
betahat.fun.koh ............................................. 5
blockdiag ...................................................... 7
C1 ............................................................. 8
cov.p5.supp ................................................... 9
create.new.toy.datasets .................................... 11
D1.fun ........................................................ 12
D2.fun ........................................................ 13
dists.2frames ............................................... 14
E.theta.toy .................................................. 15
Description

Least squares estimator for beta1

Usage

beta1hat.fun(D1, H1, y, phi)

Arguments

D1  code run points
H1  regressor basis funs
y   code outputs
phi hyperparameters
Author(s)
Robin K. S. Hankin

References

See Also
- beta2hat.fun

Examples
```r
data(toys)
y.toy <- create.new.toy.datasets(D1=D1.toy, D2=D2.toy)$y.toy
beta1hat.fun(D1=D1.toy, H1=H1.toy, y=y.toy, phi=phi.toy)

# now cheat: force the hyperparameters to have the correct psi:
phi.fix <- phi.change(old.phi=phi.toy,psi=c(1,0.5,1.0,1.0,0.5,0.4),phi.fun=phi.fun.toy)
# The value for psi is obtained by cheating and examining the source
# code for computer.model(); see ?phi.change

# Create a new toy dataset with 40 observations:
D1.big <- latin.hypercube(40,5)
jj <- create.new.toy.datasets(D1=D1.big, D2=D2.toy)

# We know that the real coefficients are 4:9 because we
# we can cheat and look at the source code for computer.model()

# Now estimate the coefficients without cheating:
beta1hat.fun(D1=D1.big, H1=H1.toy, jj$y, phi=phi.toy)

# Not bad!

# We can do slightly better by cheating and using the
# correct value for the hyperparameters:
beta1hat.fun(D1=D1.big, H1=H1.toy, jj$y, phi=phi.true.toy(phi=phi.toy))

#marginally worse.
```
beta2hat.fun

estimator for beta2

Description

estimates beta2 as per the equation of page 4 of the supplement. Used by p.page4()

Usage

beta2hat.fun(D1, D2, H1, H2, V, z, etahat.d2, extractor, E.theta, Edash.theta, phi)

Arguments

D1 Matrix of code run points
D2 Matrix of observation points
H1 regression basis functions
H2 regression basis functions
V overall covariance matrix
z vector of observations
etahat.d2 expectation as per etahat.vector
extractor extractor function
E.theta Expectation
Edash.theta Expectation wrt thetadash
phi hyperparameters

Author(s)

Robin K. S. Hankin

References

**betahat.fun.koh**

**Description**

Determines the mean of $\beta$, given parameters $\theta$, hyperparameters $\phi$, and the vector of code outputs and observations $d$. It is named so as to avoid conflict with function betahat.fun of package emulator.

**Usage**

```r
betahat.fun.koh(D1, D2, H1, H2, theta, d, phi)
betahat.fun.koh.vector(D1, D2, H1, H2, theta, d, phi)
```

**Examples**

```r
data(toys)

etahat.d2 <- etahat(D1=D1.toy, D2=D2.toy, H1=H1.toy, y=y.toy,
E.theta=E.theta.toy, extractor=extractor.toy, phi=phi.toy)

beta2hat.fun(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, V=NULL,
z=z.toy, etahat.d2=etahat.d2, extractor=extractor.toy,
E.theta=E.theta.toy, Edash.theta=Edash.theta.toy, phi=phi.toy)

jj <- create.new.datasets(D1.toy, D2.toy)
phi.true <- phi.true.toy(phi=phi.toy)
y.toy <- jj$y.toy
z.toy <- jj$z.toy
d.toy <- jj$d.toy
e
e
etahat.d2 <- etahat(D1=D1.toy, D2=D2.toy, H1=H1.toy, y=y.toy,
E.theta=E.theta.toy, extractor=extractor.toy, phi=phi.toy)

beta2hat <- beta2hat.fun(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, V=NULL,
z=z.toy, etahat.d2=etahat.d2, extractor=extractor.toy,
E.theta=E.theta.toy, Edash.theta=Edash.theta.toy,
phi=phi.toy)

print(beta2hat)

plot(z.toy, H2.toy(D2.toy) %*% beta2hat)
```
Arguments

- **D1**: Matrix whose rows are observation points and parameter values at which the code has been run.
- **D2**: Matrix whose rows are the observation points.
- **H1**: Regression function for D1.
- **H2**: Regression function for D2.
- **theta**: Parameters.
- **d**: Vector of code outputs and observations.
- **phi**: Hyperparameters.

Details

This function is defined between equations 2 and 3 of the supplement. It is used in functions `ezNeqn9NsuppHI` and `pNeqnXNsuppHI`. The user should always use `betahatNfunNkohHI`, which is a wrapper for `betahatNfunNkohNvectorHI`. The forms differ in their treatment of \( \theta \). In the former, \( \theta \) must be a vector; in the latter, \( \theta \) may be a matrix, in which case `betahatNfunNkohNvectorHI` is applied to the rows.

In `betahatNfunNkohHI`, the rownames are assigned by a kludgy call to `hNfunHI`, which itself uses a kludge to determine colnames.

The function returns

\[
\hat{\beta}(\theta) = W(\theta)^T H(\theta)^T V_d(\theta)^{-1} d.
\]

Author(s)

Robin K. S. Hankin

References


Examples

```r
data(toys)
betahat.fun.koh(theta=theta.toy, d=d.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)

betahat.fun.koh.vector(theta=theta.toy, d=d.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)
```

---

*betahat.fun.koh*
jj.theta <- rbind(theta.toy, theta.toy+1, theta.toy+2, theta.toy+0)
betalhats.koh(theta=jj.theta, d=d.toy, D1=D1.toy, D2=D2.toy,
           H1=H1.toy, H2=H2.toy, phi=phi.toy)

## Now try with true hyperparameters:
phi.true <- phi.true.toy(phi=phi.toy)

## And magically create the REAL parameters:
theta.REAL <- create.new.toy.datasets(export=TRUE)$REAL.PARAMS
jj.theta <- rbind(jj.theta, theta.REAL)

## Generate some data:
jj <- create.new.toy.datasets(D1.toy, D2.toy)
d.toy <- jj$d.toy

## And finally, observe that the estimated values for beta are pretty
## close to the real values (which omniscient beings can extract using
## reality() and computer.model()):
betalhats.koh(theta=jj.theta, d=d.toy, D1=D1.toy, D2=D2.toy,
           H1=H1.toy, H2=H2.toy, phi=phi.true)

## [  
## that is, compare the last column of the above with
## c(computer.model(ex=T)$REAL.COEFFS, reality(ex=T)$REAL.BETA2)
## ]

blockdiag

Assembles matrices blockwise into a block diagonal matrix

Description

Assembles matrices blockwise into a block diagonal matrix with optional padding value

Usage

blockdiag(m1, m2, p.tr = 0, p.ll = 0)

Arguments

m1 Upper left matrix
m2 Lower right matrix
p.tr Padding value for top right quadrant. Defaults to zero.
p.ll Padding value for lower left quadrant. Defaults to zero.
Note
The function documented here is a subset of adiag of package magic

Author(s)
Robin K. S. Hankin

Examples
data(toys)
blockdiag(D1.toy,D2.toy)

C1 Matrix of distances from D1 to D2

Description
Returns a matrix of distances from the code run points to the augmented observation points. A wrapper for c1.fun().

Usage
C1(D1, D2, theta, phi)

Arguments
D1 D1
D2 D2
theta Parameters
phi Hyperparameters

Author(s)
Robin K. S. Hankin

References
See Also

t.fun

Examples

data(toys)
C1(D1=D1.toy, D2=D2.toy, theta=theta.toy, phi=phi.toy)

Description

Covariance function for posterior distribution of \( z(\cdot) \) conditional on estimated hyperparameters and calibration parameters \( \theta \).

Usage

\[
\text{Cov.eqn9.supp}(x, \text{xdash=NULL, theta, d, D1, D2, H1, H2, phi})
\]
\[
\text{cov.p5.supp}(x, \text{xdash=NULL, theta, d, D1, D2, H1, H2, phi})
\]

Arguments

- **x**: first point, or (Cov.eqn9.supp()) a matrix whose rows are the points of interest
- **xdash**: The second point, or (Cov.eqn9.supp()) a matrix whose rows are the points of interest. The default of NULL means to use xdash=x
- **theta**: Parameters. For Cov.eqn9.supp(), supply a vector which will be interpreted as a single point in parameter space. For cov.p5.supp(), supply a matrix whose rows will be interpreted as points in parameter space
- **d**: Observed values
- **D1**: Code run design matrix
- **D2**: Observation points of real process
- **H1**: Basis function for D1
- **H2**: Basis function for D2
- **phi**: Hyperparameters

Details

Evaluates the covariance function: the last formula on page 5 of the supplement. The two functions documented here are vectorized differently.

Function Cov.eqn9.supp() takes matrices for arguments x and xdash and a single vector for theta. Evaluation is thus taken at a single, fixed value of theta. The function returns a matrix whose rows correspond to rows of x and whose columns correspond to rows of xdash.
Function `cov.p5.supp()` takes a vector for arguments `x` and `xdash` and a matrix for argument `theta` whose rows are the points in parameter space. A vector `V`, with elements corresponding to the rows of argument `theta` is returned:

\[ V[i] = \text{cov}(z(x), z(x')|\theta_i) \]

**Value**

Returns a matrix of covariances

**Note**

May return the transpose of the desired object

**Author(s)**

Robin K. S. Hankin

**References**


**Examples**

data(toys)
x <- rbind(x.toy,x.toy+1,x.toy,x.toy,x.toy)ownames(x) <- letters[1:5]
xdash <- rbind(x*2,x.toy)ownames(xdash) <- LETTERS[1:6]

Cov.eqn9.supp(x=x,xdash=xdash,theta=theta.toy,d=d.toy,D1=D1.toy, D2=D2.toy,H1=H1.toy,H2=H2.toy,phi=phi.toy)

phi.true <- phi.true.toy(phi=phi.toy)

Cov.eqn9.supp(x=x,xdash=xdash,theta=theta.toy,d=d.toy,D1=D1.toy, D2=D2.toy,H1=H1.toy,H2=H2.toy,phi=phi.true)

# Now try a sequence of thetas:
cov.p5.supp(x=x.toy,theta=t.vec.toy,d=d.toy,D1=D1.toy,D2=D2.toy, H1=H1.toy,H2=H2.toy,phi=phi.toy)
create.new.toy.datasets

Create new toy datasets

Description

Creates new toy datasets, by sampling from an explicitly specified multivariate Gaussian distribution whose covariance matrix is that required for a Gaussian process.

