

# Package: calibrator (via r-universe)

August 22, 2024

**Type** Package

**Title** Bayesian Calibration of Complex Computer Codes

**Version** 1.2-9

**Depends** R (>= 3.5.0), emulator (>= 1.2-21), mvtnorm

**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-2

**URL** <https://github.com/RobinHankin/calibrator>

**BugReports** <https://github.com/RobinHankin/calibrator/issues>

**Repository** <https://robinhankin.r-universe.dev>

**RemoteUrl** <https://github.com/robinhankin/calibrator>

**RemoteRef** HEAD

**RemoteSha** 8637464ef2b57c603c2e0122b078560852d4999c

## Contents

calibrator-package . . . . .	2
beta1hat.fun . . . . .	3
beta2hat.fun . . . . .	5
betahat.fun.koh . . . . .	6
blockdiag . . . . .	8
C1 . . . . .	9
cov.p5.suppl . . . . .	10
create.new.toy.datasets . . . . .	11

D1.fun . . . . .	13
D2.fun . . . . .	14
dists.2frames . . . . .	15
E.theta.toy . . . . .	16
EK.eqn10.supp . . . . .	18
etahat . . . . .	20
extractor.toy . . . . .	21
Ez.eqn7.supp . . . . .	22
Ez.eqn9.supp . . . . .	24
H.fun . . . . .	25
H1.toy . . . . .	26
h1.toy . . . . .	27
hbar.fun.toy . . . . .	29
is.positive.definite . . . . .	30
MH . . . . .	30
p.eqn4.supp . . . . .	32
p.eqn8.supp . . . . .	33
p.page4 . . . . .	35
phi.fun.toy . . . . .	36
prob.psi1 . . . . .	39
reality . . . . .	41
stage1 . . . . .	43
symmetrize . . . . .	46
tee . . . . .	47
toys . . . . .	48
tt.fun . . . . .	50
V.fun . . . . .	52
V1 . . . . .	55
V2 . . . . .	56
Vd . . . . .	57
W . . . . .	58
W1 . . . . .	59
W2 . . . . .	60
<b>Index</b>	<b>62</b>

**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/.

**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.rob@gmail.com](mailto:hankin.rob@gmail.com)>

**References**

- M. C. Kennedy and A. O'Hagan 2001. "Bayesian calibration of computer models". Journal of the Royal Statistical Society, Series B, 63(3): 425–464
- R. K. S. Hankin 2005. "Introducing BACCO, an R bundle for Bayesian analysis of computer code output", Journal of Statistical Software, 14(16)

---

beta1hat.fun

*beta1 estimator*

---

**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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[beta2hat.fun](#)

## Examples

```
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 psi1:
phi.fix <- phi.change(old.phi=phi.toy,psi1=c(1, 0.5, 1.0, 1.0, 0.5, 0.4),phi.fun=phi.fun.toy)
# The value for psi1 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	<i>estimator for beta2</i>
--------------	----------------------------

---

### 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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[W2](#)

**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)

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.toy.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

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)

```

---

betahat.fun.koh

*Expectation of beta, given theta, phi and d*


---

**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**

```

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

```

**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 `Ez.eqn9.supp()` and `p.eqn8.supp()`.

The user should always use `betahat.fun.koh()`, which is a wrapper for `betahat.fun.koh.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 `betahat.fun.koh.vector()` is applied to the rows.

In `betahat.fun.koh()`, the rownames are assigned by a kludgy call to `H.fun()`, which itself uses a kludge to determine colnames.

The function returns

$$\hat{\beta}(\theta) = \mathbf{W}(\theta)^T \mathbf{H}(\theta)^T \mathbf{V}_d(\theta)^{-1} \mathbf{d}.$$

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464

M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>

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(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)
## should be identical

jj.theta <- rbind(theta.toy, theta.toy+1, theta.toy+2, theta.toy*0)
betahat.fun.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()):

betahat.fun.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 addiag 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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[t.fun](#)

**Examples**

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

cov.p5.supp

*Covariance function for posterior distribution of z***Description**

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

**Usage**

```
Cov.eqn9.supp(x, xdash=NULL, theta, d, D1, D2, H1, H2, phi)
cov.p5.supp(x, 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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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(toys)
x <- rbind(x.toy,x.toy+1,x.toy,x.toy,x.toy)
rownames(x) <- letters[1:5]
xdash <- rbind(x*2,x.toy)
rownames(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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[toys](#), [reality](#), [latin.hypercube](#)

**Examples**

