# Package ‘approximator’

August 29, 2018

**Type** Package

**Title** Bayesian Prediction of Complex Computer Codes

**Version** 1.2-7

**Author** Robin K. S. Hankin

**Depends** R (>= 2.0.0), emulator (>= 1.2-11)

**Imports** mvtnorm

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

**Description** Performs Bayesian prediction of complex computer codes when fast approximations are available. It uses a hierarchical version of the Gaussian process, originally proposed by Kennedy and O'Hagan (2000), Biometrika 87(1):1.



**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2018-08-29 04:24:34 UTC

## R topics documented:

---

`approximator-package` *Bayesian approximation of computer models when fast approximations are available*

---

## Description

Implements the ideas of Kennedy and O'Hagan 2000 (see references).

## Details

| | |
|---|---|
| Package: | approximator |
| Type: | Package |
| Version: | 1.0 |
| Date: | 2006-01-10 |
| License: | GPL |

This package implements the Bayesian approximation techniques discussed in Kennedy and O'Hagan 2000.

In its simplest form, it takes input from a "slow" but accurate code and a "fast" but inaccurate code, each run at different points in parameter space. The approximator package then uses both sets of model runs to infer what the slow code would produce at a given, untried point in parameter space.

The package includes functionality to work with a hierarchy of codes with increasing accuracy.

## Author(s)

Robin K. S. Hankin

Maintainer: <hankin.robin@gmail.com>

## References

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

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
mdash.fun(x=1:3, D1=D1.toy, subsets=subsets.toy, hpa=hpa.toy, z=z.toy, basis=basis.toy)
```

---

Afun *Matrix of correlations between two sets of points*

---

### Description

Returns the matrix of correlations of code output at each level evaluated at points on the design matrix.

### Usage

```
Afun(level, Di, Dj, hpa)
```

### Arguments

| | |
|---|---|
| `level` | The level. This enters via the correlation scales |
| `Di` | First set of points |
| `Dj` | Second set of points |
| `hpa` | Hyperparameter object |

### Details

This is essentially a convenient wrapper for function `corr.matrix`. It is not really intended for the end user.

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### See Also

`corr`,`c_fun`

## Examples

```
data(toyapps)
D2 <-  D1.toy[subsets.toy[[2]],]
D3 <-  D1.toy[subsets.toy[[3]],]

Afun(1,D2,D3,hpa.toy)
Afun(2,D2,D3,hpa.toy)
```

---

`as.sublist` *Converts a level one design matrix and a subsets object into a list of design matrices, one for each level*

---

## Description

Given a level one design matrix, and a subsets object, convert into a list of design matrices, each one of which is the design matrix for its own level

## Usage

```
as.sublist(D1, subsets)
```

## Arguments

| | |
|---|---|
| `D1` | Design matrix for level one code |
| `subsets` | subsets object |

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(toyapps)
as.sublist(D1=D1.toy , subsets=subsets.toy)
```

---

`basis.toy` *Toy basis functions*

---

## Description

A working example of a basis function

## Usage

```
basis.toy(x)
```

## Arguments

| | |
|---|---|
| `x` | Point in parameter space |

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(toyapps)
basis.toy(D1.toy)
```

---

`betahat.app` *Estimate for beta*

---

## Description

Returns the estimator for beta; equation 5. Function `betahat.app()` returns the estimate in terms of fundamental variables; `betahat.app.H()` requires the `H` matrix.

## Usage

```
betahat.app.H(H, V = NULL, Vinv = NULL, z)
betahat.app(D1, subsets, basis, hpa, z, use.Vinv=TRUE)
```

## Arguments

| | |
|---|---|
| `H` | In `betahat.app.H()`, the H matrix, eg that returned by `H.fun()` |
| `V` | Variance matrix |
| `Vinv` | Inverse of variance matrix. If not supplied, it is calculated |
| `use.Vinv` | In function `betahat.app()`, a Boolean argument with default `TRUE` meaning to calculate the inverse of the `V` matrix; and `FALSE` meaning to use a method which does not involve calculating the inverse of `V`. The default method seems to be faster; YMMV |
| `z` | vector of observations |
| `D1` | Design matrix for level 1 code |
| `subsets` | Subsets object |
| `basis` | Basis function |
| `hpa` | Hyperparameter object |

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(toyapps)

 betahat.app(D1=D1.toy, subsets=subsets.toy, basis=basis.toy, hpa=hpa.toy, z=z.toy, use.Vinv=TRUE)

 H <- H.fun.app(D1=D1.toy, subsets=subsets.toy, basis=basis.toy,hpa=hpa.toy)
 V <- V.fun.app(D1=D1.toy, subsets=subsets.toy, hpa=hpa.toy)
 betahat.app.H(H=H,V=V,z=z.toy)
```

---

`c.fun` *Correlations between points in parameter space*

---

## Description

Correlation matrices between (sets of) points in parameter space, both prior (`c_fun()`) and posterior (`cdash.fun()`).