Usage

create.new.toy.datasets(D1, D2, export = FALSE)

Arguments

- **export** Boolean, with default FALSE meaning to return toy datasets and TRUE meaning to return, instead, a list of the true values of the parameters
- **D1** D1; set of code run points
- **D2** D2; set of field observation points

Value

Returns a list of three elements:

- y.toy
- z.toy
- d.toy

Note

Because function create.new.toy.datasets() calls computer.model() and model.inadequacy(), the datasets returned are drawn from a multivariate Gaussian distribution which is a Gaussian process.

References


See Also

toys, reality, latin.hypercube
**Examples**

```r
data(toys)
create.new.toy.datasets(D1=D1.toy, D2=D2.toy)
```

---

**D1.fun**

*Function to join x.star to t.vec to give matrix D1*

---

**Description**

Function to join x.star to t.vec to give matrix D1 with correct row- and column- names.

**Usage**

```r
D1.fun(x.star, t.vec)
```

**Arguments**

- `x.star`: Matrix of code run points
- `t.vec`: Matrix of parameter theta values

**Details**

Note that the matrix returned is a D1 matrix: it is a design matrix for code observations as it contains both x and theta

**Author(s)**

Robin K. S. Hankin

**References**


**See Also**

`toys`
### Examples

```r
data(toys)
nj <- extractor.toy(D1.toy)
x.star.toy <- nj$x.star
t.vec.toy <- nj$t.vec
D1.fun(x.star.toy, t.vec.toy)  # both dataframes
D1.fun(x.star.toy, theta.toy) # one dataframe, one vector
D1.fun(x.toy, t.vec.toy)      # one vector, one dataframe
D1.fun(x.toy, theta.toy)     # two vectors
```

---

### D2.fun

**Augments observation points with parameters**

---

### Description

Augments observation points with parameters; will recycle if necessary

### Usage

```r
D2.fun(D2, theta)
```

### Arguments

- **D2**: Observation points
- **theta**: Parameters

### Author(s)

Robin K. S. Hankin

### References


### See Also

- `D1.toy, theta.toy`
**Examples**

```r
data(toys)
D2.fun(D2=D2.toys, theta=theta.toys)
D2.fun(D2=t(x.toys), theta=theta.toys)
D2.fun(D2=D2.toys[1,,drop=FALSE], theta=theta.toys)
```

---

**dists.2frames**  
*Distance between two points*

**Description**

Distance between points specified by rows of two matrices, according to a positive definite matrix. If not specified, the second matrix used is the first.

**Usage**

```r
dists.2frames(a, b=NULL, A=NULL, A.lower=NULL, test.for.symmetry=TRUE)
```

**Arguments**

- **a**  
  First dataframe whose rows are the points

- **b**  
  Second dataframe whose rows are the points; if NULL, use a

- **A**  
  Positive definite matrix; if NULL, a value for A.lower is needed. If a value for A is supplied, use a clear but possibly slower method

- **A.lower**  
  The lower triangular Cholesky decomposition of A (only needed if A is NULL).

- **test.for.symmetry**  
  Boolean, with default TRUE meaning to calculate all element arrays (elegantly), and FALSE meaning to calculate only the upper triangular elements (using loops), which ought to be faster. The value of this argument should not affect the returned value, up to numerical accuracy

**Author(s)**

Robin K. S. Hankin

**References**


**E.theta.toy**

**Description**

Function `E.theta.toy` returns expectation of $H_1(D)$ with respect to $\theta$; `Edash.theta.toy` returns expectation with respect to $E'$. Function `E.theta.toy` also returns information about nonlinear behaviour of $h_1(x, \theta)$. 

**Usage**

```r
E.theta.toy(D2=NULL, H1=NULL, x1=NULL, x2=NULL, phi, give.mean=TRUE)
Edash.theta.toy(x, t.vec, k, H1, fast.but.opaque=FALSE, a=NULL, b=NULL, phi=NULL)
```

**Arguments**

- **D2**: Observation points
- **H1**: Regression function for D1
- **phi**: hyperparameters. Default value of NULL only to be used in `Edash.theta.toy()` when fast.but.opaque is TRUE
- **x**: lat/long point (for `Edash.theta.toy()`) 
- **t.vec**: Matrix whose rows are parameter values (for `Edash.theta.toy()`) 
- **k**: Integer specifying column (for `Edash.theta.toy()`) 
- **give.mean**: In `E.theta.toy()`, Boolean, with default TRUE meaning to return the mean (expectation), and FALSE meaning to return the “variance”
fast.but.opaque

In Edash.theta.toy(), Boolean, with default FALSE meaning to use a slow but clear method. If TRUE, use faster code but parameters a and b must then be specified.

- a: Constant term, needed if fast.but.opaque is TRUE: \((V^{-1} + 2\Omega^{-1})^{-1} V^{-1} m_\theta\). Specifying a in advance saves execution time.

- b: Linear term, needed if fast.but.opaque is TRUE: \(2 (V^{-1} + 2\Omega^{-1})^{-1} \Omega \) (multiplied by \(t[k,] \) in Edash.theta.toy()).

x1: In E.theta.toy(g=F,...), the value of \(x \) in \(h_1(x, \theta)\). The default value is NULL because in simple cases such as that implemented here, the output is independent of \(x1\) and \(x2\).

x2: In E.theta.toy(g=F,...), the value of \(x \) in \(h_1(x, \theta)\).

**Note**

A terse discussion follows; see the calex.pdf vignette and the 1D case study in directory inst/doc/one_dim/ for more details and examples.

Function E.theta.toy(give.mean=FALSE,...) does not return the variance! The matrix returned is a different size from the variance matrix!

It returns the thing that must be added to crossprod(E_theta(h1(x,theta)),t(E_theta(h1(x,theta)))) to give E_theta(h1(x,theta)).t(h1(x,theta))).

In other words, it returns E_theta(h1(x,theta)).t(h1(x,theta))-crossprod(E_theta(h1(x,theta)),t(E_theta(h1(x,theta))))

If the terms of h1() are of the form \(c(o, \theta)\) (where \(o\) is a vector that is a function of \(x\) alone, and independent of \(\theta\), then the function will include the variance matrix, in the lower right corner (zeroes elsewhere).

Function E.theta() must be updated if h1.toy() changes: unlike E.theta() and Edash.theta(), it does not “know” where the elements that vary with \(\theta\) are, nor their (possibly \(x\)-dependent) coefficients.

This form of the function requires \(x1\) and \(x2\) arguments, for good form’s sake, even though the returned value is independent of \(x\) in the toy example. To see why it is necessary to include \(x\), consider a simple case with \(h_1(x, \theta) = (1, x \theta)^T\). Now \(E_\theta(h(x, \theta))\) is just \((1, x \theta)^T\) but

\[
E_\theta \left( h_1(x, \theta)h_1(x, \theta)^T \right)
\]

is a 2-by-2 matrix \((M, \text{say})\) with \(E_\theta(M) = h_1(x, \theta)h_1(x, \theta)^T + \text{variance terms.}

\[
E_\theta \begin{pmatrix}
1 & x \theta \\
x \theta & x^2 \theta^2
\end{pmatrix}
\]

All three functions here are intimately connected to the form of h1.toy() and changing it (or indeed H1.toy()) will usually require rewriting all three functions documented here. Look at the definition of E.theta.toy(give=F), and you will see that even changing the meat of h1.toy() from \(c(1,x)\) to \(c(x,1)\) would require a redefinition of E.theta.toy(g=F).

The only place that E.theta.toy(g=F) is used is internally in hh.fun().
Author(s)
Robin K. S. Hankin

References


See Also

toys

Examples

data(toys)
E.theta.toy(D2=D2.toy, H1=H1.toy, phi=phi.toy)
E.theta.toy(D2=D2.toy[,1], H1=H1.toy, phi=phi.toy)
E.theta.toy(D2=x.toy, H1=H1.toy, phi=phi.toy)
Edash.theta.toy(x=x.toy, t.vec=t.vec.toy,k=1, H1=H1.toy, phi=phi.toy)

--
EK.eqn10.supp Posterior mean of K
--

Description

Estimates the posterior mean of K as per equation 10 of KOH2001S, section 4.2

Usage

EK.eqn10.supp(X.dist, D1, D2, H1, H2, d, hbar.fun,
lower.theta, upper.theta, extractor, give.info=FALSE,
include.prior=FALSE, phi, ...)

Arguments

X.dist Probability distribution of X, in the form of a two-element list. The first element is the mean (which should have name “mean”), and the second element is the variance matrix, which should be a positive definite matrix of the correct size, and have name “var”
D1 Matrix whose rows are the code run points
D2 Matrix whose rows are field observation points
Regression function for $d_1$

Regression function for $d_2$

Vector of code outputs and field observations

Boolean; passed to function p.eqn8 supp() (qv)

Function that gives expectation (with respect to $x$) of $h_1(x, \theta)$ and $h_2(x)$ as per section 4.2

Lower integration limit for $\theta$ (NB: a vector)

Lower integration limit for $\theta$ (NB: a vector)

Extractor function; see extractor.toy() for an example

Boolean, with default FALSE meaning to return just the answer and TRUE to return the answer along with all output from both integrations as performed by adaptIntegrate()

Hyperparameters

Extra arguments passed to the integration function. If multidimensional (ie length(\theta)>1), then the arguments are passed to adaptIntegrate(); if one dimensional, they are passed to integrate()

Details

This function evaluates a numerical approximation to equation 10 of section 4.2 of the supplement. Equation 10 integrates over the prior distribution of $\theta$. If $\theta$ is a vector, multidimensional integration is necessary.

