# Package 'quack'

June 23, 2022

**Title** Methods for Quantification of Uncertainty and Calibration for Physical Parameters

**Version** 0.0.0.9000

**Description** Functions for efficient and robust UQ.

**License** `use_mit_license()`, `use_gpl3_license()` or friends to
pick a license

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**Suggests** knitr,
rmarkdown

**VignetteBuilder** knitr

**Imports** RANN,
laGP,
lhs,
MHadaptive,
rootSolve

## R topics documented:

---

`boot_accel` *Accelerated Bootstrap*

---

## Description

Flexible code for constructing a confidence interval using accelrated bootstrap

## Usage

```
boot_accel(x, est_theta, B = 10000, alpha = 0.042, a = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | the data vector |
| `est_theta` | function to estimate parameter |
| `B` | number of bootstrap samples |
| `alpha` | significance level. Default is 0.042 (because 0.05 is arbitrary) |
| `a` | acceleration constant. See details |
| `...` | additional parameters passed to est_theta |

## Details

By default, the acceleration constant is a=NULL which leads to estimation of the constant using the est_accel() function. This can be skipped by specifying a particular value for a. Note that a=0 corresponds to the usual "quantile" bootstrap.

The code uses a special variable called 'tmp_indx' which can be exploited to handle more complicated cases using the dynGet() function. See the example below for more details.

## Examples

```
#Simulate data
x <- rnorm(50)
y <- x + rnorm(50)
#Simple example
boot_accel(x, function(z) quantile(z, 0.75) , B=1e4)

#Bootstrap for correlation using dynGet()
my_cor <- function(x, y, indx=dynGet('tmp_indx')){
```

```
   cor(x, y[indx])
}
boot_accel(x, my_cor, y=y)
```

---

`corr_matrix` *Correlation Matrix*

---

## Description

Genersates the power exponential or Matern correlation matrix for a set of n design points in d-dimensional space. Default is Gaussian (power=2). See GPfit::corr_matrix for details.

## Usage

```
corr_matrix(X, beta, corr = list(type = "exponential", power = 2))
```

## Arguments

| | |
|---|---|
| `X` | nxd matrix of design points |
| `beta` | correlation parameter. If 1/kappa is length scale then beta=log10(kappa). |
| `corr` | list specifying the correlation function. See GPfit package for details. |

## Examples

```
R <- corr_matrix(seq(0, 1, length.out=10), beta=log10(1.0))
```

---

`diagnose_ppc` *Diagnostic Plot for Probability of Prior Coherency*

---

## Description

Diagnostic plot for detecting overfitting using probability of prior coherency

## Usage

```
diagnose_ppc(M, V, p, MC = 10000, control = NULL, ...)
```

## Arguments

| | |
|---|---|
| `M` | a vector of posterior M values (mean of a parameter set) |
| `V` | a vector of posterior V values (variance of a parameter set) |
| `p` | dimension of the parameter set |
| `MC` | number of Monte Carlo samples. |
| `control` | a list of control parameters for plotting. See details. |
| `...` | additional parameters passed to plot() function. |

## Details

control is a named list consisting of any of the following.

levels - the desired contour levels for plotting (default c(0.5, 0.9, 0.95, 0.99))

h - step-size for plotting contours (default 0.005)

tol - tolerance for plotting contours (default 1e-7)

prob - a probability bound for setting xlim and ylim (default 0.9999)

fill_col - the color of the contours regions (default 'dodgerblue')

point_col - the color of the points (default 'orange')

cex - 0.2 the size of the points representing PPC posterior samples (default 0.5)

leg_cex - the size of the legend (default 1.4)

## Examples

```
X <- rMP(100, 7, 1, 3)
```

---

# dMP *Density of the Moment Penalization Prior*

---

## Description

Returns the unnormalized density of the MP prior with parameters w1 and w2

## Usage

```
dMP(x, w1 = 1, w2 = 1, log = FALSE, normalized = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | a vector of length p |
| w1 | normalized penalty associated with second moment. Default is 1 |
| log | logical. Should density be returned on a log scale? |
| normalized | logical. Should density be normalized (default is FALSE) |
| ... | additional parameters passed to get_constMP (if norm=TRUE) |