## Usage

```
c_fun(x, xdash=x, subsets, hpa)
cdash.fun(x, xdash=x, V=NULL, Vinv=NULL, D1, subsets, basis, hpa, method=2)
```

## Arguments

| | |
|---|---|
| `x,xdash` | Points in parameter space; or, if a matrix, interpret the rows as points in parameter space. Note that the default value of `xdash` (viz `x`) will return the variance-covariance matrix of a set of points |
| `D1` | Design matrix |
| `subsets` | Subset object |
| `hpa` | hyperparameter object |
| `basis` | Basis function |
| `V,Vinv` | In function `cdash.fun()`, the data covariance matrix and its inverse. If `NULL`, the matrix will be calculated from scratch. Supplying a precalculated value for `V`, and especially `Vinv`, makes for very much faster execution (edepending on `method`) |
| `method` | Integer specifying which of several algebraically identical methods to use. See the source code for details, but default option 2 seems to be the best. Bear in mind that option 3 does not require inversion of a matrix, but is not faster in practice |

## Value

Returns a matrix of covariances

## Note

Do not confuse function `c_fun()`, which computes $c(x, x')$ defined just below equation 7 on page 4 with $c_t(x, x')$ defined in equation 3 on page 3.

Consider the example given for two levels on page 4 just after equation 7: $c(x, x') = c_2(x, x') + \rho_1^2 c_1(x, x')$ is a kind of prior covariance matrix. Matrix $c'(x, x')$ is a posterior covariance matrix, conditional on the code observations.

Function `Afun()` evaluates $c_t(x, x')$ in a nice vectorized way.

Equation 7 of KOH2000 contains a typo.

## Author(s)

Robin K. S. Hankin

## References

KOH2000

## See Also

`Afun`

## Examples

```
data(toyapps)

x <- latin.hypercube(4,3)
rownames(x) <- c("ash" , "elm" , "oak", "pine")
xdash <- latin.hypercube(7,3)
rownames(xdash) <- c("cod","bream","skate","sole","eel","crab","squid")

cdash.fun(x=x,xdash=xdash, D1=D1.toy, basis=basis.toy,subsets=subsets.toy, hpa=hpa.toy)

# Now add a point whose top-level value is known:
x <- rbind(x,D1.toy[subsets.toy[[4]][1],])

cdash.fun(x=x,xdash=xdash, D1=D1.toy, basis=basis.toy,subsets=subsets.toy, hpa=hpa.toy)
# Observe how the bottom row is zero (up to rounding error)
```

---

`generate.toy.observations` *Er, generate toy observations*

---

## Description

Generates toy observations on four levels using either internal (unknown) parameters and hyperparameters, or user-supplied versions.

## Usage

```
generate.toy.observations(D1, subsets, basis.fun, hpa = NULL, betas = NULL,
export.truth = FALSE)
```

## Arguments

| | |
|---|---|
| `D1` | Design matrix for level 1 code |
| `subsets` | Subset object |
| `basis.fun` | Basis function |
| `hpa` | Hyperparameter object. If `NULL`, use the internal (true but unknown) hyperparameter object |
| `betas` | Regression coefficients. If `NULL`, use the internal (true but unknown) regression coefficients |
| `export.truth` | Boolean, with default `FALSE` meaning to return synthetic observations and `TRUE` meaning to return the actual hyperparameters and coefficients. |

## Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
generate.toy.observations(D1=D1.toy, subsets=subsets.toy, basis.fun=basis.toy)
```

---

## genie — *Genie datasets for approximator package*

---

### Description

Genie datasets that illustrate the package.

### Usage