In the case of multidimensional integration, function adaptIntegrate() is used.

In the case of one dimensional integration—$\theta$ being a scalar—function integrate() of the stats package is used.

Note that equation 10 is conditional on the observed data and the hyperparameters

Value

Returns a scalar

Note

The function was not reviewed by the Journal of Statistical Software.

The adapt package is no longer available on CRAN: so the adapt() function is not available either.

You may be able to install the adapt package notwithstanding its availability on CRAN or is license.

If you are happy with this (I am), install the adapt package and everything should work.

I am working on providing a replacement for adapt(), but this is low on my list of priorities. Sorry about this.

Author(s)

Robin K. S. Hankin
References


Examples

```r
1+1
## Not run:
# Not run because: (i) it takes R CMD check too long, and (ii) a working
# version needs adapt(), which is not currently available
data(toys)
EK.eqn10.supp(X.dist=X.dist.toy, D1=D1.toy, D2=D2.toy,
               H1=H1.toy, H2=H2.toy, d=d.toy,
               hbar.fun=hbar.fun.toy, lower.theta=c(-3,-3,-3),
               upper.theta=c(3,3,3), extractor=extractor.toy,
               phi=phi.toy)
## End(Not run)
```

**etahat**  
*Expectation of computer output*

Description

Returns the a-postori expectation of the computer program at a particular point with a particular set of parameters, given the code output.

Usage

```r
etahat(D1, D2, H1, y, E.theta, extractor, phi)
```

Arguments

- **D1**  
  Matrix of code observation points and parameters
- **D2**  
  Matrix of field observation points
- **H1**  
  Basis functions
- **y**  
  Code observations corresponding to rows of D1
- **E.theta**  
  expectation wrt theta; see details
- **extractor**  
  Extractor function
- **theta**  
  Parameters
- **phi**  
  Hyperparameters
Details

Argument E.theta is officially a function that, given x,y returns $E_\theta(h_1(x, \theta))$.
However, if supplied a non-function (this is tested by is.function() in the code), E.theta is interpreted as values of \( \theta \) to use. Recycling is carried out by function D1.fun().

Author(s)

Robin K. S. Hankin

References


See Also

p.page4

Examples

data(toys)

etahat(D1=D1.toy, D2=D2.toy, H1=H1.toy, y=y.toy, E.theta=E.theta.toy, extractor=extractor.toy, phi=phi.toy)

# Now try giving E.theta=1:3, which will be interpreted as a value for theta:
etahat(D1=D1.toy, D2=D2.toy, H1=H1.toy, y=y.toy, E.theta=1:3, extractor=extractor.toy, phi=phi.toy)

extractor.toy  Exports lat/long matrix and theta matrix from D2.

Description

Extracts x.star.toy and t.vec.toy from D2; toy example needed because the extraction differs from case to case.

Usage

extractor.toy(D1)
Arguments

D1 Matrix of code run points

Details

The first two columns give the elements of \( x.\text{star} \) and columns 3 through 5 give the elements of \( t.\text{vec} \).

Function extractor.toy is the inverse of function D1.fun, in the sense that extractor.toy splits up D1 into \( x.\text{star} \) and \( t.\text{vec} \), while D1.fun joins them up again.

Value

Returns a list with two elements:

- \( x.\text{star} \) A matrix containing the lat/longs of the code run points
- \( t.\text{vec} \) A matrix containing the parameters used for the code runs

Author(s)

Robin K. S. Hankin

References


See Also
toys, D1.fun

Examples

data(toys)
extractor.toy(D1.toy)
extractor.toy(D1.toy[1,,drop=FALSE])
(jj <- extractor.toy(D1.fun(x.star=x.toy, t.vec=theta.toy)))
D1.fun(jj$x.star,jj$t.vec)
Expectation of $z$ given $y$, $\beta_2$, $\phi$

Description

Expectation as per equation 7 on the supplement

Usage

```r
ez.eqn7.supp(z, D1, H1, D2, H2, extractor, beta2, y, E.theta, phi)
```

Arguments

- `z`: Vector of observations
- `D1`: Matrix whose rows are code run points
- `H1`: Regressor basis functions
- `D2`: Matrix whose rows are observation points
- `H2`: Regressor basis functions
- `extractor`: Function to split $D_1$
- `beta2`: coefficients
- `y`: Code outputs at points corresponding to rows of $D_1$
- `E.theta`: Expectation function to use
- `phi`: hyperparameters

Author(s)

Robin K. S. Hankin

References


See Also

- `V.fun`
Examples

data(toys)
etahat.d2 <- etahat(D1=D1(toy), D2=D2(toy), H1=H1(toy), y=y(toy),
  E.theta=E.theta(toy), extractor=extractor(toy), phi=phi(toy))
beta2 <- beta2hat.fun(D1=D1(toy), D2=D2(toy), H1=H1(toy), H2=H2(toy), V=V(toy), z=z(toy),
etahat.d2=etahat.d2, extractor=extractor(toy), E.theta=E.theta(toy),
  Edash.theta=Edash.theta(toy), phi=phi(toy))
Ez.eqn7.supp(z=z(toy),
  D1=D1(toy), H1=H1(toy), D2=D2(toy), H2=H2(toy),
  extractor=extractor(toy), beta2=beta2, y=y(toy),
  E.theta=E.theta(toy),
  phi=phi(toy))

---

Ez.eqn9.supp

Expectation as per equation 10 of KOH2001

Description

Expectation as per equation 10 of KOH2001 (not the supplement)

Usage

Ez.eqn9.supp(x, theta, d, D1, D2, H1, H2, phi)
Ez.eqn9.supp.vector(x, theta, d, D1, D2, H1, H2, phi)

Arguments

- x: point at which expectation is needed
- theta: parameters
- d: observations and code outputs
- D1: code run points
- D2: observation points
- H1: regression function for D1
- H2: regression function for D2
- phi: hyperparameters

Details

The user should always use Ez.eqn9.supp(), which is a wrapper for Ez.eqn9.supp.vector(). The forms differ in their treatment of \( \theta \). In the former, \( \theta \) must be a vector; in the latter, \( \theta \) may be a matrix, in which case Ez.eqn9.supp.vector() is applied to the rows.

Note that Ez.eqn9.supp.vector() is vectorized in x but not \( \theta \) (if given a multi-row object, apply(theta,1,...) is used to evaluate the function for each row supplied).

Function Ez.eqn9.supp() will take multiple-row arguments for x and theta. The output will be a matrix, with rows corresponding to the rows of x and columns corresponding to the rows of theta. See the third example below.
Note that function `ez.eqn9.supp()` determines whether there are multiple values of $\theta$ by `is.vector(theta)`. If this returns `TRUE`, it is assumed that $\theta$ is a single point in multidimensional parameter space; if `FALSE`, it is assumed to be a matrix whose rows correspond to points in parameter space.

So if $\theta$ is one dimensional, calling `ez.eqn9.supp()` with a vector-valued $\theta$ will fail because the function will assume that $\theta$ is a single, multidimensional, point. To get round this, use `as.matrix(theta)`, which is not a vector; the rows are the (1D) parameter values.

**Author(s)**

Robin K. S. Hankin

**References**


**See Also**

tee

**Examples**

data(toys)
`ez.eqn9.supp(x=x.toy, theta=theta.toy, d=d.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)`

`ez.eqn9.supp(x=D2.toy, theta=t.vec.toy, d=d.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)`

`ez.eqn9.supp(x=x.vec, theta=t.vec.toy, d=d.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)`

---

**H.fun**

*H function*

**Description**

H. See front page of KOHsupp.

**Usage**

`H.fun(theta, D1, D2, H1, H2, phi)`
Arguments

theta parameters
D1 matrix of code run points
D2 matrix of observation points
H1 Regressor function for D1
H2 Regressor function for D2
phi hyperparameters

Author(s)

Robin K. S. Hankin

References


Examples

data(toys)
H.fun(theta=theta.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)
H.fun(theta=theta.toy, D1=D1.toy[1,,drop=FALSE], D2=D2.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)
H.fun(theta=theta.toy, D1=D1.toy[1,,drop=FALSE], D2=D2.toy[1,,drop=FALSE], H1=H1.toy, H2=H2.toy, phi=phi.toy)

H1.toy Basis functions for D1 and D2

Description

Applies basis functions to rows of D1 and D2

Usage

H1.toy(D1)
H2.toy(D2)
Arguments

D1   Matrix of code run points
D2   Matrix of observation points

Value

Returns a matrix whose rows are the basis functions of the code run points or observation points. Function H1.toy() operates on datasets like D1.toy (latlong and parameters) and function H2.toy() operates on datasets like D2.toy (latlong only)

Note

See package goldstein for a less trivial example of h().

Author(s)

Robin K. S. Hankin

References


See Also

D1.toy,

Examples

data(toys)
jj <- extractor.toy(D1.toy)
x.star.toy <- jj$x.star
t.vec.toy <- jj$t.vec
H1.toy(D1=D1.toy)
H1.toy(D1.toy[1,, drop=FALSE])
H1.toy(D1.fun(x.star.toy, theta.toy)[1,, drop=FALSE])
H1.toy(D1.fun(x.star=x.star.toy, t.vec=theta.toy))
H1.toy(D1.fun(x.star=x.star.toy[1,, t.vec=t.vec.toy[1,,]))
H1.toy(D1.fun(x.star=x.star.toy[1,, t.vec=t.vec.toy[1:2,,]))
H2.toy(D2.toy)
H2.toy(t(x.toy))
Description

Basis functions for D1 and D2 respectively.

Usage

\texttt{h1.toy(x)}
\texttt{h2.toy(x)}

Arguments

\texttt{x} \hspace{1cm} \text{Vector of lat/long or lat/long and theta}

Details

Note that \texttt{h1()} operates on a vector: for dataframes, use \texttt{H1.toy()} which is a wrapper for \texttt{apply(D1, 1, h1)}.