## Value

returns the density of the MP(w1, w2) prior

## Examples

```
x <- rnorm(10)
dMP(x, w1=5, w2=2)
X <- matrix(rnorm(10*30), nrow=30)
apply(X, 1, dMP, w1=5, w2=2)
```

## d_mahal *Mahalanobis Distance*

### Description

Computes the mahalanobis distance between two vectors given a precision matrix S

### Usage

```
d_mahal(x, y = NULL, S = diag(rep(1, length(x))))
```

### Arguments

| | |
|---|---|
| x | a vector |
| y | an optional vector. The zero vector by default |
| S | a precision matrix of dimension equal to length(x) |

### Examples

```
x <- rnorm(5); y <- rnorm(5); S <- GPfit::corr_matrix((1:5)/6, beta=0.2)
d_mahal(x, y, S)
```

## ECP *Emulation of the Conditional Posterior (General)*

### Description

This function models the conditional posterior distribution of alpha conditional on a set of nuisance parameters gamma. Draws from the modularization posterior can be obtained by pairing this function with ECP_sample. This function is designed to give maximal flexibility. Consider using ECP_norm for a more straightforward implementation and ECP_multi for multiple alpha

### Usage

```
ECP(
  lpost,
  L = 30,
  init = NULL,
  MCMC = list(pars = NULL, iterations = 5000, burn_in = 1000, thin = 1, prop_sigma =
    NULL, adapt_par = c(100, 20, 0.5, 0.75)),
  f_alpha = function(x) x[1],
  estimate_psi = function(x) c(mean(x), sd(x)),
  gamma_generate = NULL,
  verbose = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| lpost | The log posterior. First argument must be a vector of parameters for which full Bayesian inference is desired. Second argument is a vector of nuisance parameters for which modularization is desired. Second argument must be named "gamma". |
| L | the budget. |
| init | a set of initial values for alpha. Can also be a function which generates starting values based on gamma |
| MCMC | a named list of arguments for the Metro_Hastings function. |
| f_alpha | an optional function for transforming the physical parameters. Helpful for reducing to a single dimension. |
| estimate_psi | function whose argument is the result of f_alpha(al), where al is a posterior sample from the conditional posterior. Should return a vector of r parameter estimates. Defaults to univariate normal. |
| gamma_generate | can be either (i) an L x q matrix of nuisance parameters, (ii) a function (with no arguments) that generates a single instance of gamma (length q) from the prior or (iii) a named list with two components (mu and sigma) which are q-length vectors of prior mean/sd's. Please ensure that class(gamma_generate) is either 'matrix', 'function' or 'list'. |
| ... | additional arguments passed to lpost |

### Examples

```
set.seed(42)
a <- 0; g <- 0.1
n1 <- 10; n2 <- 90
y <- rnorm(n1+n2, a + c(rep(0, n1), rep(g, n2)), 1)
lpost <- function(a, gamma, y, n1, n2){
  res <- sum(dnorm(y, a + c(rep(0, n1), rep(gamma, n2)), 1, log=TRUE))
  res <- res + dnorm(a, 0, 1, log=TRUE) + dnorm(gamma, 0, 1, log=TRUE)
  return(res)
}
ecp <- ECP(lpost, L=30, init=0, gamma_generate=list(mu=0, sigma=1), y=y, n1=n1, n2=n2)
hist(ECP_sample(ecp, M=1000, gamma_generate=list(mu=0, sigma=1)), breaks=30)
ECP_profile_plot(ecp)
```

---

## ECP_profile_plot *Profile Plot for ECP Algorithm*

---

### Description

Plots quantiles of the posterior for alpha conditional on gamma.

### Usage

```
ECP_profile_plot(
  ecp,
  bounds = c(0, 1, 0.01),
  fix = NULL,
  quantiles = c(0.95, 0.5, 0.05),
  ...
)
```

### Arguments

| | |
|---|---|
| ecp | an object generated from the ecp function (requires normality) |
| bounds | defines the bounds of gamma (from, to, by). Should be a q x 3 matrix when there are multiple gamma. |
| fix | when q > 1, what values should the remaining modularization parameters be fixed to? |
| quantiles | the quantiles of the conditional distribution to plot |
| ... | additional arguments passed to plot() and lines() |

### Examples