```
data(genie)
D1.genie
hpa.genie
z.genie
subsets.genie
basis.genie(x)
hpa.fun.genie(x)
hpa.genie.start
hpa.genie.optimal
```

### Arguments

| | |
|---|---|
| x | A 4-element vector (for `basis.genie()`); a 19-element vector (for `hpa.fun.genie()`) |

### Format

The genie example is a case with three levels.

The `D1.genie` matrix is 36 rows of code run points, corresponding to the observations of the level 1 code. It has four columns, one per parameter.

`hpa.genie` is a hyperparameter object.

`subsets.genie` is a list of three elements. Element $i$ corresponds to the rows of `D1.genie` at which level $i$ has been observed.

`z.genie` is a three element list. Each element is a vector; element $i$ corresponds to observations of level $i$. The lengths will match those of `subsets.genie`.

Function `basis.genie()` is a suitable basis function.

Function `hpa.fun.genie()` creates a hyperparameter object in a form suitable for passing to the other functions in the library.

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(genie)
z.genie


jj <- list(trace=100,maxit=10)

hpa.genie.level1 <- opt.1(D=D1.genie, z=z.genie,
                          basis=basis.genie, subsets=subsets.genie,
                          hpa.start=hpa.genie.start,control=jj)

hpa.genie.level2 <- opt.gt.1(level=2, D=D1.genie, z=z.genie,
                          basis=basis.genie, subsets=subsets.genie,
                          hpa.start=hpa.genie.level1,control=jj)

hpa.genie.level3 <- opt.gt.1(level=3, D=D1.genie, z=z.genie,
                          basis=basis.genie, subsets=subsets.genie,
                          hpa.start=hpa.genie.level2,control=jj)
```

---

H.fun *The H matrix*

---

## Description

Returns the matrix of bases H. The "app" of the function name means "approximator", to distinguish it from function `H.fun()` of the calibrator package.

## Usage

```
H.fun.app(D1, subsets, basis, hpa)
```

## Arguments

| | |
|---|---|
| `D1` | Design matrix for level 1 code |
| `subsets` | Subsets object |
| `basis` | Basis function |
| `hpa` | Hyperparameter object |

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
H.fun.app(D1.toy , subsets=subsets.toy , basis=basis.toy , hpa=hpa.toy)
```

---

## hdash.fun *Hdash*

---

### Description

Returns the thing at the top of page 6

### Usage

```
hdash.fun(x, hpa, basis)
```

### Arguments

| | |
|---|---|
| x | Point in question |
| hpa | Hyperparameter object |
| basis | Basis functions |

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
hdash.fun(x=1:3 , hpa=hpa.toy,basis=basis.toy)

uu <- rbind(1:3,1,1:3,3:1)
rownames(uu) <- paste("uu",1:4,sep="_")
hdash.fun(x=uu, hpa=hpa.toy,basis=basis.toy)
```

hpa.fun.toy *Toy example of a hyperparameter object creation function*

## Description

Creates a hyperparameter object from a vector of length 19. Intended as a toy example to be modified for real-world cases.

## Usage

```
hpa.fun.toy(x)
```

## Arguments

| | |
|---|---|
| `x` | Vector of length 19 that specifies the correlation scales |

## Details

Elements 1-4 of x specify the sigmas for each of the four levels in the toy example. Elements 5-7 specify the correlation scales for level 1, elements 8-10 the scales for level 2, and so on.

Internal function `pdm.maker()` shows how the B matrix is obtained from the various elements of input argument `x`. Note how, in this simple example, the B matrices are diagonal, but generalizing to non-diagonal matrices should be straightforward (if you can guarantee that they remain positive definite).

## Value

| | |
|---|---|
| `sigmas` | The four sigmas corresponding to the four levels |
| `B` | The four B matrices corresponding to the four levels |
| `rhos` | The three (sic) matrices corresponding to levels 1-3 |

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
hpa.fun.toy(1:19)
```

`is.consistent` *Checks observational data for consistency with a subsets object*

## Description

Checks observational data for consistency with a subsets object: the length of the vectors should match

## Usage

```
is.consistent(subsets, z)
```

## Arguments

| | |
|---|---|
| `subsets` | A subsets object |
| `z` | Data |

## Value

Returns `TRUE` or `FALSE` depending on whether z is consistent with subsets.

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## See Also

`is.nested`

## Examples

```
data(toyapps)
stopifnot(is.consistent(subsets.toy,z.toy))

z.toy[[4]] <- 1:6
is.consistent(subsets.toy,z.toy)
```

`mdash.fun` *Mean of Gaussian process*

### Description

Returns the mean of the Gaussian process conditional on the observations and the hyperparameters

### Usage