NB If the definition of \texttt{h1.toy()} or \texttt{h2.toy()} is changed, then function \texttt{hbar.toy()} must be changed to match. This cannot be done automatically, as the form of \texttt{hbar.toy()} depends on the distribution of \texttt{x}. The shibboleth is whether \(E_X(h)\) commutes with \(h_1()\); it does in this case but does not in general (for example, consider \(h(x, \theta) = c(1, x, x^2)\) and \(X \sim N(m, V)\). Then \(E_X(h(x, \theta))\) will be \((1, m, m^2 + V, \theta)\); note the \(V)\)

Value

Returns basis functions of a vector; in the toy case, just prepend a 1.

Author(s)

Robin K. S. Hankin

References


• M. C. Kennedy and A. O’Hagan 2001. Supplementary details on Bayesian calibration of computer models, Internal report, University of Sheffield. Available at \url{http://www.shef.ac.uk/~stlao/ps/calsup.ps}


See Also

\texttt{H1.toy}
Examples

data(toys)
h1.toy(D1.toy[1,])

Description
A toy example of the expectation of h as per section 4.2

Usage
hbar.fun.toy(theta, X.dist, phi)

Arguments
theta Parameter set
X.dist Distribution of variable inputs X as per section 4.2
phi Hyperparameters

Details
Note that if h1.toy() or h2.toy() change, then hbar.fun.toy() will have to change too; see ?h1.toy for an example in which nonlinearity changes the form of E.theta.toy()

Value
Returns a vector as per section 4.2 of KOH2001S

Author(s)
Robin K. S. Hankin

References

See Also
h1.toy
**Examples**

```r
data(toys)
hbar.fun.toy(theta=theta.toy, X.dist=X.dist.toy, phi=phi.toy)
```

**Description**

Returns TRUE if and only if a matrix is positive definite.

**Usage**

```r
is.positive.definite(a, ...)
```

**Arguments**

- `a` Matrix to be tested
- `...` Extra arguments passed to `eigen()`, such as `symmetric`.

**Details**

A wrapper for `eigen()` (a matrix is positive definite if all its eigenvalues are positive). This function is included for convenience only.

**Author(s)**

Robin K. S. Hankin

**Examples**

```r
is.positive.definite(diag(3), sym=TRUE)
is.positive.definite(diag(6)-0.1)
```

---

**MH**

*Very basic implementation of the Metropolis-Hastings algorithm*

**Description**

Very basic implementation of the Metropolis-Hastings algorithm using a multivariate Gaussian proposal distribution. Useful for sampling from `p.eqn8.supp()`.

**Usage**

```r
MH(n, start, sigma, pi)
```
Arguments

- **n**: Number of samples to take
- **start**: Start value
- **sigma**: Variance matrix for kernel
- **pi**: Functional proportional to the desired sampling pdf

Details

This is a basic implementation. The proposal distribution $q(X|Y) = q(\cdot|X) = N(X, \sigma^2)$

Value

Returns a matrix whose rows are samples from $\pi()$. Note that the first few rows will be “burn-in”, so should be ignored

Note

This function is a little slow because it is not vectorized.

Author(s)

Robin K. S. Hankin

References

- N. Metropolis and others 1953. *Equation of state calculations by fast computing machines*. The Journal of Chemical Physics, volume 21, number 6, pages 1087-1092

See Also

- `p.eqn8.supp`

Examples

```r
# First, a bivariate Gaussian:
A <- diag(3) + 0.7
quad.form <- function(M, x)(drop(crossprod(crossprod(M,x),x)))
pi.gaussian <- function(x)(exp(-quad.form(A/2,x)))
x.gauss <- mh(n=1000, start=c(0,0,0),sigma=diag(3),pi=pi.gaussian)
cov(x.gauss)/solve(A) # Should be a matrix of 1s.

# Now something a bit weirder:
pi.triangle <- function(x){
  1*as.numeric( (abs(x[1])<1.0) & (abs(x[2])<1.0) ) +
  5*as.numeric( (abs(x[1])<0.5) & (abs(x[2])<0.5) ) *
  as.numeric(x[1]>x[2])
}
```
x.tri <- MH(n=100, start=c(0,0), sigma=diag(2), pi=pi.triangle)
plot(x.tri, main="Try with a higher n")

# Now a Gaussian mixture model:
pi.2gauss <- function(x){
expt(-quad.form(A/2, x)) +
expt(-quad.form(A/2, x+c(2, 2, 2)))
}
x.2 <- MH(n=100, start=c(0,0,0), sigma=diag(3), pi=pi.2gauss)
## Not run: p3d(x.2, theta=44, d=1e4, d0=1, main="Try with more points")

---

**Aposterior probability of psi1**

**Description**

Gives the probability of \( \psi_1 \), given observations. Equation 4 of the supplement

**Usage**

```r
p.eqn4.supp(D1, y, H1, include.prior=TRUE, lognormally.distributed, return.log, phi)
```

**Arguments**

- `D1` Matrix of code run points
- `y` Vector of code outputs
- `H1` Regression function
- `include.prior` Boolean with default TRUE meaning to return the likelihood multiplied by the aprior probability and FALSE meaning to return the likelihood without the prior.
- `lognormally.distributed` Boolean; see ?prob.theta for details
- `return.log` Boolean, with default FALSE meaning to return the probability and TRUE meaning to return the logarithm of the probability
- `phi` hyperparameters

**Author(s)**

Robin K. S. Hankin
References


See Also

W1

Examples

```r
data(toys)
p.eqn4 supp(D1=D1 toy, y=y toy, H1 = H1 toy, lognormally.distributed=TRUE, phi=phi toy)
```

### Description

Function to determine the a-posteriori probability of hyperparameters $\rho$, $\lambda$ and $\psi_2$, given observations and $\psi_1$.

### Usage

```r
p.eqn8 supp(theta, D1, D2, H1, H2, d, include.prior=FALSE, lognormally.distributed=FALSE, return.log=FALSE, phi)
p.eqn8 supp.vector(theta, D1, D2, H1, H2, d, include.prior=FALSE, lognormally.distributed=FALSE, return.log=FALSE, phi)
```

### Arguments

- **theta**: Parameters
- **D1**: Matrix of code run points
- **D2**: Matrix of observation points
- **H1**: Regression function for D1
- **H2**: Regression function for D2
- **d**: Vector of code output values and observations
- **include.prior**: Boolean, with TRUE meaning to include the prior PDF for $\theta$ and default FALSE meaning return the likelihood, multiplied by an undetermined constant
lognormally distributed
   Boolean, with TRUE meaning to assume prior is lognormal (see prob.theta() for more info)

return.log
   Boolean, with default FALSE meaning to return the probability; TRUE means to return the (natural) logarithm of the answer

phi
   Hyperparameters

Details

The user should always use p.eqn8.supp(), which is a wrapper for p.eqn8.supp.vector(). The forms differ in their treatment of \( \theta \). In the former, \( \theta \) must be a vector; in the latter, \( \theta \) may be a matrix, in which case p.eqn8.supp.vector() is applied to the rows

Author(s)

Robin K. S. Hankin

References


See Also

W2, stage1

Examples

data(toys)
p.eqn8.supp(theta=theta.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, d=d.toy, phi=phi.toy)

   ## Now try using the true hyperparameters, and data directly drawn from
   ## the appropriate multivariate distn:

   phi.true <- phi.true.toy(phi=phi.toy)
   jj <- create.new.toy.datasets(D1.toy, D2.toy)
   d.toy <- jj$d.toy
   p.eqn8.supp(theta=theta.toy, D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, d=d.toy, phi=phi.true)

   ## Now try p.eqn8.supp() with a vector of possible thetas:
   p.eqn8.supp(theta=sample.theta(n=11,phi=phi.true), D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, d=d.toy, phi=phi.true)
A posteriori probability of hyperparameters

Description

Function to determine a posteriori probability of hyperparameters $\rho$, $\lambda$, and $\psi_2$, given observations and $\psi_1$.

Usage

p.posterior(D1, D2, H1, H2, V, y, z, E.theta, Edash.theta, extractor, include.prior=FALSE, lognormally.distributed=FALSE, return.log=FALSE, phi)

Arguments

- **D1**: Matrix of code run points
- **D2**: Matrix of observation points
- **H1**: Basis function (vectorized)
- **H2**: Regression function for D2
- **V**: Covariance matrix; default value of NULL results in the function evaluating it (but this takes a long time, so supply V if known)
- **y**: Vector of code outputs
- **z**: Vector of observation values
- **E.theta**: Expectation over theta
- **Edash.theta**: Expectation over theta WRT $E'$$\theta$
- **extractor**: Function to extract independent variables and parameters from D1
- **include.prior**: Boolean, with TRUE meaning to include the prior PDF for $\theta$ and default value of FALSE meaning to return the likelihood multiplied by an undetermined constant
- **lognormally.distributed**: Boolean with TRUE meaning to assume lognormality. See prob.psi1 for details
- **return.log**: Boolean, with default FALSE meaning to return the probability, and TRUE meaning to return the (natural) logarithm of the probability (which is useful when considering very small probabilities)
- **phi**: Hyperparameters

Author(s)

Robin K. S. Hankin
References


See Also

W2

Examples

data(toys)

```r
p.page4(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, V=NULL, y=y.toy, 
  z=z.toy,E.theta=E.theta.toy, Edash.theta=Edash.theta.toy, extractor=extractor.toy, phi=phi.toy)
```

```r
# Now compare the above value with p.page4() calculated with phi
# differing only in psi2:

```r
phi.toy.new <- phi.change(phi.fun=phi.fun.toy, old.phi = phi.toy, psi2=c(8,8,8))
```

```r
p.page4(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, V=v.toy, y=y.toy, z=z.toy, 
  E.theta=E.theta.toy, Edash.theta=Edash.theta.toy, extractor=extractor.toy, phi=phi.toy.new)
```

## different!

---

**phi.fun.toy**

*Functions to create or change hyperparameters*

**Description**

Function to create (phi.fun.toy) or modify (phi.change) toy hyperparameters $\phi$ in a form suitable for passing to the other functions in the library.

The user should never make $\phi$ by hand; always use one of these functions

**Usage**