```
set.seed(42)
a <- 0; g <- 0.1
n1 <- 10; n2 <- 90
y <- rnorm(n1+n2, a + c(rep(0, n1), rep(g, n2)), 1)
lpost <- function(a, gamma, y, n1, n2){
  res <- sum(dnorm(y, a + c(rep(0, n1), rep(gamma, n2)), 1, log=TRUE))
  res <- res + dnorm(a, 0, 1, log=TRUE) + dnorm(gamma, 0, 1, log=TRUE)
  return(res)
}
ecp <- ECP(lpost, L=30, init=0, gamma_generate=list(mu=0, sigma=1), y=y, n1=n1, n2=n2)
hist(ECP_sample(ecp, M=1000, gamma_generate=list(mu=0, sigma=1)), breaks=30)
ECP_profile_plot(ecp)
```

---

## ECP_sample *Sampling from the Modularization Posterior with ECP*

---

### Description

This function produces draws from the Modularization Posterior of alpha (with respect to gamma).

### Usage

```
ECP_sample(ecp, M = 1, gamma_generate = NULL, alpha_generate = NULL)
```

### Arguments

| | |
|---|---|
| ecp | An object returned from the ECP() function. |
| M | number of samples requested |
| gamma_generate | can be either (i) an L x q matrix of nuisance parameters, (ii) a function (with no arguments) that generates a single instance of gamma (length q) from the prior or (iii) a named list with two components (mu and sigma) which are q-length vectors of prior mean/sd's. Please ensure that class(gamma_generate) is either 'matrix', 'function' or 'list'. |
| alpha_generate | a function to generate random samples of alpha conditional on psi. Should correspond to the estimate_psi function used in ECP() function. Takes one argument: psi and returns a single alpha draw. Defaults to univariate normal. |

## Examples

```
set.seed(42)
a <- 0; g <- 0.1
n1 <- 10; n2 <- 90
y <- rnorm(n1+n2, a + c(rep(0, n1), rep(g, n2)), 1)
lpost <- function(a, gamma, y, n1, n2){
  res <- sum(dnorm(y, a + c(rep(0, n1), rep(gamma, n2)), 1, log=TRUE))
  res <- res + dnorm(a, 0, 1, log=TRUE) + dnorm(gamma, 0, 1, log=TRUE)
  return(res)
}
ecp <- ECP(lpost, L=30, init=0, gamma_generate=list(mu=0, sigma=1), y=y, n1=n1, n2=n2)
hist(ECP_sample(ecp, M=1000, gamma_generate=list(mu=0, sigma=1)), breaks=30)
ECP_profile_plot(ecp)
```

---

# est_accel *Estimation of the "Accelration Constant"*

---

## Description

Estimates the accleration constant using a simple jackknife estimator (Efron 1987).

## Usage

```
est_accel(x, est_theta, ...)
```

## Arguments

| | |
|---|---|
| x | the data vector |
| est_theta | function to estimate parameter |

## Examples

```
est_accle(rnorm(30), function(z) quantile(z, 0.75))
```

---

# fast_det *Fast Matrix Algebra for BMC*

---

## Description

Function for obtaining the determinant in linear time - after pre-processing - for covariance matrices of suitable structure

## Usage

```
fast_det(FM, phi = NULL, tau = NULL, log = TRUE)
```

### Arguments

| | |
|---|---|
| FM | an object created by the fast_process() function |
| phi | current value of phi (can be omitted if phi is stored in FM object) |
| tau | current value of tau (can be omitted if tau is stored in FM object) |
| log | logical. Should log-determinant be returned? Default is TRUE. |

### Value

Returns the (log) determinant of the covariance matrix.

### Examples

```
n <- 100
R <- corr_matrix(seq(0,1,length.out=n), beta=0.2)
FM <- fast_process(R, method='approx', control=list(tol=1e-9))
#Compute determinant (approximately)
fast_det(FM, phi=1.5, tau=0.2)
#Compute inverse (approximately)
fast_inv(FM, phi=1.5, tau=0.2)
```

---

## fast_inv *Fast Matrix Algebra for BMC*

---

### Description

See fast_inv.class for details.

### Usage

```
fast_inv(FM, ...)
```

### Arguments

| | |
|---|---|
| FM | an object of class "FM_fast", "FM_approx" or "FM_ts" |
| ... | appropriate arguments for the corresponding class. |

### Value

Returns (possibly an approximation of) the inverse of the covariance matrix

## fast_inv.FM_approx *Fast Matrix Algebra for BMC*

### Description

Function for (approximately) computing the inverse of a BMC Covariance matrix. Near-quadratic runtime leads to significant time-savings when covariance matrix is moderate to large.

### Usage

