Package ‘deepgp’

December 7, 2021

Type Package
Title Sequential Design for Deep Gaussian Processes using MCMC
Version 0.3.1
Date 2021-11-23
Author Annie Sauer <anniees@vt.edu>
Maintainer Annie Sauer <anniees@vt.edu>
Depends R (>= 3.6)
Description Performs model fitting and sequential design for deep Gaussian processes following Sauer, Gramacy, and Higdon (2020) <arXiv:2012.08015>. Models extend up to three layers deep; a one layer model is equivalent to typical Gaussian process regression. Covariance kernel options are Matern (default) and squared exponential. Sequential design criteria include integrated mean-squared error (IMSE), active learning Cohn (ALC), and expected improvement (EI). Applicable to both noisy and deterministic functions. Incorporates SNOW parallelization and utilizes C and C++ under the hood.
License LGPL
Encoding UTF-8
NeedsCompilation yes
Imports grDevices, graphics, stats, doParallel, foreach, parallel, Rcpp, mvtnorm
LinkingTo Rcpp, RcppArmadillo, BH
Suggests akima, knitr
RoxygenNote 7.1.1
Repository CRAN
Date/Publication 2021-12-07 08:00:07 UTC

R topics documented:

deepgp-package ...................................................... 2
**Package deepgp**

**Description**

Performs model fitting and sequential design for deep Gaussian processes following Sauer, Gramacy, and Higdon (2020) <arXiv:2012.08015>. Models extend up to three layers deep; a one layer model is equivalent to typical Gaussian process regression. Both Matern and squared exponential kernels are implemented. Sequential design criteria include integrated mean-squared error (IMSE), active learning Cohn (ALC), and expected improvement (EI). Applicable to both noisy and deterministic functions. Incorporates SNOW parallelization and utilizes C and C++ under the hood.

**Important Functions**

- **fit_one_layer**: conducts MCMC sampling of hyperparameters for a one layer GP
- **fit_two_layer**: conducts MCMC sampling of hyperparameters and hidden layer for a two layer deep GP
- **fit_three_layer**: conducts MCMC sampling of hyperparameters and hidden layers for a three layer deep GP
- **continue**: collects additional MCMC samples
- **trim**: cuts off burn-in and optionally thins samples
- **predict**: calculates posterior mean and variance over a set of input locations (optionally calculates EI)
- **plot**: produces trace plots, hidden layer plots, and posterior plots
- **ALC**: calculates active learning Cohn over set of input locations using reference grid
- **IMSE**: calculates integrated mean-squared error over set of input locations

**Author(s)**

Annie Sauer <anniees@vt.edu>
References


Examples

# 1. One Layer and EI ---------------------------------------------------------

```r
f <- function(x) {
  sin(5 * pi * x) / (2 * x) + (x - 1) ^ 4
}

# Training data
x <- seq(0.5, 2, length = 30)
y <- f(x) + rnorm(30, 0, 0.01)

# Testing data
xx <- seq(0.5, 2, length = 100)
yy <- f(xx)

# Standardize inputs and outputs
xx <- (xx - min(x)) / (max(x) - min(x))
x <- (x - min(x)) / (max(x) - min(x))

# Conduct MCMC
fit <- fit_one_layer(x, y, nmmcm = 10000)
plot(fit) # investigate trace plots
fit <- trim(fit, 8000, 2)

# Predict and calculate EI
```
fit <- predict(fit, xx, EI = TRUE)

# Visualize Fit
plot(fit)
par(new = TRUE) # overlay EI
plot(xx, fit$EI, type = 'l', lty = 2, axes = FALSE, xlab = '', ylab = '')

# Select next design point
x_new <- xx[which.max(fit$EI)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

# 2. Two Layer and ALC ------------------------------------------------------

f <- function(x) {
  exp(-10 * x) * (cos(10 * pi * x - 1) + sin(10 * pi * x - 1)) * 5 - 0.2
}

# Training data
x <- seq(0, 1, length = 30)
y <- f(x) + rnorm(30, 0, 0.05)

# Testing data
xx <- seq(0, 1, length = 100)
yy <- f(xx)

# Conduct MCMC
fit <- fit_two_layer(x, y, D = 1, nmcmc = 9000, cov = "exp2")
fit <- continue(fit, 1000)
plot(fit) # investigate trace plots
fit <- trim(fit, 8000, 2)