```r
phi.fun.toy(rho, lambda, psi1, psi1.apriori, psi2, psi2.apriori, 
  theta.apriori)
```

```r
phi.change(phi.fun, old.phi = NULL, rho = NULL, lambda = NULL, 
  psi1 = NULL, psi1.apriori=NULL, psi1.apriori.mean=NULL, 
  psi1.apriori.sigma=NULL, psi2 = NULL, psi2.apriori=NULL, 
```
psi2.apriori.mean=NULL, psi2.apriori.sigma=NULL,
theta.apriori=NULL, theta.apriori.mean=NULL,
theta.apriori.sigma=NULL)

Arguments

phi.fun
In phi.change(), the name of the function that creates the hyperparameters.
Use phi.fun.toy() for the toy dataset

old.phi
In function phi.change(), the hyperparameter object φ to be modified

rho
Correlation hyperparameter appearing in main equation

lambda
Noise hyperparameter

psi1
Roughness lengths hyperparameter for design matrix D1. Internal function pdm.maker.psi1() takes psi1 as an argument and returns omega_x, omega_t and sigma2squared. Recall that Ω_x and Ω_y are arbitrary functions of ψ_1. In this case, the values are omega_x=psi1[1:2], omega_t=psi1[3:4] and sigma2squared=psi1[6]

psi1.apriori A priori PDF for ψ_1. In the form of a two element list with first element (mean) the mean and second element (sigma) the covariance matrix; distribution of the logarithms is assumed to be multivariate normal. In the toy example, the mean is a vector of length six (the first five are ψ_1 and the sixth is for σ_1^2), and the variance is the corresponding six-by-six matrix. Use function prob.psi1() to calculate the apriori probability density for a particular value of ψ_1

psi1.apriori.mean
In function phi.change.toy(), use this argument to change just the mean of psi1 (and leave the value of sigma unchanged)

psi1.apriori.sigma
In function phi.change.toy(), use this argument to change just the variance matrix of psi1

psi2
Roughness lengths hyperparameter for D2.
Internal function pdm.maker.psi2() takes psi2 as an argument and returns omegastar_x and sigma2squared. In phi.fun.toy(), the values are omegastar_x=psi2[1:2] and sigma2squared=psi2[3].
NB: function stage2() optimizes more than just psi2. It simultaneously optimizes psi2 and lambda and rho

psi2.apriori A priori PDF for ψ_2 and hyperparameters ρ and λ (in that order).
As for psi1.apriori, this is in the form of a list with the first element (mean) the mean and second element (sigma) the covariance matrix; the logs are multivariate normal. In the toy example, the mean is a vector of length five. The first and second elements of the mean are the apriori mean of ρ and λ respectively; the third and fourth elements are the apriori mean of ψ_2 (that is, x and y respectively); and the fifth is the mean of σ_2^2.
The second element of phi.toy$psi2.apriori, sigma, is the corresponding four-by-four variance matrix. Use function prob.psi2() to calculate the apriori probability density of a particular value of ψ_2

psi2.apriori.mean
In phi.change.toy(), use to change just the mean of psi2
psi2.apriori.sigma
   In `phi.change.toy()`, use to change just the variance matrix of psi2
theta.apriori
   Apriori PDF for $\theta$. As above, in the form of a list with elements for the mean and covariance. The distribution is multivariate normal (NB: The distribution is multivariate normal and NOT lognormal! To be explicit: $\log(\theta)$ is lognormally distributed). Use function `prob.theta()` to calculate the apriori probability density of a particular value of $\theta$
theta.apriori.mean
   In `phi.change.toy()`, use to change just the mean of $\theta$
theta.apriori.sigma
   In `phi.change.toy()`, use to change just the variance matrix of $\theta$

Details

Note that this toy function contains within itself `pdm.maker.toy()` which extracts $\omega_x$ and $\omega_t$ and `sigmaQsquared` from `psiQ`. This will need to be changed for real-world applications. Earlier versions of the package had `pdm.maker.toy()` defined separately.

Value

Returns a list of several elements:

- `rho`: Correlation hyperparameter
- `lambda`: Noise hyperparameter
- `psi1`: Roughness lengths hyperparameter for $dQ$
- `psi1.apriori`: Apriori mean and variance matrix for `psi1`
- `psi2`: Roughness lengths hyperparameter for $dR$
- `psi2.apriori`: Apriori mean and variance matrix for `psi2`
- `theta.apriori`: Apriori mean and variance matrix for the parameters
- `omega_x`: Positive definite matrix for the lat/long part of $dQ$, whose diagonal is `psiQ[1:2]`
- `omega_t`: Positive definite matrix for the code parameters `theta`, whose diagonal is `psiQ[3:5]`
- `omegastar_x`: Positive definite matrix for use in equation 13 of the supplement; represents distances between rows of $dR$
- `sigmaQsquared`: Variance
- `sigmaRsquared`: Variance
- `omega_x.upper`: Upper triangular Cholesky decomposition for `omega_x`
- `omega_x.lower`: Lower triangular Cholesky decomposition for `omega_x`
- `omega_t.upper`: Upper triangular Cholesky decomposition for `omega_t`
- `omega_t.lower`: Lower triangular Cholesky decomposition for `omega_t`
- `a`: Precalculated matrix for use in `Edash.theta(..., fast.but.opaque=TRUE)`
- `b`: Precalculated matrix for use in `Edash.theta(..., fast.but.opaque=TRUE)`
- `c`: Precalculated scalar for use in `ht.fun(..., fast.but.opaque=TRUE)`
- `A`: Precalculated scalar for use in `tt.fun()`
- `A.upper`: Upper triangular Cholesky decomposition for `A`
- `A.lower`: Lower triangular Cholesky decomposition for `A`
Author(s)

Robin K. S. Hankin

References


See Also
toys, H1.toy

Examples

```r
phi.fun.toy(100, 101, 1:6, list(mean=rep(1,6), sigma=1+diag(6)), 50:55,
list(mean=rep(0,4), sigma=0.1+diag(4)),
list(mean=0.1+(1:3), sigma=2.1+diag(3)))
phi.fun.toy(rho=1, lambda=1,
  psi1 = structure(c(1.1, 1.2, 1.3, 1.4, 1.5, 0.7),
    .Names = c("x", "y", "A", "B", "C", "s1sq"),
  psi1.aprioi = list(
    mean=rep(0,6), sigma=0.4+diag(6)),
  psi2=structure(c(2.1, 2.2), .Names = c("x", "y")),
  psi2.aprioi = list(mean=rep(0,5), sigma=0.2+diag(5)),
  theta.aprioi = list(mean=0.1+(1:3), sigma=2.1+diag(3))
)