```
mdash.fun(x, D1, subsets, hpa, Vinv = NULL, use.Vinv = TRUE, z, basis)
```

### Arguments

| | |
|---|---|
| `x` | Point at which mean is desired |
| `D1` | Code design matrix for level 1 code |
| `subsets` | subsets object |
| `hpa` | Hyperparameter object |
| `Vinv` | Inverse of the variance matrix; if `NULL`, the function will calculate it |
| `use.Vinv` | Boolean, with default `TRUE` meaning to use the inverse of V and `FALSE` meaning to use a method that does not involve inverting V |
| `z` | observations |
| `basis` | Basis functions |

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
mdash.fun(x=1:3,D1=D1.toy,subsets=subsets.toy,hpa=hpa.toy,z=z.toy,basis=basis.toy)

uu <- rbind(1:3,1,3:1,1:3)
rownames(uu) <- c("first","second","third","fourth")

mdash.fun(x=uu,D1=D1.toy,subsets=subsets.toy,hpa=hpa.toy,z=z.toy,basis=basis.toy)
```

`object` *Optimization of posterior likelihood of hyperparameters*

### Description

Returns the likelihood of a set of hyperparameters given the data. Functions `opt1()` and `opt.gt.1()` find hyperparameters that maximize the relevant likelihood for level 1 and higher levels respectively. Function `object()` returns the expression given by equation 9 in KOH2000, which is minimized `opt1()` and `opt.gt.1()`.

### Usage

```
object(level, D, z, basis, subsets, hpa)
opt.1(D, z, basis, subsets, hpa.start, give.answers=FALSE, ...)
opt.gt.1(level, D, z, basis, subsets, hpa.start, give.answers=FALSE, ...)
```

### Arguments

| | |
|---|---|
| `level` | level |
| `D` | Design matrix for top-level code |
| `z` | Data |
| `basis` | Basis function |
| `subsets` | subsets object |
| `hpa` | hyperparameter object |
| `hpa.start` | Starting value for hyperparameter object |
| `give.answers` | Boolean, with default `FALSE` meaning to return just the point estimate, and `TRUE` meaning to return extra information from the call to `optim()` |
| `...` | Extra arguments passed to `optim()`. A common one would be `control=list(trace=100)` |

### Details

This function is the object function used in toy optimizers `optimal.hpa()`.

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### See Also

`genie`

### Examples

```
data(toyapps)
object(level=4, D=D1.toy , z=z.toy,basis=basis.toy,
   subsets=subsets.toy, hpa=hpa.fun.toy(1:19))
object(level=4, D=D1.toy , z=z.toy,basis=basis.toy,
   subsets=subsets.toy, hpa=hpa.fun.toy(3+(1:19)))


# Now a little example of finding optimal hyperpameters in the toy case
# (a bigger example is given on the genie help page)
jj <- list(trace=100,maxit=10)

hpa.toy.level1 <- opt.1(D=D1.toy, z=z.toy, basis=basis.toy,
          subsets=subsets.toy, hpa.start=hpa.toy,control=jj)

hpa.toy.level2 <- opt.gt.1(level=2, D=D1.toy, z=z.toy,
           basis=basis.toy, subsets=subsets.toy,
           hpa.start=hpa.toy.level1, control=jj)

hpa.toy.level3 <- opt.gt.1(level=3, D=D1.toy, z=z.toy,
           basis=basis.toy, subsets=subsets.toy,
           hpa.start=hpa.toy.level2, control=jj)

hpa.toy.level4 <- opt.gt.1(level=4, D=D1.toy, z=z.toy,
           basis=basis.toy, subsets=subsets.toy,
           hpa.start=hpa.toy.level3, control=jj)
```

---

Pi *Kennedy's Pi notation*

---

### Description

Evaluates Kennedy's $\prod$ product

### Usage

```
Pi(hpa, i, j)
```

### Arguments

| | |
|---|---|
| hpa | Hyperparameter object |
| i | subscript |
| j | superscript |

### Details

This function evaluates Kennedy's $\prod$ product, but with the additional feature that $\prod_i^j = 0$ if $i > j+1$. This seems to work in practice.