```
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

```
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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[toys](#)

### Examples

```
data(toys)
jj <- extractor.toy(D1.toy)
x.star.toy <- jj$x.star
t.vec.toy <- jj$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**

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

**Arguments**

D2	Observation points
theta	Parameters

**Author(s)**

Robin K. S. Hankin

**References**

M. C. Kennedy and A. O'Hagan 2001. "Bayesian calibration of computer models". Journal of the Royal Statistical Society B, 63(3) pp425-464

M. C. Kennedy and A. O'Hagan 2001. "Supplementary details on Bayesian calibration of computer models", Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>

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**

[D1.toy](#), [theta.toy](#)

**Examples**

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

---

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

```
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). If a value for A.lower is specified, this means that a relatively opaque but possibly faster method will be used. The time saving ought to be negligible unless nrow(a) (or nrow(b) if supplied), is huge. <b>Note that this option does not test for symmetry of matrix A</b>
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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>
- 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

[dists.2frames](#)

**Examples**

```

data(toys)

dists.2frames(a=D2.toy,A=diag(2))

A <- diag(2) + matrix(0.2,2,2)
A.lower <- t(chol(A))
jj.1 <- dists.2frames(a=D2.toy, A=A, test=TRUE)
jj.2 <- dists.2frames(a=D2.toy, A=A, test=FALSE)

jj.3 <- dists.2frames(a=D2.toy, A.lower=A.lower, test=FALSE)
jj.4 <- dists.2frames(a=D2.toy, A.lower=A.lower, test=TRUE)

```

---

E.theta.toy

*Expectation and variance with respect to theta*


---

**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**

```

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 <code>Edash.theta.toy()</code> when <code>fast.but.opaque</code> is TRUE
x	lat/long point (for <code>Edash.theta.toy</code> )
t.vec	Matrix whose rows are parameter values (for <code>Edash.theta.toy</code> )
k	Integer specifying column (for <code>Edash.theta.toy</code> )
give.mean	In <code>E.theta.toy()</code> , Boolean, with default TRUE meaning to return the mean (expectation), and FALSE meaning to return the “variance”
fast.but.opaque	In <code>Edash.theta.toy()</code> , Boolean, with default FALSE meaning to use a slow but clear method. If TRUE, use faster code but parameters <code>a</code> and <code>b</code> must then be specified

- a Constant term, needed if `fast.but.opaque` is TRUE:  $(V_\theta^{-1} + 2\Omega_t)^{-1} V_\theta^{-1} m_\theta$ . Specifying `a` in advance saves execution time
- b Linear term, needed if `fast.but.opaque` is TRUE:  $2(V_\theta^{-1} + 2\Omega_t)^{-1} \Omega_t$  (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  $(1, x\bar{\theta})^T$  but

$$E_\theta(h_1(x, \theta)h_1(x, \theta)^T)$$

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

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

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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[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
H1	Regression function for D1
H2	Regression function for D2
d	Vector of code outputs and field observations

<code>include.prior</code>	Boolean; passed to function <code>p.eqn8.supp()</code> ( <code>qv</code> )
<code>hbar.fun</code>	Function that gives expectation (with respect to $X$ ) of $h_1(x, \theta)$ and $h_2(x)$ as per section 4.2
<code>lower.theta</code>	Lower integration limit for $\theta$ (NB: a vector)
<code>upper.theta</code>	Upper integration limit for $\theta$ (NB: a vector)
<code>extractor</code>	Extractor function; see <code>extractor.toy()</code> for an example
<code>give.info</code>	Boolean, with default <code>FALSE</code> meaning to return just the answer and <code>TRUE</code> to return the answer along with all output from both integrations as performed by <code>adaptIntegrate()</code>
<code>phi</code>	Hyperparameters
<code>...</code>	Extra arguments passed to the integration function. If multidimensional (ie <code>length(theta)&gt;1</code> ), then the arguments are passed to <code>adaptIntegrate()</code> ; if one dimensional, they are passed to <code>integrate()</code>

### 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 package formerly used `adapt` package, but this is no longer available on CRAN. The package now uses the `cubature` package.