# Option 1 - calculate ALC from MCMC iterations
alc <- ALC(fit, xx)

# Option 2 - calculate ALC after predictions
fit <- predict(fit, xx, store_latent = TRUE)
alc <- ALC(fit)

# Visualize fit
plot(fit)
par(new = TRUE) # overlay ALC
plot(xx, alc$value, type = 'l', lty = 2, axes = FALSE, xlab = '', ylab = '')

# Select next design point
x_new <- xx[which.max(alc$value)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

# 3. Three Layer and IMSE ------------------------------------------------------
f <- function(x) {
  i <- which(x <= 0.48)
  x[i] <- 2 * sin(pi * x[i] * 4) + 0.4 * cos(pi * x[i] * 16)
  x[-i] <- 2 * x[-i] - 1
  return(x)
}

# Training data
x <- seq(0, 1, length = 30)
y <- f(x) + rnorm(30, 0, 0.05)

# Testing data
xx <- seq(0, 1, length = 100)
yy <- f(xx)

# Conduct MCMC
fit <- fit_three_layer(x, y, D = 1, nmcmc = 10000, cov = "exp2")
plot(fit) # investigate trace plots
fit <- trim(fit, 8000, 2)

# Option 1 - calculate IMSE from only MCMC iterations
imse <- IMSE(fit, xx)

# Option 2 - calculate IMSE after predictions
fit <- predict(fit, xx, store_latent = TRUE)
imse <- IMSE(fit)

# Visualize fit
plot(fit)
par(new = TRUE) # overlay IMSE
plot(xx, imse$value, col = 2, type = "l", lty = 2, axes = FALSE, xlab = "", ylab = "")

# Select next design point
x_new <- xx[which.min(imse$value)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

---

**ALC**

*Active Learning Cohn for Sequential Design*

**Description**

Acts on a gp, dgp2, or dgp3 object. Current version requires squared exponential covariance (cov = "exp2"). Calculates ALC over the input locations x_new using specified reference grid. If no reference grid is specified, x_new is used as the reference. Optionally utilizes SNOW parallelization. User should select the point with the highest ALC to add to the design.
Usage

```r
ALC(object, x_new, ref, cores)
```

## S3 method for class 'gp'

```r
ALC(object, x_new = NULL, ref = NULL, cores = 1)
```

## S3 method for class 'dgp2'

```r
ALC(object, x_new = NULL, ref = NULL, cores = 1)
```

## S3 method for class 'dgp3'

```r
ALC(object, x_new = NULL, ref = NULL, cores = 1)
```

Arguments

- **object**: object of class `gp`, `dgp2`, or `dgp3`
- **x_new**: matrix of possible input locations, if object has been run through `predict` the previously stored `x_new` is used
- **ref**: optional reference grid for ALC approximation, if `ref = NULL` then `x_new` is used
- **cores**: number of cores to utilize in parallel, by default no parallelization is used

Details

All iterations in the object are used in the calculation, so samples should be burned-in. Thinning the samples using `trim` will speed up computation. This function may be used in two ways:

- **Option 1**: called on an object with only MCMC iterations, in which case `x_new` must be specified
- **Option 2**: called on an object that has been predicted over, in which case the `x_new` from `predict` is used

In Option 2, it is recommended to set `store_latent = TRUE` for `dgp2` and `dgp3` objects so latent mappings do not have to be re-calculated. Through `predict`, the user may specify a mean mapping (`mean_map = TRUE`) or a full sample from the MVN distribution over `w_new` (`mean_map = FALSE`). When the object has not yet been predicted over (Option 1), the mean mapping is used.

SNOW parallelization reduces computation time but requires more memory storage. C code derived from the "laGP" package (Robert B Gramacy and Furong Sun).