data(toys)
phi.change(phi.fun=phi.fun.toy, old.phi = phi.toy, rho = 100)
phi.change(phi.fun=phi.fun.toy, old.phi = phi.toy,
  theta.apriori.sigma = 4*diag(3))
identical(phi.toy, phi.change(phi.fun=phi.fun.toy, old.phi=phi.toy))
```

---

prob.psi1

A priori probability of psi1, psi2, and theta

Description

Function to determine the a-priori probability of $\psi_1$ and $\psi_2$ of the hyperparameters, and $\theta$, given the apriori means and standard deviations.

Function sample.theta() samples $\theta$ from its prior distribution.
Usage

\[
\text{prob.psi1}(\phi, \text{lognormally.distributed}=\text{TRUE}) \\
\text{prob.psi2}(\phi, \text{lognormally.distributed}=\text{TRUE}) \\
\text{prob.theta}(\theta, \phi, \text{lognormally.distributed}=\text{FALSE}) \\
\text{sample.theta}(n=1, \phi)
\]

Arguments

- \(\phi\)  
  Hyperparameters
- \(\theta\)  
  Parameters
- \text{lognormally.distributed}  
  Boolean variable with \text{FALSE} meaning to assume a Gaussian distribution and \text{TRUE} meaning to use a lognormal distribution.
- \(n\)  
  In function \(\text{sample.theta()}\), the number of observations to take

Details

These functions use package \text{mvtnorm} to calculate the probability density under the assumption that the PDF is lognormal. One implication would be that \(\phi{\psi}_2.\text{apriori}\_\text{mean}\) and \(\phi{\psi}_1.\text{apriori}\_\text{mean}\) are the means of the logarithms of the elements of \(\psi_1\) and \(\psi_2\) (which are thus assumed to be positive). The \text{sigma} matrix is the covariance matrix of the logarithms as well.

In these functions, interpretation of argument \(\phi\) depends on the value of Boolean argument \text{lognormally.distributed}. Take \text{prob.theta()} as an example. If \text{lognormally.distributed} is \text{TRUE}, then \(\log(\theta)\) is normally distributed with mean \(\phi{\theta}_\text{apriori}\_\text{mean}\) and variance \(\phi{\theta}_\text{apriori}\_\text{sigma}\). If \text{FALSE}, \(\theta\) is normally distributed with mean \(\phi{\theta}_\text{apriori}\_\text{mean}\) and variance \(\phi{\theta}_\text{apriori}\_\text{sigma}\).

Interpretation of \(\phi{\theta}_\text{apriori}\_\text{mean}\) depends on the value of \text{lognormally.distributed}: if \text{TRUE} it is the expected value of \(\log(\theta)\); if \text{FALSE}, it is the expectation of \(\theta\).

The reason that \text{prob.theta} has a different default value for \text{lognormally.distributed} is that some elements of \(\theta\) might be negative, contraindicating a lognormal distribution.

Author(s)

Robin K. S. Hankin

References

See Also

p.eqn4.supp, stage1.p.eqn8.supp

Examples

data(toys)
prob.psi1(phi=phi.toy)
prob.psi2(phi=phi.toy)

prob.theta(theta=theta.toy,phi=phi.toy)

sample.theta(n=4,phi=phi.toy)

<table>
<thead>
<tr>
<th>reality</th>
<th>Reality</th>
</tr>
</thead>
</table>

Description

Function to compute reality, gratis *deus ex machina*. Includes a simple computer model that substitutes for a complex climate model, and a simple function that substitutes for the base system, in this case the climate.

Usage

model.inadequacy(X, set.seed.to.zero=TRUE, draw.from.prior=FALSE, export.true.hyperparameters=FALSE, phi=NULL)
computer.model(X, params=NULL, set.seed.to.zero=TRUE, draw.from.prior=FALSE, export.true.hyperparameters=FALSE, phi=NULL)
phi.true.toy(phi)

Arguments

- **X**  
  Observation point

- **params**  
  Parameters needed by computer.model()

- **set.seed.to.zero**  
  Boolean, with the default value of TRUE meaning to set the RNG seed to zero

- **draw.from.prior**  
  Boolean, with default FALSE meaning to generate observations from the “true” values of the parameters, and TRUE meaning to draw from the relevant apriori distribution.

- **export.true.hyperparameters**  
  Boolean, with default value of FALSE meaning to return the observed scalar. Set to TRUE to exercise omniscience and access the true values of the parameters and hyperparameters. Only the omnipotent should set this variable, and only the omniscient may see its true value.
phi

In function phi.true.toy() the hyperparameters $\phi$. Note that apriori distributions are unchanged (they are irrelevant to omniscient beings).
In functions reality() and computer.model(), the prior distributions of the hyperparameters is passed via phi (so it only elements psi1.apriori and psi2.apriori need to be set).

Details

Function reality() provides the scalar value observed at a point $x$. Evaluation expense is zero; there is no overhead.

(However, it does not compute “reality”: the function returns a value subject to observational error $N(0, \lambda)$ as per equation 5. It might be better to call this function observation().)

Function computer.model() returns the output of a simple, nonlinear computer model.
Both functions documented here return a random variable drawn from an appropriate (correlated) multivariate Gaussian distribution, and are thus Gaussian processes.
The approach is more explicit in the help pages of the emulator package. There, Gaussian processes are generated by directly invoking rmvnorm() with a suitable correlation matrix.

Author(s)

Robin K. S. Hankin

References


See Also

computer.model

Examples

data(toys)

computer.model(X=D2.toy, params=theta.toy)
computer.model(D1.toy)
computer.model(X=x.toy, params=extractor.toy(D1.toy)$vec)

phi.fix <- phi.change(old.phi=phi.toy,
psil=c(1, 0.5, 1, 1, 0.5, 0.4), phi.fun=phi.fun.toy)
# The values come from c(REAL.SCALES, REAL.SIGMA1SQUARED) as
# seen in the sourcecode for computer.model().

computer.model(D1.toy)  # use debug(computer.model) and examine
# var.matrix directly. It should match the
# output from V1():

# first fix phi so that it has the correct values for psil (see the
# section on psil in _phi.fun.toy for how to get this):

phi.fix <- phi.change(old.phi=phi.toy, psil=c(1, 0.5, 1.0, 1.0, 0.5,
0.4), phi.fun=phi.fun.toy)
V1(D1.toy, phi=phi.fix)

# What are the hyperparameters that were used to create reality?
phi.true.toy(phi=phi.toy)

#
computer.model(X=D2.toy, params=theta.toy, draw.from.prior=TRUE, phi=phi.toy)

---

stage1  
Stage 1, 2 and 3 optimization on toy dataset

Description

Perform O’Hagan’s three stage optimization on the toy dataset. Function stage1() and stage2() find the optimal values for the hyperparameters and stage3() finds the optimal values for the three parameters.

Usage

stage1(D1, y, H1, maxit, trace=100, method="Nelder-Mead",
directory = ".", do.filewrite=FALSE, do.print=TRUE,
phi.fun, lognormally.distributed=FALSE, include.prior=TRUE, phi)
stage2(D1, D2, H1, H2, y, z, maxit, trace=100, method = "Nelder-Mead",
directory = ".", do.filewrite=FALSE, do.print=TRUE, extractor,
phi.fun, E.theta, Edash.theta, isotropic=FALSE,
lognormally.distributed = FALSE, include.prior = TRUE,
use.standin = FALSE, rho.eq.1 = TRUE, phi)
stage3(D1, D2, H1, H2, d, maxit, trace=100, method="Nelder-Mead",
directory = ".", do.filewrite=FALSE, do.print=TRUE,
include.prior = TRUE, lognormally.distributed=FALSE, theta.start=NULL, phi)

Arguments

maxit Maximum number of iterations as passed to optim()
trace Amount of information displayed, as passed to optim()
D1 Matrix whose rows are points at which code output is known
D2 Matrix whose rows are points at which observations were made
H1,H2 Regressor basis functions for D1 and D2
y Code outputs. Toy example is y.toy
z Observations. Toy example is z.toy
d Data vector consisting of the code runs and observations
extractor extractor function for D1
E.theta,Edash.theta Expectation WRT theta, and dashed theta. Toy examples are E.theta.toy() and Edash.theta.toy()
phi.fun Function to create hyperparameters; passed (in stage1() and stage2()) to phi.change(). Toy version is phi.fun.toy()
method Method argument passed to optim(); qv
include.prior Boolean variable with default TRUE meaning to include the prior distribution in the optimization process and FALSE meaning to use an uninformative prior (effectively uniform support). This variable is passed to p.eqn4.supp() for stage1(), p.page4() for stage2(), and p.eqn8.supp() for stage3()
lognormally.distributed Boolean with TRUE meaning to use a lognormal distn. See prob.theta for details
do.filewrite Boolean, with TRUE meaning to save a loadable file stage[123].<date>, containing the interim value of phi and the corresponding optimand to directory at each evaluation of the optimizer. If FALSE, don’t write the files
directory The directory to write files to; only matters if do.filewrite is TRUE
isotropic In function stage2(). Boolean with default FALSE meaning to carry out a full optimization, and TRUE meaning to restrict the scope to isotropic roughness matrices. See details section below
do.print Boolean, with default TRUE meaning to print interim values of phi at each evaluation
use.standin In stage2(), a Boolean argument, with default FALSE meaning to use the real value for matrix V.temp. and TRUE meaning to use a standing that is the same size but contains fictitious values. The only time to set use.standin to TRUE is when debugging as it runs several orders of magnitude faster
rho.eq.1 Boolean, with default TRUE meaning to hold the value of rho constant at one (1)
theta.start In stage3(), the starting point of the optimization with default NULL meaning to use the maximum likelihood point of the apriori distribution (ie phi$theta.apriori$mean)
phi Hyperparameters. Used as initial values for the hyperparameters in the optimization routines
Details

The three functions documented here carry out the multi-stage optimization detailed in KOH2001 (actually, KOH2001 only defined stage 1 and stage 2, which estimated the hyperparameters. What is here called “stage3()” estimates the true value of $\theta$ given the hyperparameters).

stage1() carries out stage 1 of KOH2001 which is used to estimate $\psi_1$ using optimization.

In function stage2(), setting argument isotropic=TRUE will force $\phi_*\omega_{x}$ to be a function of a length one scalar. The value of $\phi_*\omega_{x}$ used will depend on pdm.maker.psi2() (an internal function appearing in hpa.fun.toy()). In stage2(), several kludges are made. The initial conditions are provided by argument phi. The relevant part of this is $\phi_{12}$.

Function stage2() estimates $\psi_2$ and $\rho$ and $\lambda$, using optimization. Note that $\psi_2$ includes $\sigma^2$ in addition to $\omega_{x}$ (in the toy case, $\psi_2$ has three elements: the first two are the diagonal of $\omega_{x}$ and the third is $\sigma^2$ but this information is encoded in phi.fun.toy(), which changes from application to application).

Function stage3() attempts to find the maximum likelihood estimate of $\theta$, given hyperparameters and observations, using optimization.

Author(s)

Robin K. S. Hankin

References


See Also

toys, phi.fun.toy

Examples

data(toys)
stage1(D1=D1.toy, y=y.toy, H1=H1.toy, maxit=5, phi.fun=phi.fun.toy, phi=phi.toy)

## now try with a slightly bigger dataset:
## Examples below take a few minutes to run:

set.seed(0)
data(toys)
jj <- create.new.toy.datasets(D1.toy, D2.toy)
y.toy <- jj$y.toy
z.toy <- jj$z.toy
d.toy <- jj$d.toy
symmetrize

Symmetrize an upper triangular matrix

Description

Symmetrize an upper triangular matrix by copying the upper triangular elements into the lower triangular places.

Usage

symmetrize(a)

Arguments

a

Upper triangular matrix to be symmetrized

Details

Also works for lower triangular matrices

Author(s)

Robin K. S. Hankin

Examples

jj <- matrix(rnorm(50),10,5)
X <- crossprod(jj,jj)  # X has a Wishart distribution (and in # particular is positive definite)
chol(X)
symmetrize(chol(X))
Auxiliary functions for equation 9 of the supplement

Description

Returns a vector whose elements are the “distances” from a point to the observations and code run points (tee()); and basis functions for use in Ez.eqn9_supp()

Usage

tee(x, theta, D1, D2, phi)
h.fun(x, theta, H1, H2, phi)

Arguments

x Point from which distances are calculated
theta Value of parameters
D1,D2 Design matrices of code run points and field observation points respectively (tee())
H1,H2 Basis functions for eta and model inadequacy term respectively (h.fun())
phi Hyperparameters

Details

Equation 9 of the supplement is identical to equation 10 of KOH2001.

Function h.fun() returns the first of the subsidiary equations in equation 9 of the supplement and function tee() returns the second (NB: do not confuse this with functions t1bar() and t2bar() which are internal to EK.eqn10_supp())

Author(s)

Robin K. S. Hankin

References


See Also

Ez.eqn9 supp
Examples

data(toys)
tee(x=x.toy, theta=theta.toy, D1=D1.toy, D2=D2.toy, phi=phi.toy)

# Now some vectorized examples:
jj <- rbind(x.toy, x.toy, x.toy+0.01, x.toy+1, x.toy*10)
tee(x=jj, theta=theta.toy, D1=D1.toy, D2=D2.toy, phi=phi.toy)
h.fun(x=jj, theta=theta.toy, H1=H1.toy, H2=H2.toy, phi=phi.toy)

---

Toys

Toy datasets

Description

Toy datasets that illustrate the package.

Usage

data(toys)
D1.toy
D2.toy
d.toy
phi.toy
theta.toy
V.toy
X.dist.toy

Format

The D1.toy matrix is 8 rows of code run points, with five columns. The first two columns are the lat and long and the next three are parameter values.

The D2.toy matrix is five rows of observations on two variables, x and y which are styled “latitude and longitude”.

d.toy is the “data” vector consisting of length 13: elements 1-8 are code runs and elements 9-13 are observations.

theta.toy is a vector of length three that is a working example of θ. The parameters are designed to work with computer.model().

t.vec.toy is a matrix of eight rows and three columns. Each row specifies a value for θ. The eight rows correspond to eight code runs.

x.toy and x.toy2 are vectors of length two that gives a sample point at which observations may be made (or the code run). The gloss of the two elements is latitude and longitude.

x.vec is a matrix whose rows are reasonable x values but not those in D2.toy.
y.toy is a vector of length eight. Each element corresponds to the output from a code run at each of the rows of D1.toy.

z.toy is a vector of length five. Each element corresponds to a measurement at each of the rows of D2.toy.

V.toy is a five by five variance-covariance matrix for the toy datasets.

X.dist.toy is a toy example of a distribution of X for use in calibrated uncertainty analysis, section 4.2.

**Brief description of toy functions fully documented under their own manpage**

Function create.new.toy.datasets() creates new toy datasets with any number of observations and code runs.

Function E.theta.toy() returns expectation of H(D) with respect to \( \theta \); Edash.theta.toy() returns expectation with respect to \( E' \).

Function extractor.toy() extracts x.star.toy and t.vec.toy from D2; toy example needed because the extraction differs from case to case.

Function H1.toy() applies basis functions to rows of D1 and D2.

Function phi.fun.toy() creates a hyperparameter object such as phi.toy in a form suitable for passing to the other functions in the library.

Function phi.change.toy() modifies the hyperparameter object.

**See the helpfiles listed in the “see also” section below**

**Details**

All toy datasets are documented here. There are also several toy functions that are needed for a toy problem; these are documented separately (they are too diverse to document fully in a single manpage). Nevertheless a terse summary for each toy function is provided on this page. All toy functions in the package are listed under “See Also”.

**Author(s)**

Robin K. S. Hankin

**References**


**See Also**

create.new.toy.datasets, E.theta.toy, extractor.toy, H1.toy, phi.fun.toy, stage1
**Examples**