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
Pi(hpa.toy,1,2)
Pi(hpa.toy,2,2)
Pi(hpa.toy,3,2)
Pi(hpa.toy,4,2)
```

---

## `subsets.fun` *Generate and test subsets*

---

### Description

Create a list of subsets (`subsets.fun()`); or, given a list of subsets, test for correct inclusion (`is.nested()`), or strict inclusion (`is.strict()`).

### Usage

```
is.nested(subsets)
is.strict(subsets)
subsets.fun(n, levels = 4, prob = 0.7)
```

### Arguments

| | |
|---|---|
| `subsets` | In `is.nested()`, a list of subsets to be tested |
| `n` | Number of observations in the lowest level (ie level 1, the fastest code) |
| `levels` | Number of levels |
| `prob` | Probability of choosing an observation at level $n + 1$ given that there is one at the same place at level $n$ |

### Author(s)

Robin K. S. Hankin (`subsets.fun()`); Peter Dalgaard (via R-help)

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
is.nested(subsets.fun(20))  # Should be TRUE

data(toyapps)
stopifnot(is.nested(subsets.toy))
```

---

subset_maker *Create a simple subset object*

---

## Description

Given an integer vector whose $i^{\mathrm{th}}$ element is the number of runs at level $i$, return a subset object in echelon form.

## Usage

```
subset_maker(x)
```

## Arguments

| | |
|---|---|
| x | A vector of integers |

## Details

In this context, `x` being in "echelon form" means that

- `x` is consistent in the sense of passing `is.consistent()`
- For each $i$, `x[[i]] = 1:n` for some $n$.

## Value

A list object suitable for use as a `subset` object

## Author(s)

Robin K. S. Hankin

## See Also

`is.consistent`, `is.nested`, `is.strict`

## Examples

```
subset_maker(c(10,4,3))


is.nested(subset_maker(c(4,9,6))) #should be FALSE
is.nested(subset_maker(c(9,6,4))) #should be TRUE
```

## tee.fun *Returns generalized distances*

### Description

Returns generalized distances from a point to the design matrix as per equation 10

### Usage

```
tee.fun(x, D1, subsets, hpa)
```

### Arguments

| | |
|---|---|
| x | Point in parameter space |
| D1 | Design matrix for level 1 code |
| subsets | subsets object |
| hpa | Hyperparameter object |

### Details

See equation 10

### Author(s)

Robin K. S. Hankin

### References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

### Examples

```
data(toyapps)
tee.fun(x=1:3, D1=D1.toy, subsets=subsets.toy, hpa=hpa.toy)
```

## toyapps — *Toy datasets for approximator package*

### Description

Toy datasets that illustrate the package.

### Usage

```
data(toyapps)
D1.toy
hpa.toy
z.toy
subsets.toy
betas.toy
```

### Format

The toy example is a case with four levels.

The `D1.toy` matrix is 20 rows of code run points, corresponding to the observations of the level 1 code. It has three columns, one per parameter.

`hpa.toy` is a hyperparameter object. It is a list of three elements: `sigmas`, `B`, and `rhos`.

`subsets.toy` is a list of four elements. Element $i$ corresponds to the rows of `D1.toy` at which level $i$ has been observed.

`z.toy` is a four element list. Each element is a vector; element $i$ corresponds to obsevations of level $i$. The lengths will match those of `subsets.toy`.

`betas.toy` is a matrix of coefficients.

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

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

Function `basis.toy()` is an example of a basis function

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

**See the helpfiles listed in the "see also" section below**

### Details

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

### Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(toyapps)

is.consistent(subsets.toy , z.toy)

generate.toy.observations(D1.toy, subsets.toy, basis.toy, hpa.toy, betas.toy)
```

---

# V.fun.app *Variance matrix*

---

## Description

Given a design matrix, a subsets object and a hyperparameter object, return the variance matrix. The "app" of the function name means "approximator", to distinguish it from function `V.fun()` of the calibrator package.

## Usage

```
V.fun.app(D1, subsets, hpa)
```

## Arguments

| | |
|---|---|
| `D1` | Design matrix for level 1 code |
| `subsets` | Subsets object |
| `hpa` | Hyperparameter object |

## Author(s)

Robin K. S. Hankin

## References

M. C. Kennedy and A. O'Hagan 2000. "Predicting the output from a complex computer code when fast approximations are available" Biometrika, 87(1): pp1-13

## Examples

```
data(toyapps)
V.fun.app(D1.toy,subsets.toy,hpa.toy)
```

# Index