### Author(s)

Robin K. S. Hankin

### References

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- R. K. S. Hankin 2005. *Introducing BACCO, an R bundle for Bayesian analysis of computer code output*, Journal of Statistical Software, 14(16)

**Examples**

```

1+1
## Not run:
# Not run because it takes R CMD check too long

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

**Usage**

```
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

M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464

M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>

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

[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

*Extracts 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.star` and columns 3 through 5 give the elements of `t.vec`.

Function `extractor.toy` is the inverse of function `D1.fun`, in the sense that `extractor.toy` splits up `D1` into `x.star` and `t.vec`, while `D1.fun` joins them up again

**Value**

Returns a list with two elements:

<code>x.star</code>	A matrix containing the lat/longs of the code run points
<code>t.vec</code>	A matrix containing the parameters used for the code runs

**Author(s)**

Robin K. S. Hankin

**References**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[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)
```

---

Ez.eqn7.supp

*Expectation of z given y, beta2, phi*

---

**Description**

Expectation as per equation 7 on the supplement

**Usage**

```
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 D1
beta2	coefficients
y	Code outputs at points corresponding to rows of D1
E.theta	Expectation function to use
phi	hyperparameters

**Author(s)**

Robin K. S. Hankin

**References**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[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**

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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(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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[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.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))
```

---

h1.toy

*Basis functions*

---

**Description**

Basis functions for D1 and D2 respectively.

**Usage**

```
h1.toy(x)
h2.toy(x)
```

## Arguments

x                      Vector of lat/long or lat/long and theta

## Details

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

**NB** If the definition of `h1.toy()` or `h2.toy()` is changed, then function `hbar.toy()` must be changed to match. This cannot be done automatically, as the form of `hbar.toy()` depends on the distribution of  $X$ . The shibboleth is whether  $E_X()$  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. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[H1.toy](#)

## Examples

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

---

hbar.fun.toy      *Toy example of hbar (section 4.2)*

---

**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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>
- 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**

[h1.toy](#)

**Examples**

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

---

`is.positive.definite` *Is a matrix positive definite?*

---

### Description

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

### Usage

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

### Arguments

<code>a</code>	Matrix to be tested
<code>...</code>	Extra arguments passed to <code>eigen()</code> , such as <code>symmetric</code> .

### 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

```
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

```
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  $q(X|Y)$  is  $q(\cdot|X) = N(X, \sigma^2)$

**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**

- W. R. Gilks et al 1996. *Markov Chain Monte Carlo in practice*. Chapman and Hall, 1996. ISBN 0-412-05551-1
- 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**

```
# 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){
  exp(-quad.form(A/2,x)) +
  exp(-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")

```

---

p.eqn4.supp

*Apostiori probability of psi1*


---

## Description

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

## Usage

```
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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[W1](#)

## Examples

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

---

p.eqn8.supp

*A posteriori probability of hyperparameters*

---

## Description

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

## Usage

```
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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>
- 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

[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)
```

**Description**

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

**Usage**

```
p.page4(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 $E'$
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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

W2

## Examples

```
data(toys)

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)