```
## S3 method for class 'FM_approx'
fast_inv(FM, phi = NULL, tau = NULL, ...)
```

### Arguments

| | |
|---|---|
| FM | an object of class "FM_approx" |
| phi | current value of phi (can be omitted if phi is stored in FM object) |
| tau | current value of tau (can be omitted if tau is stored in FM object) |
| ... | appropriate arguments for the corresponding class. |

### Value

Returns an approximation of the inverse of the covariance matrix

### Examples

```
n <- 100
R <- corr_matrix(seq(0,1,length.out=n), beta=0.2)
FM <- fast_process(R, method='approx', control=list(tol=1e-3))
#Compute determinant (approximately)
fast_det(FM, phi=1.5, tau=0.2)
#Compute inverse (approximately)
fast_inv(FM, phi=1.5, tau=0.2)
```

## fast_inv.FM_fast *Fast Matrix Algebra for BMC*

### Description

Function for computing the inverse of a BMC Covariance matrix. Leads to speedup of roughly 2 when matrix dimension is moderate or large.

### Usage

```
## S3 method for class 'FM_fast'
fast_inv(FM, phi = NULL, tau = NULL, ...)
```

### Arguments

| | |
|---|---|
| FM | an object of class "FM_fast" |
| phi | current value of phi (can be omitted if phi is stored in FM object) |
| tau | current value of tau (can be omitted if tau is stored in FM object) |
| ... | appropriate arguments for the corresponding class. |

### Value

Returns the inverse of the covariance matrix

---

## fast_inv.FM_ts *Fast Matrix Algebra for BMC*

---

### Description

Function for (approximately) computing the inverse of a BMC Covariance matrix. Near-quadratic runtime leads to significant time-savings when covariance matrix is moderate to large.

### Usage

```
## S3 method for class 'FM_ts'
fast_inv(FM, phi = NULL, ...)
```

### Arguments

| | |
|---|---|
| FM | an object of class "FM_ts" |
| phi | current value of phi (can be omitted if phi is stored in FM object) |
| ... | appropriate arguments for the corresponding class. |

### Value

Returns an approximation of the inverse of the covariance matrix

### Examples

```
n <- 100
R <- corr_matrix(seq(0,1,length.out=n), beta=0.2)
FM <- fast_process(R, method='approx', control=list(tol=1e-9))
#Compute determinant (approximately)
fast_det(FM, phi=1.5, tau=0.2)
#Compute inverse (approximately)
fast_inv(FM, phi=1.5, tau=0.2)
```

## fast_process *Fast Matrix Algebra for BMC*

### Description

Functions for obtaining matrix inversions and determinants quickly in the context of BMC. Processes matrices of the form Sigma = phi x R + tau x I

### Usage

```
fast_process(R, phi = NULL, tau = NULL, method = "fast", control = list())
```

### Arguments

| | |
|---|---|
| R | fixed correlation matrix |
| phi | Optional. Indicates that phi is fixed throughout the analysis. |
| tau | Optional. Indicates that tau is fixed throughout the analysis. Required when method == 'ts' |
| method | Must be 'fast', 'approx', or 'ts'. |
| control | a list of control parameters. Will be ignored when method == 'fast'. Method 'approx' needs control$tol (1e-9) recomended, and method 'ts' needs an expansion value control$a and an expansion order control$P |

### Value

returns an object of class "FM_method" containing the information needed for fast computation of matrix inverse and determinant

### Examples

```
n <- 100
R <- corr_matrix(seq(0,1,length.out=n), beta=0.2)
FM <- fast_process(R, method='approx', control=list(tol=1e-9))
#Compute determinant (approximately)
fast_det(FM, phi=1.5, tau=0.2)
#Compute inverse (approximately)
fast_inv(FM, phi=1.5, tau=0.2)
```

## get_constMP *Normalizing Constant for the Moment Penalization Prior*

### Description

Returns the normalizing constant for the Moment Penalization Prior with parameters p, w1 and w2

### Usage

```
get_constMP(p, w1 = 1, w2 = 1, MC = 1e+07)
```

### Arguments

| | |
|---|---|
| p | the dimension of the vector having an MP prior |
| w1 | normalized penalty associated with second moment. Default is 1 |
| MC | number of MC iterations. Default is 1e7, but larger values are recommended when speed is not important. |

### Value

returns the normalizing constant of the MP prior

### Examples