```r
data(toys)
D1.toy
extractor.toy(D1.toy)

D2.fun(theta=theta.toy, D2=D2.toy)
D2.fun(theta=theta.toy, D2=D2.toy[1], drop=FALSE)

library("emulator")
corr.matrix(D1.toy, scales=rep(1, 5))
corr.matrix(D1.toy, pos.def.matrix=diag(5))
```

---

**tt.fun**

*Integrals needed in KOH2001*

---

**Description**

Calculates the three integrals needed for $V$, under the restrictions specified in the KOH2001 supplement.

**Usage**

```r
tt.fun(D1, extractor, x.i, x.j, test.for.symmetry=FALSE, method=1, phi)
ht.fun(x.i, x.j, D1, extractor, Edash.theta, H1, fast.but.opaque=TRUE, x.star=NULL, t.vec=NULL, phi)
hh.fun(x.i, x.j, H1, E.theta, phi)
t.fun(x, D1, extractor, phi)
```

**Arguments**

- **D1**: Matrix of code run points
- **H1**: regression basis functions for D1
- **extractor**: Function to extract x.star and t.vec from D1
- **x**: Lat and long of a point in t.fun() (eg D2[1,])
- **x.i**: Lat and long of first point (eg D2[1,])
- **x.j**: Lat and long of second point (eg D2[2,])
- **theta**: parameters
- **Edash.theta**: Function to return expectation of $H$ with respect to the alternative distribution of $\theta$; Edash.theta.toy is the example for the toy dataset
- **E.theta**: Function to return expectation of $H$ with respect to $\theta$
tt.fun

In tt.fun(), Boolean with TRUE meaning to calculate each element of $C$ explicitly. If FALSE, then calculate only the elements of $C$ that lie on or over the diagonal and use the fact that $C$ is symmetric to calculate the other matrix elements. For $n$ observations, this means $n(n+1)/2$ evaluations, compared with $n^2$ for the full case.

Set this argument to TRUE only when debugging, or testing accuracy.

fast.but.opaque

In ht.fun(), Boolean with default TRUE meaning to pass some precalculated results as arguments, to save time. Set this argument to FALSE only when debugging.

x.star In ht.fun(), value of $x^*$ (required only if fast.but.opaque is TRUE)

t.vec In ht.fun(), value of $t$ (required only if fast_but.opaque is TRUE)

method In tt.fun(), zero means use the old method and nonzero means use the new method. The new method is faster, but the code is harder to understand. The two methods should give identical results.

phi Hyperparameters

The four functions return integrals representing means taken over theta. To wit:

- Function tt.fun() evaluates

$$
\int t(x_j, \theta)t(x_i, \theta)^T p(\theta) d\theta
$$

and is used in V.fun(). Note that this function is symmetric in $x_i$ and $x_j$.

- Function ht.fun() evaluates

$$
\int h_1(x_j, \theta)t(x_i, \theta)^T p(\theta) d\theta
$$

and is used in V.fun(). Note that this function is not symmetric in $x_i$ and $x_j$.

- Function hh.fun() evaluates

$$
\int h_1(x_j, \theta)h_1(x_i, \theta)^T p(\theta) d\theta
$$

and is used in V.fun(). Note that this function is symmetric in $x_i$ and $x_j$.

- Function t.fun() evaluates

$$
\int t(x_i, \theta)^T p(\theta) d\theta = \int c_1 \left((x_i, \theta), (x^*_j, t_j)\right) p(\theta) d\theta
$$

using the formula

$$
s^2 \left| I + 2V_\theta \Omega_x \right|^{-1/2} \exp \left\{ - (x_i - x^*_j)^T \Omega_x (x_i - x^*_j) \right\} \times \exp \left\{ - (m_\theta - t_j)^T (2V_\theta + \Omega_x^{-1})^{-1} (m_\theta - t_j) \right\}.
$$

It is used in E2_eq7_supp(). NB: do not confuse this function with tee(), which is different.

These functions are not generally of much interest to the end user; they are called by V.fun(). They are defined separately as a debugging aid, and to simplify the structure of V.fun().
tt.fun

Value

Each function returns a matrix as described in KOH2001

Author(s)

Robin K. S. Hankin

References


See Also

V.fun

Examples

data(toys)

```r
tt.fun(D1=D1.toy, extractor=extractor.toy, x.i=D2.toy[1,], x.j=D2.toy[2,], phi=phi.toy)

ht.fun(x.i=D2.toy[1,], x.j=D2.toy[2,], D1=D1.toy, extractor=extractor.toy, Edash.theta=Edash.theta.toy, H1=H1.toy, fast.but.opaque=FALSE, phi=phi.toy)

ht.fun(x.i=D2.toy[1,], x.j=D2.toy[2,], D1=D1.toy, extractor=extractor.toy, Edash.theta=Edash.theta.toy, H1=H1.toy, fast.but.opaque=TRUE, x.star=extractor.toy(D1.toy)$x.star, t.vec=extractor.toy(D1.toy)$t.vec, phi=phi.toy)

hh.fun(x.i=D2.toy[1,], x.j=D2.toy[2,], H1=H1.toy, E.theta=E.theta.toy, phi=phi.toy)

t.fun(x=x.toy, D1=D1.toy, extractor=extractor.toy, phi=phi.toy)
```
V.fun  

Variance matrix for observations

Description

Determines the variance/covariance matrix for the observations and code run points.

Usage

\[
\text{V.fun}(D_1, D_2, H_1, H_2, \ extractor, \\
E\.theta, Edash\.theta, \text{give.answers=}\text{FALSE}, test\.for\.symmetry=}\text{FALSE, phi})
\]

Arguments

- **D1**: Matrix of code run points
- **D2**: Matrix of observation points
- **H1**: Regression function for D1
- **H2**: Regression function for D2
- **extractor**: Function to extract x.star and t.vec from D1
- **Edash\.theta**: Function to return expectation of H with respect to the alternative distribution of \(\theta\); Edash\.theta\.toy is the example for the toy dataset
- **E\.theta**: Expectation of h WRT theta over the apriori distribution. Note that this function must be updated if h1() changes.
- **give\.answers**: Boolean (defaulting to \text{FALSE}) with \text{TRUE} meaning to return a list whose elements are \(V\) and its constituent parts, viz line1 to line6. This argument is used mainly for debugging.
- **test\.for\.symmetry**: Boolean with \text{TRUE} meaning to calculate each element of \(C\) explicitly, and default \text{FALSE} meaning to calculate only the elements of \(C\) that lie on or over the diagonal and use the fact that \(C\) is symmetric to calculate the other matrix elements. For \(n\) observations, this means \(n(n+1)/2\) evaluations, compared with \(n^2\) for the full case. The time saving is considerable, even for small matrices. Set this argument to \text{TRUE} only when debugging, or testing accuracy
- **phi**: Hyperparameters

Details

See KOH2001 for full details on page 3 of the supplement

Value

If \text{give\.answers} is the default value of \text{FALSE}, returns a matrix of covariances for use in p.page4().

If \text{give\.answers} is \text{TRUE}, returns a named list of (currently) 17 elements. Elements one to six are lines one to six respectively from page 3 of the supplement; subsequent lines give intermediate steps in the calculation. The final element is the matrix is the covariances as returned when \text{give\.answers} is \text{FALSE}. 
Note
This function takes a long time to run

Author(s)
Robin K. S. Hankin

References


See Also

fun, page4

Examples

data(toys)
(jj <- V.fun(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy,
   extractor=extractor.toy,
   Edash.theta=Edash.theta.toy,
   E.theta=E.theta.toy, phi=phi.toy))

## Now note that V.fun() changes with the PRIOR used for theta:
phi.different.theta <- phi.change(old.phi=phi.toy,
   theta.apriori.mean=c(100,100,100),phi.fun=phi.fun.toy)
V.fun(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy,
   extractor=extractor.toy,
   Edash.theta=Edash.theta.toy,
   E.theta=E.theta.toy, phi=phi.different.theta)
## different!

## Now compare jj above with V.fun() calculated with
## different phi2:

## different phi2:

phi.toy.new <- phi.change(phi.fun=phi.fun.toy, old.phi = phi.toy, psi2=c(8,8))
V.fun(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy,
   extractor=extractor.toy,
   Edash.theta=Edash.theta.toy,
   E.theta=E.theta.toy, phi=phi.toy.new)
## different!
## Not run:
```r
# 1. Create new toy datasets
jj <- create.new.toys.datasets(D1=D1.toy, D2=D2.toy)
y.toy <- jj$y.toy
z.toy <- jj$z.toy
d.toy <- jj$d.toy