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

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

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

```
phi.fun.toy(rho, lambda, psi1, psi1.apriori, psi2, psi2.apriori,
theta.apriori)
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 $\phi$ 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 sigma1squared. Recall that $\Omega_x$ and $\Omega_t$ are arbitrary functions of $\psi_1$ . In this case, the values are omega_x=psi1[1:2], omega_t=psi1[3:4] and sigma1squared=psi1[6]
psi1.apriori	A priori PDF for $\psi_1$ . 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 $\psi_1$ and the sixth is for $\sigma_1^2$ ), 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 $\psi_1$
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 $\psi_2$ <b>and</b> hyperparameters $\rho$ and $\lambda$ (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 $\rho$ and $\lambda$ respectively; the third and fourth elements are the apriori mean of $\psi_2$ (that is, x and y respectively); and the fifth is the mean of $\sigma_2^2$ . 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 $\psi_2$
psi2.apriori.mean	In phi.change.toy(), use to change just the mean of psi2

<code>psi2.apriori.sigma</code>	In <code>phi.change.toy()</code> , use to change just the variance matrix of <code>psi2</code>
<code>theta.apriori</code>	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 <code>prob.theta()</code> to calculate the apriori probability density of a particular value of $\theta$
<code>theta.apriori.mean</code>	In <code>phi.change.toy()</code> , use to change just the mean of <code>theta</code>
<code>theta.apriori.sigma</code>	In <code>phi.change.toy()</code> , use to change just the variance matrix of <code>theta</code>

### Details

Note that this toy function contains within itself `pdm.maker.toy()` which extracts `omega_x` and `omega_t` and `sigma1squared` from `psi1`. 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:

<code>rho</code>	Correlation hyperparameter
<code>lambda</code>	Noise hyperparameter
<code>psi1</code>	Roughness lengths hyperparameter for D1
<code>psi1.apriori</code>	Apriori mean and variance matrix for <code>psi1</code>
<code>psi2</code>	Roughness lengths hyperparameter for D2
<code>psi2.apriori</code>	Apriori mean and variance matrix for <code>psi2</code>
<code>theta.apriori</code>	Apriori mean and variance matrix for the parameters
<code>omega_x</code>	Positive definite matrix for the lat/long part of D1, whose diagonal is <code>psi1[1:2]</code>
<code>omega_t</code>	Positive definite matrix for the code parameters <code>theta</code> , whose diagonal is <code>psi1[3:5]</code>
<code>omegastar_x</code>	Positive definite matrix for use in equation 13 of the supplement; represents distances between rows of D2
<code>sigma1squared</code>	variance
<code>sigma2squared</code>	variance
<code>omega_x.upper</code>	Upper triangular Cholesky decomposition for <code>omega_x</code>
<code>omega_x.lower</code>	Lower triangular Cholesky decomposition for <code>omega_x</code>
<code>omega_t.upper</code>	Upper triangular Cholesky decomposition for <code>omega_t</code>
<code>omega_t.lower</code>	Lower triangular Cholesky decomposition for <code>omega_t</code>
<code>a</code>	Precalculated matrix for use in <code>Edash.theta(..., fast.but.opaque=TRUE)</code>
<code>b</code>	Precalculated matrix for use in <code>Edash.theta(..., fast.but.opaque=TRUE)</code>
<code>c</code>	Precalculated scalar for use in <code>ht.fun(..., fast.but.opaque=TRUE)</code>
<code>A</code>	Precalculated scalar for use in <code>tt.fun()</code>
<code>A.upper</code>	Upper triangular Cholesky decomposition for <code>A</code>
<code>A.lower</code>	Lower triangular Cholesky decomposition for <code>A</code>

**Author(s)**

Robin K. S. Hankin

**References**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**[toys](#), [H1.toy](#)**Examples**

```
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.apriori = list(
    mean=rep(0,6), sigma=0.4+diag(6)),
  psi2=structure(c(2.1, 2.2), .Names = c("x","y")),
  psi2.apriori = list(mean=rep(0,5),sigma=0.2+diag(5)),
  theta.apriori = 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**

```

prob.psi1(phi, lognormally.distributed=TRUE)
prob.psi2(phi, lognormally.distributed=TRUE)
prob.theta(theta, phi, lognormally.distributed=FALSE)
sample.theta(n=1, phi)

```

**Arguments**

phi	Hyperparameters
theta	Parameters
lognormally.distributed	Boolean variable with FALSE meaning to assume a Gaussian distribution and TRUE meaning to use a lognormal distribution.
n	In function <code>sample.theta()</code> , the number of observations to take

**Details**

These functions use package `mvtnorm` to calculate the probability density under the assumption that the PDF is lognormal. One implication would be that `phi$psi2.apriori$mean` and `phi$psi1.apriori$mean` are the means of the **logarithms** of the elements of `psi1` and `psi2` (which are thus assumed to be positive). The 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 `lognormally.distributed`. Take `prob.theta()` as an example. If `lognormally.distributed` is TRUE, then `log(theta)` is normally distributed with mean `phi$theta.apriori$mean` and variance `phi$theta.apriori$sigma`. If FALSE, `theta` is normally distributed with mean `phi$theta.apriori$mean` and variance `phi$theta.apriori$sigma`.

Interpretation of `phi$theta.apriori$mean` depends on the value of `lognormally.distributed`: if TRUE it is the expected value of `log(theta)`; if FALSE, it is the expectation of `theta`.