```
get_constMP(7, 1, 3, MC=1e9)
```

---

## inside_CR *Is Inside Credible Region*

---

### Description

Checks to see whether a 2-D query point is inside, on or outside the convex hull representing an empirical credible region

### Usage

```
inside_CR(q, CR)
```

### Arguments

| | |
|---|---|
| q | a query point |
| CR | the vertices of the convex hull returned by joint_CR (with two_dim = TRUE) |

### Details

Only implemented (currently) for 2 dimensions.

### Value

Returns +1, 0, or -1 for inside JCR, on JCR or outside JCR respectively.

joint_CR *Joint Confidence Regions using Mahalanobis Distance*

## Description

Finds the minimal convex hull, with respect to Mahalanobis distance, which contains (1-alpha)x100% of the points.

## Usage

```
joint_CR(X, alpha = 0.042, two_dim = TRUE)
```

## Arguments

| | |
|---|---|
| X | a matrix - each row represents a sample |
| alpha | the confidence level is 1-alpha. Defaults to 0.042 (because 0.05 is so arbitrary) |
| two_dim | logical. If TRUE, then the convex hull surrounding the interior points in the first two dimensions is returned. If FALSE, then we return the entire set of interior points (for arbitrary dimension). |

## Details

Note that a set of N-dimensional points is always returned. When two_dim=TRUE, this set of points represents the points on the convex hull of the JCR - but only in the first two dimensions. This is a limitation of the built-in chull() function. Future versions may extend the functionality. For now, one can operate in higher dimensions by setting two_dim = FALSE to return the full set of (1-alpha)*N points which are in the interior of the JCR.

## Value

the samples laying on the convex hull which contains (1-alpha)100% of the samples.

## Examples

```
x <- rnorm(1000); y <- x + rnorm(1000, 0, .5)
X <- cbind(x, y)
CR <- joint_CR(X)
plot(X)
polygon(CR, border='blue', lwd=2)

#A point inside of JCR
inside_CR(c(0,0), CR)
#A point on the convex hull of JCR
lambda <- 0.3
inside_CR(lambda*CR[3,] + (1-lambda)*CR[4,], CR)
#A point outside of JCR
inside_CR(c(-3, 7), CR)
```

leapGP *Localized Ensemble of Approximate Gaussian Processes*

## Description

A wrapper for combining training and prediction for leapGP

## Usage

```
leapGP(
  Xnew,
  X,
  Y,
  H = NA,
  scale = F,
  n = NA,
  frac = FALSE,
  start = NA,
  verbose = TRUE,
  justdoit = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| X | a matrix of training locations (1 row for each training instance) |
| Y | a vector of training responses (length(y) == nrow(X)) |
| H | the number of prediction hubs desired. Defaults to ceiling(sqrt(length(Y))). |
| scale | logical. Do we want the scale parameter to be returned for predictions? If TRUE, the matrix K^-1 will be stored for each hub. |
| n | local neighborhood size |
| frac | logical. If TRUE, information is returned about the fraction of training points which are in their own prediction neighborhoods. |
| verbose | logical. Deault is FALSE |
| justdoit | logical. Force leapGP to run using specified parameters (may be incredibly time consuming). |
| ... | optional arguments to be passed to laGP() |
| iso | boolean. Is correlation function isotropic? (Current version ignores this argument) |

## Value

a univariate prediction and an updated list of hubs. Also returns scale parameter if scale=TRUE

## Examples

```
Xnew <- matrix(runif(100), nrow=50, ncol=2)
X <- matrix(runif(100), nrow=50, ncol=2)
Y <- apply(X, 1, prod)
preds1 <- leapGP(Xnew, X, Y)

#Or equivalently
leap <- leapGP_build(X, Y)
preds2 <- rep(NA, 100)
for(i in 1:100){
   preds2[i] <- leapGP_predict(leap, Xnew[i,])
}

#Or used with slapGP
leap <- leapGP_synch(leap, rho=0.95)
hubs <- leap$hubs
preds3 <- rep(NA, 100)
for(i in 1:100){
   emulator <- slapGP(Xnew[i,], X, Y, hubs=hubs)
   preds3[m] <- emulator$pred
   hubs <- emulator$hubs
}
```

---

`leapGP_build` *Localized Ensemble of Approximate Gaussian Processes*

---

## Description

This function is a modification of the LA-GP framework of Gramacy and Apley designed for cases where parallel predictions are not possible (i.e. MCMC). The leapGP offers a quadratic training algorithm which leads to fast predictions.