# Compute distance matrix using V1 function
v.fun <- function(...)(V1(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy,
                          extractor=extractor.toy, Edash.theta=Edash.theta.toy,
                          E.theta=E.theta.toy, phi=phi.toy, give=TRUE))

Rprof(file="~/f.txt"); ignore <- v.fun(); Rprof(file=NULL)
```
## End(Not run)

---

### Distance matrix

**Description**

Gives the distance matrix between rows of D1 and D1 (or, if supplied, another matrix)

**Usage**

`V1(D1, other = NULL, phi)`

**Arguments**

- `D1` 
  Matrix of code run points
- `other` 
  Second matrix to compute distances to. If NULL, use the first supplied matrix
- `phi` 
  Hyperparameters

**Value**

Returns a matrix

**Author(s)**

Robin K. S. Hankin
References


See Also

V2

Examples

data(toys)
V2(D1=D1.toy, other=NULL, phi=phi.toy)
V2(D1=D1.toy[1,,drop=FALSE], other=NULL, phi=phi.toy)
V2(D1=D1.toy, other=D1.toy[1:3,, phi=phi.toy)
V2(D1=D1.toy,other=D1.fun(x.star=x.vec,t.vec=theta.toy),phi=phi.toy)

distance between observation points

Description

distance between observation points

Usage

V2(x, other = NULL, phi)

Arguments

- **x**: Matrix whose rows are observation points
- **other**: Second matrix; if NULL, use x
- **phi**: Hyperparameters

Details

This function returns the variance matrix of observations of the real process \( z \) at points \( D_2 = \{x_1, \ldots, x_n\} \).

It appears in the lower right corner of the variance matrix on the bottom of page 1 of the supplement, calculated by function \( V_d() \).

It is also used in functions \( \text{Cov.eqn9 supp}() \) and \( \text{V.fun}() \).
Author(s)
Robin K. S. Hankin

References

See Also
V1

Examples
```r
data(toys)
V2(D2.toy,other=NULL, phi=phi.toy)
V2(D2.toy,x.vec,phi=phi.toy)
```

Vd
Variance matrix for d

Description
Variance matrix for d, as per the bottom of page 1 of the supplement

Usage
```r
Vd(D1, D2, theta, phi)
```

Arguments
- D1 matrix of code run points
- D2 matrix of observation points
- theta Parameters
- phi hyperparameters

Author(s)
Robin K. S. Hankin
References


See Also

H.fun,V1,V2,C1

Examples

data(toys)
Vd(D1=D1.toy, D2=D2.toy, theta=theta.toy, phi=phi.toy)

\[
W
\]

\textit{covariance matrix for beta}

Description

Covariance matrix of beta given theta, phi, d

Usage

\(W(D1, D2, H1, H2, \text{theta, det=}FALSE, \phi)\)

Arguments

- \textbf{D1} Matrix whose rows are code run points
- \textbf{D2} Matrix whose rows are observation points
- \textbf{H1} regression function
- \textbf{H2} regression function
- \textbf{theta} parameters
- \textbf{det} Boolean, with default FALSE meaning to return the covariance matrix, and TRUE meaning to return its determinant.
- \textbf{phi} Hyperparameters
Details

This function is defined between equations 2 and 3 of the supplement. It is used in functions 
`betahat.f.fun.koh()`, `p.eqn supp()`, and `p.joint()`.

Returns

\[ W(\theta) = \left( H(\theta)^T V_d(\theta)^{-1} H(\theta) \right)^{-1} \]

If only the determinant is required, setting argument `det` to `TRUE` is faster than using `det(W(..., det=FALSE))`, as the former avoids an unnecessary use of `solve()`.

Author(s)

Robin K. S. Hankin

References


See Also

`betahat.f.fun.koh`

Examples

```r
data(toys)
W(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, theta=theta.toy, phi=phi.toy)
```

---

\[ W1 \]  

*Variance matrix for beta1hat*

---

Description

returns the variance-covariance matrix for the estimate of beta1hat

Usage

\[ W1(D1, H1, \text{det=FALSE, phi}) \]
Arguments

- `D1` matrix of code points
- `H1` Basis function generator
- `phi` Hyperparameters
- `det` Boolean, with default `FALSE` meaning to return the matrix, and `TRUE` meaning to return its determinant only

Details

If only the determinant is required, setting argument `det` to `TRUE` is faster than using `det(W1(..., det=FALSE))`, as the former avoids an unnecessary use of `solve()`.

Author(s)

Robin K. S. Hankin

References


See Also

`betaHat.fun`

Examples

```r
data(toys)
W1(D1=D1.toy, H1=H1.toy, phi=phi.toy)
```

Description

Variance matrix for beta2 as per page 4 of the supplement

Usage

```r
W2(D2, H2, V, det=FALSE)
```
Arguments

- **D2** matrix of observation points
- **H2** regression function
- **V** Overall covariance matrix
- **det** Boolean, with default FALSE meaning to return the matrix, and TRUE meaning to return its determinant only

Details

If only the determinant is required, setting argument det to TRUE is faster than using det(W2(..., det=FALSE)), as the former avoids an unnecessary use of solve().

Author(s)

Robin K. S. Hankin

References


See Also

- `V.fun`

Examples

data(toys)
W2(D2=D2.toy, H2=H2.toy, V=V.toy)
Index

*Topic array
  beta1hat.fun, 2
  beta2hat.fun, 4
  betahat.fun.koh, 5
  blockdiag, 7
  C1, 8
  cov.p5.supp, 9
  D1.fun, 12
  D2.fun, 13
  dists.2frames, 14
  E.theta.toy, 15
  EK.eqn10.supp, 17
  etahat, 19
  extractor.toy, 20
  Ez.eqn7.supp, 22
  Ez.eqn9.supp, 23
  H.fun, 24
  H1.toy, 25
  h1.toy, 27
  hbar.fun.toy, 28
  is.positive.definite, 29
  M4, 29
  p.eqn4.supp, 31
  p.eqn8.supp, 32
  p.page4, 34
  phi.fun.toy, 35
  prob.psi1, 38
  reality, 40
  stage1, 42
  symmetrize, 45
  tee, 46
  tt.fun, 49
  V.fun, 52
  V1, 54
  V2, 55
  Vd, 56
  W, 57
  W1, 58
  W2, 59

*Topic datasets
  create.new.toy.datasets, 11
  toys, 47
  beta1hat.fun, 2, 59
  beta2hat.fun, 3, 4
  betahat.fun.koh, 5, 58
  blockdiag, 7
  C1, 8, 57
  computer.model, 41
  computer.model(reality), 40
  Cov.eqn9.supp (cov.p5.supp), 9
  cov.p5.supp, 9
  create.new.toy.datasets, 11, 48
  d.toy(toys), 47
  D1.fun, 12, 21
  D1.toy, 13, 26
  D1.toy(toys), 47
  D2.fun, 13
  D2.toy(toys), 47
  dists.2frames, 14, 15
  E.theta.toy, 15, 48
  Edash.theta.toy(E.theta.toy), 15
  EK.eqn10.supp, 17
  etahat, 19
  extractor.toy, 20, 48
  Ez.eqn7.supp, 22
  Ez.eqn9.supp, 23, 46
  H.fun, 24, 57
  h.fun(tee), 46
  H1.toy, 25, 27, 38, 48
  h1.toy, 27, 28
  H2.toy(H1.toy), 25
  h2.toy(h1.toy), 27
  hbar.fun.toy, 28
  hh.fun(tt.fun), 49
  ht.fun(tt.fun), 49
is.positive.definite, 29

latin.hypercube, 11

MH, 29

model.inadequacy (reality), 40

p.eqn4.supp, 31, 40
p.eqn8.supp, 30, 32, 40
p.equationn4 supp (p.eqn4.supp), 31
p.page4, 20, 34, 53
phi.change (phi.fun.toy), 35
phi.fun.toy, 35, 44, 48
phi.toy (toys), 47
phi.true (reality), 40
prob.psi1, 38
prob.psi2 (prob.psi1), 38
prob.theta (prob.psi1), 38

reality, 11, 40

sample.theta (prob.psi1), 38
stage1, 33, 40, 42, 48
stage2 (stage1), 42
stage3 (stage1), 42
symmetrize, 45

t.fun, 9
t.fun (tt.fun), 49
t.vec.toy (toys), 47
tee, 24, 46
theta.toy, 13
theta.toy (toys), 47
toys, 11, 12, 17, 21, 38, 44, 47
tt.fun, 49, 53

V.fun, 22, 51, 52, 60
V.toy (toys), 47
V1, 54, 56, 57
V2, 55, 55, 57
Vd, 56

W, 57
W1, 32, 58
W2, 5, 33, 35, 59

X.dist.toy (toys), 47
x.toy (toys), 47
x.toy2 (toys), 47
x.vec (toys), 47