The reason that `prob.theta` has a different default value for `lognormally.distributed` is that some elements of `theta` might be negative, contraindicating a lognormal distribution

**Author(s)**

Robin K. S. Hankin

**References**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[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)

```

---

reality

*Reality*


---

**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 <i>true</i> 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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[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)$t.vec)

phi.fix <- phi.change(old.phi=phi.toy,
  psi1=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 psi1 (see the
# section on psi1 in ?phi.fun.toy for how to get this):

phi.fix <- phi.change(old.phi=phi.toy,psi1=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.stdin = 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

<code>maxit</code>	Maximum number of iterations as passed to <code>optim()</code>
<code>trace</code>	Amount of information displayed, as passed to <code>optim()</code>

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 <code>y.toy</code>
z	Observations. Toy example is <code>z.toy</code>
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 <code>E.theta.toy()</code> and <code>Edash.theta.toy()</code>
phi.fun	Function to create hyperparameters; passed (in <code>stage1()</code> and <code>stage2()</code> ) to <code>phi.change()</code> . Toy version is <code>phi.fun.toy()</code>
method	Method argument passed to <code>optim()</code> ; <code>qv</code>
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 <code>p.eqn4.supp()</code> for <code>stage1()</code> , <code>p.page4()</code> for <code>stage2()</code> , and <code>p.eqn8.supp()</code> for <code>stage3()</code>
lognormally.distributed	Boolean with TRUE meaning to use a lognormal distn. See <code>prob.theta</code> for details
do.filewrite	Boolean, with TRUE meaning to save a loadable file <code>stage[123].&lt;date&gt;</code> , 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 <code>do.filewrite</code> is TRUE
isotropic	In function <code>stage2()</code> , 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 <code>stage2()</code> , a Boolean argument, with default FALSE meaning to use the real value for matrix <code>V.temp</code> , and TRUE meaning to use a standing that is the same size but contains fictitious values. The only time to set <code>use.standin</code> 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 <code>stage3()</code> , the starting point of the optimization with default NULL meaning to use the maximum likelihood point of the apriori distribution (ie <code>phi\$theta.apriori\$mean</code> )
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 to TRUE will force phi\$omegastar\_x to be a function of a length one scalar. The value of phi\$omegastar\_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\$psi2.

Function stage2() estimates  $\psi_2$  **and**  $\rho$  **and**  $\lambda$ , using optimization. Note that  $\psi_2$  includes  $\sigma_2^2$  in addition to omegastar\_x (in the toy case,  $\psi_2$  has three elements: the first two are the diagonal of omegastar\_x and the third is  $\sigma_2^2$  **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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[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

phi.toy.stage1 <- stage1(D1=D1.toy, y=y.toy, H1=H1.toy, maxit=10, phi.fun=phi.fun.toy, phi=phi.toy)

phi.toy.stage2 <- stage2(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy,
y=y.toy, z=z.toy, extractor=extractor.toy,
```

```
phi.fun=phi.fun.toy, E.theta=E.theta.toy, Edash.theta=Edash.theta.toy,
maxit=3, phi=phi.toy.stage1)

stage3(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, d=d.toy, maxit=3, phi=phi.toy.stage2)

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

stage3(D1=D1.toy, D2=D2.toy, H1=H1.toy, H2=H2.toy, d=d.toy, maxit=3, phi=phi.true)
```

---

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))
```

---

tee

*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

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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[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)
```

**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  $\theta$ . 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  $\theta$ . 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; they are created by running `inst/toys.R`. 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

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>
- 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

[create.new.toy.datasets](#), [E.theta.toy](#), [extractor.toy](#), [H1.toy](#), [phi.fun.toy](#), [stage1](#)

### Examples

```
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

```
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 $\theta$ ; <code>Edash.theta.toy</code> is the example for the toy dataset
E.theta	Function to return expectation of H with respect to $\theta$
test.for.symmetry	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

## Details

The four functions return integrals representing means taken over the  $\theta$ . 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

$$\int t(x_i, \theta)^T p(\theta) d\theta = \int c_1((x_i, \theta), (x_j^*, t_j)) p(\theta) d\theta$$

using the formula

$$\sigma_1^2 |I + 2V_\theta \Omega_x|^{-1/2} \exp \left\{ - (x_i - x_j^*)^T \Omega_x (x_i - x_j^*) \right\} \times \exp \left\{ - (m_\theta - t_j)^T (2V_\theta + \Omega_t^{-1})^{-1} (m_\theta - t_j) \right\}.$$

It is used in Ez\_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().