## Usage

```
leapGP_build(
  X,
  Y,
  H = NA,
  scale = F,
  iso = T,
  n = NA,
  start = NA,
  frac = TRUE,
  verbose = FALSE,
  justdoit = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `X` | a matrix of training locations (1 row for each training instance) |
| `Y` | a vector of training responses (length(y) == nrow(X)) |

| | |
|---|---|
| H | the number of prediction hubs desired. Defaults to ceiling(sqrt(length(Y))). |
| scale | logical. Do we want the scale parameter to be returned for predictions? If TRUE, the matrix K^-1 will be stored for each hub. |
| iso | boolean. Is correlation function isotropic? (Current version ignores this argument) |
| n | local neighborhood size |
| start | number of starting points for neighborhood (between 6 and n inclusive) |
| frac | logical. If TRUE, information is returned about the fraction of training points which are in their own prediction neighborhoods. |
| verbose | logical. Deault is FALSE |
| justdoit | logical. Force leapGP to run using specified parameters (may be incredibly time consuming). |
| ... | optional arguments to be passed to laGP() |

## Value

a univariate prediction and an updated list of hubs. Also returns scale parameter if scale=TRUE

## Examples

```
Xnew <- matrix(runif(100), nrow=50, ncol=2)
X <- matrix(runif(100), nrow=50, ncol=2)
Y <- apply(X, 1, prod)
preds1 <- leapGP(Xnew, X, Y)

#Or equivalently
leap <- leapGP_build(X, Y)
preds2 <- rep(NA, 100)
for(i in 1:100){
   preds2[i] <- leapGP_predict(leap, Xnew[i,])
}

#Or used with slapGP
leap <- leapGP_synch(leap, rho=0.95)
hubs <- leap$hubs
preds3 <- rep(NA, 100)
for(i in 1:100){
   emulator <- slapGP(Xnew[i,], X, Y, hubs=hubs)
   preds3[m] <- emulator$pred
   hubs <- emulator$hubs
}
```

---

# leapGP_predict *Localized Ensemble of Approximate Gaussian Processes*

---

## Description

A function for prediction with an object of class leapGP.

## Usage

```
leapGP_predict(leapGP, Xnew, scale = F, iso = T, ...)
```

### Arguments

| | |
|---|---|
| leapGP | an object of class "leapGP" |
| Xnew | the location at which prediction is requested |
| scale | logical. Do we want the scale parameter to be returned for predictions? If TRUE, the matrix K^-1 will be stored for each hub. |
| iso | logical. Is correlation function isotropic? (Currently not supported) |

### Value

an object of class "leapGP" AND of class "slapGP"

### Examples

```
Xnew <- matrix(runif(100), nrow=50, ncol=2)
X <- matrix(runif(100), nrow=50, ncol=2)
Y <- apply(X, 1, prod)
preds1 <- leapGP(Xnew, X, Y)

#Or equivalently
leap <- leapGP_build(X, Y)
preds2 <- rep(NA, 100)
for(i in 1:100){
   preds2[i] <- leapGP_predict(leap, Xnew[i,])
}

#Or used with slapGP
leap <- leapGP_synch(leap, rho=0.95)
hubs <- leap$hubs
preds3 <- rep(NA, 100)
for(i in 1:100){
   emulator <- slapGP(Xnew[i,], X, Y, hubs=hubs)
   preds3[m] <- emulator$pred
   hubs <- emulator$hubs
}
```

---

## leapGP_synch *Localized Ensemble of Approximate Gaussian Processes*

---

### Description

This function synchronizes an object of class "leapGP", so that is also an object of "slapGP"

### Usage

```
leapGP_synch(leapGP, rho = 0.95)
```

### Arguments

| | |
|---|---|
| leapGP | an object of class "leapGP" |
| rho | parameter controlling time-accuracy tradeoff (default = 0.95) |

## Value

an object of class "leapGP" AND of class "slapGP"

## Examples

```
Xnew <- matrix(runif(100), nrow=50, ncol=2)
X <- matrix(runif(100), nrow=50, ncol=2)
Y <- apply(X, 1, prod)
preds1 <- leapGP(Xnew, X, Y)

#Or equivalently
leap <- leapGP_build(X, Y)
preds2 <- rep(NA, 100)
for(i in 1:100){
   preds2[i] <- leapGP_predict(leap, Xnew[i,])
}