## Value

Each function returns a matrix as described in KOH2001

## Author(s)

Robin K. S. Hankin

## References

- M. C. Kennedy and A. O’Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O’Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[V.fun](#)

## Examples

```
data(toys)

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

```
V.fun(D1, D2, H1, H2, extractor,
      E.theta, Edash.theta, give.answers=FALSE, test.for.symmetry=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 <code>x.star</code> and <code>t.vec</code> from D1
Edash.theta	Function to return expectation of H with respect to the alternative distribution of $\theta$ ; <code>Edash.theta.toy</code> 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 <code>h1()</code> changes.
give.answers	Boolean (defaulting to FALSE) with TRUE meaning to return a list whose elements are V and its constituent parts, viz <code>line1</code> to <code>line6</code> . This argument is used mainly for debugging.
test.for.symmetry	Boolean with TRUE meaning to calculate each element of $C$ explicitly, and default 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 TRUE only when debugging, or testing accuracy
phi	Hyperparameters

**Details**

See KOH2001 for full details on page 3 of the supplement

**Value**

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

If `give.answers` is 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 `give.answers` is FALSE.

**Note**

This function takes a long time to run

**Author(s)**

Robin K. S. Hankin

## References

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[tt.fun,p.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:

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

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:
data(toys)
set.seed(0)
jj <- create.new.toy.datasets(D1=D1.toy , D2=D2.toy)
y.toy <- jj$y.toy
```

```

z.toy <- jj$z.toy
d.toy <- jj$d.toy

v.fun <- function(...){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, give=TRUE)}

Rprof(file=~ /f.txt");ignore <- v.fun();Rprof(file=NULL)
system("cd ; R CMD Rprof ~/f.txt > ~/ff.txt")

## End(Not run)

```

---

V1

*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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**[V2](#)**Examples**

```

data(toys)
V1(D1=D1.toy, other=NULL, phi=phi.toy)
V1(D1=D1.toy[1,,drop=FALSE], other=NULL, phi=phi.toy)

V1(D1=D1.toy, other=D1.toy[1:3,], phi=phi.toy)

V1(D1=D1.toy,other=D1.fun(x.star=x.vec,t.vec=theta.toy),phi=phi.toy)

```

---

**V2***distance between observation points*

---

**Description**

distance between observation points

**Usage**`V2(x, other = NULL, phi)`**Arguments**

<code>x</code>	Matrix whose rows are observation points
<code>other</code>	Second matrix; if NULL, use <code>x</code>
<code>phi</code>	Hyperparameters

**Details**

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

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

It is also used in functions `Cov.eqn9.supp()` and `V.fun()`.

**Author(s)**

Robin K. S. Hankin

## References

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

V1

## Examples

```
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

```
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

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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

[H.fun,V1,V2,C1](#)

## Examples

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

---

W

*covariance matrix for beta*

---

## Description

Covariance matrix of beta given theta, phi, d

## Usage

```
W(D1, D2, H1, H2, theta, det=FALSE, phi)
```

## Arguments

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

**Details**

This function is defined between equations 2 and 3 of the supplement. It is used in functions `betahat.fun.koh()`, `p.eqn8.supp()`, and `p.joint()`.

Returns

$$\mathbf{W}(\theta) = (\mathbf{H}(\theta)^T \mathbf{V}_d(\theta)^{-1} \mathbf{H}(\theta))^{-1}$$

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**

M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464

M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>

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**

[betahat.fun.koh](#)

**Examples**

```
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 betaIhat*

---

**Description**

returns the variance-covariance matrix for the estimate of `betaIhat`

**Usage**

```
W1(D1, H1, 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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/cal-sup.ps>
- 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**

[beta1hat.fun](#)

**Examples**

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

---

W2

*variance matrix for beta2*

---

**Description**

Variance matrix for beta2 as per page 4 of the supplement

**Usage**