#Or used with slapGP
leap <- leapGP_synch(leap, rho=0.95)
hubs <- leap$hubs
preds3 <- rep(NA, 100)
for(i in 1:100){
   emulator <- slapGP(Xnew[i,], X, Y, hubs=hubs)
   preds3[m] <- emulator$pred
   hubs <- emulator$hubs
}
```

---

# ppc *Probability of Prior Coherency*

---

## Description

Computes the probability of prior coherency using Monte Carlo

## Usage

```
ppc(M, V, p, type = 1, MC = 10000)
```

## Arguments

| | |
|---|---|
| M | a vector of posterior M values (mean of a parameter set) |
| V | a vector of posterior V values (variance of a parameter set) |
| p | dimension of the parameter set |
| type | indicates what should be returned. See details |
| MC | number of Monte Carlo samples. |

## Details

type = 1 leads to PPC calculation based on posterior mean of M and V (as described in paper)

type = 2 is the posterior mean of PPC

type = 3 returns posterior samples of PPC

type = 4 returns additional information (used for plotting)

## Examples

```
gamma <- cbind(rnorm(1000, 1.2, 0.2),
               rnorm(1000, -0.4, 0.3),
               rnorm(1000, -0.6, 0.15))
M <- apply(gamma, 1, mean)
V <- apply(gamma, 1, var)
ppc(M, V, 3)
diagnose_ppc(M, V, p, control=list(levels=c(0.5, 0.9, 0.99), fill_col='purple'))
```

---

# rMP *Random Sampling from the Moment Penalization Prior*

---

## Description

Uses an optimized rejection sampling framework to sample from the MP prior

## Usage

```
rMP(n, p, w1, w2, acceptance = FALSE)
```

## Arguments

| | |
|---|---|
| n | number of samples requested |
| p | dimension of random vector |
| w1 | normalized penalty associated with second moment. Default is 1 |
| acceptance | logical. If TRUE, a list is returned with an attribute giving the acceptance rate. |
| ... | additional parameters passed to get_constMP (if norm=TRUE) |

## Value

returns the density of the MP(w1, w2) prior

## Examples

```
X <- rMP(100, 7, 1, 3)
```

---

# rZreg *Random Sampling from Z-Regularization Prior*

---

## Description

Draws samples from Z-regularization prior. Alternative to MP prior with w1=w2=Inf (when sigma_R = 0).

## Usage

```
rZreg(n, p, sigma_R = 0)
```

### Arguments

| | |
|---|---|
| n | number of samples requested |
| p | dimension of random vector |
| sigma_R | relaxation parameter (default 0) |

### Value

returns the density of the Z regularization prior

### Examples

```
X <- rZreg(100, 7)
```

---

## slapGP *Sequence of Local Approximate Gaussian Processes*

---

### Description

This function is a modification of the LA-GP framework of Gramacy and Apley designed for cases where parallel predictions are not possible (i.e. MCMC). The slapGP framework offers users a time-accuracy tradeoff based on the rho parameter.

### Usage

```
slapGP(
  Xnew,
  X,
  Y,
  rho = 0.95,
  hubs = list(),
  scale = F,
  iso = TRUE,
  n = NA,
  start = NA,
  ...
)
```

### Arguments

| | |
|---|---|
| Xnew | the location at which prediction is requested |
| X | a matrix of training locations (1 row for each training instance) |
| Y | a vector of training responses (length(y) == nrow(X)) |
| rho | parameter controlling time-accuracy tradeoff (default = 0.95) |
| hubs | a list of current prediction hubs |
| scale | logical. Do we want the scale parameter to be returned for predictions? If TRUE, the matrix K^-1 will be stored for each hub. |
| iso | logical. Is correlation function isotropic? (Currently not supported) |
| n | local neighborhood size |
| start | number of starting points for neighborhood (between 6 and n inclusive) |
| ... | optional arguments to be passed to laGP() |

## Value

a univariate prediction and an updated list of hubs. Also returns scale parameter if scale=TRUE

## Examples

```
hubs <- list()
X <- matrix(runif(100), nrow=50, ncol=2)
Y <- apply(X, 1, prod)
preds <- rep(NA, 1000)
for(i in 1:1000){
   Xnew <- runif(2)
   emulator <- slapGP(Xnew, X, Y, hubs=hubs)
   hubs <- emulator$hubs
   preds[i] <- emulator$pred
}
```

# Index