```
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**

- M. C. Kennedy and A. O'Hagan 2001. *Bayesian calibration of computer models*. Journal of the Royal Statistical Society B, 63(3) pp425-464
- M. C. Kennedy and A. O'Hagan 2001. *Supplementary details on Bayesian calibration of computer models*, Internal report, University of Sheffield. Available at <http://www.tonyohagan.co.uk/academic/ps/calsup.ps>
- 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**

[V.fun](#)

**Examples**

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

# Index

## \* array

- beta1hat.fun, 3
- beta2hat.fun, 5
- betahat.fun.koh, 6
- blockdiag, 8
- C1, 9
- cov.p5.supp, 10
- D1.fun, 13
- D2.fun, 14
- dists.2frames, 15
- E.theta.toy, 16
- EK.eqn10.supp, 18
- etahat, 20
- extractor.toy, 21
- Ez.eqn7.supp, 22
- Ez.eqn9.supp, 24
- H.fun, 25
- H1.toy, 26
- h1.toy, 27
- hbar.fun.toy, 29
- is.positive.definite, 30
- MH, 30
- p.eqn4.supp, 32
- p.eqn8.supp, 33
- p.page4, 35
- phi.fun.toy, 36
- prob.psi1, 39
- reality, 41
- stage1, 43
- symmetrize, 46
- tee, 47
- tt.fun, 50
- V.fun, 52
- V1, 55
- V2, 56
- Vd, 57
- w, 58
- w1, 59
- w2, 60

## \* datasets

- create.new.toy.datasets, 11
- toys, 48

## \* package

- calibrator-package, 2

- beta1hat.fun, 3, 60
- beta2hat.fun, 4, 5
- betahat.fun.koh, 6, 59
- blockdiag, 8
- C1, 9, 58
- calibrator (calibrator-package), 2
- calibrator-package, 2
- computer.model, 42
- computer.model (reality), 41
- Cov.eqn9.supp (cov.p5.supp), 10
- cov.p5.supp, 10
- create.new.toy.datasets, 11, 49
- d.toy (toys), 48
- D1.fun, 13, 22
- D1.toy, 14, 27
- D1.toy (toys), 48
- D2.fun, 14
- D2.toy (toys), 48
- dists.2frames, 15, 15
- E.theta.toy, 16, 49
- Edash.theta.toy (E.theta.toy), 16
- EK.eqn10.supp, 18
- etahat, 20
- extractor.toy, 21, 49
- Ez.eqn7.supp, 22
- Ez.eqn9.supp, 24, 47
- H.fun, 25, 58
- h.fun (tee), 47
- H1.toy, 26, 28, 39, 49
- h1.toy, 27, 29
- H2.toy (H1.toy), 26

h2.toy (h1.toy), 27  
hbar.fun.toy, 29  
hh.fun(tt.fun), 50  
ht.fun(tt.fun), 50

is.positive.definite, 30

latin.hypercube, 12

MH, 30  
model.inadequacy (reality), 41

p.eqn4.supp, 32, 40  
p.eqn8.supp, 31, 33, 40  
p.equationn4.supp (p.eqn4.supp), 32  
p.page4, 21, 35, 54  
phi.change (phi.fun.toy), 36  
phi.fun.toy, 36, 45, 49  
phi.toy (toys), 48  
phi.true (reality), 41  
prob.psi1, 39  
prob.psi2 (prob.psi1), 39  
prob.theta (prob.psi1), 39

reality, 12, 41

sample.theta (prob.psi1), 39  
stage1, 34, 40, 43, 49  
stage2 (stage1), 43  
stage3 (stage1), 43  
symmetrize, 46

t.fun, 9  
t.fun(tt.fun), 50  
t.vec.toy (toys), 48  
tee, 25, 47  
theta.toy, 14  
theta.toy (toys), 48  
toys, 12, 13, 18, 22, 39, 45, 48  
tt.fun, 50, 54

V.fun, 23, 52, 52, 61  
V.toy (toys), 48  
V1, 55, 57, 58  
V2, 56, 56, 58  
Vd, 57

W, 58  
W1, 33, 59  
W2, 5, 34, 36, 60

X.dist.toy (toys), 48  
x.toy (toys), 48  
x.toy2 (toys), 48  
x.vec (toys), 48

y.toy (toys), 48

z.toy (toys), 48