Value

- A list with elements:
  - **value**: vector of ALC values, indices correspond to `x_new`
  - **time**: computation time in seconds
References


Examples

# See "deepgp-package" or "fit_two_layer" for an example

```
continue

Acts on a gp, dgp2, or dgp3 object. Continues MCMC sampling of hyperparameters and hidden layers and appends results to existing object.

Usage

continue(object, new_mcmc, verb)

## S3 method for class 'gp'
continue(object, new_mcmc = 1000, verb = TRUE)

## S3 method for class 'dgp2'
continue(object, new_mcmc = 1000, verb = TRUE)

## S3 method for class 'dgp3'
continue(object, new_mcmc = 1000, verb = TRUE)

Arguments

object object from fit_one_layer, fit_two_layer, or fit_three_layer
new_mcmc number of MCMC iterations to conduct and append
verb logical indicating whether to print iteration progress
Details

See `fit_one_layer`, `fit_two_layer`, or `fit_three_layer` for details on MCMC. The resulting object will have `nmcmc` equal to the previous `nmcmc` plus `new_mcmc`. It is recommended to start an MCMC fit then investigate trace plots to assess burn-in. The primary use of this function is to gather more MCMC iterations in order to obtain burned-in samples.

Value

object of the same class with the new iterations appended

Examples

```r
# See "deepgp-package" or "fit_two_layer" for an example
```

---

**fit_one_layer**

*MCMC sampling for one layer GP*

Description

Conducts MCMC sampling of hyperparameters for a one layer GP. Length scale parameter \( \theta \) governs the strength of the correlation and nugget parameter \( g \) governs noise. In Matern covariance, \( v \) governs smoothness.

Usage

```r
fit_one_layer(
  x,
  y,
  nmcmc = 10000,
  verb = TRUE,
  g_0 = 0.01,
  theta_0 = 0.1,
  true_g = NULL,
  settings = list(l = 1, u = 2, alpha = list(g = 1.5, theta = 1.5), beta = list(g = 3.9, theta = 3.9/1.5)),
  cov = c("matern", "exp2"),
  v = 2.5
)
```

Arguments

- `x`: vector or matrix of input locations
- `y`: vector of response values
- `nmcmc`: number of MCMC iterations
- `verb`: logical indicating whether to print iteration progress
fit_one_layer

\[
\begin{align*}
g_0 & \quad \text{initial value for } g \\
\theta_0 & \quad \text{initial value for } \theta \\
true_g & \quad \text{if true nugget is known it may be specified here (set to a small value to make fit deterministic). Note - values that are too small may cause numerical issues in matrix inversions.} \\
\text{settings} & \quad \text{hyperparameters for proposals and priors on } g \text{ and } \theta \\
cov & \quad \text{covariance kernel, either Matern or squared exponential (exp2)} \\
v & \quad \text{Matern smoothness parameter (only used if } \text{cov } = \text{"matern"})
\end{align*}
\]

Details

Utilizes Metropolis Hastings sampling of the length scale and nugget parameters with proposals and priors controlled by settings. Proposals for \( g \) and \( \theta \) follow a uniform sliding window scheme, e.g.

\[
g_{\text{star}} \leftarrow \text{runif}(1, l \times g_t / u, u \times g_t / l),
\]

with defaults \( l = 1 \) and \( u = 2 \) provided in settings. Priors on \( g \) and \( \theta \) follow Gamma distributions with shape parameter (alpha) and rate parameter (beta) provided in settings. These priors are designed for \( x \) scaled to \([0, 1]\) and \( y \) scaled to have mean 0 and variance 1.

The output object of class \( \text{gp} \) is designed for use with \( \text{continue}, \text{trim}, \text{and predict} \).

Value

a list of the S3 class \( \text{gp} \) with elements:

- \( x \): copy of input matrix
- \( y \): copy of response vector
- \text{nmc}: number of MCMC iterations
- \text{settings}: copy of proposal/prior settings
- \text{cov}: copy of covariance kernel setting
- \( v \): copy of Matern smoothness parameter (if \( \text{cov } = \text{"matern"})
- \( g \): vector of MCMC samples for \( g \)
- \( \theta \): vector of MCMC samples for \( \theta \)
- \text{time}: computation time in seconds

References


Examples

# Toy example (runs in less than 5 seconds) --------------------------------
# This example uses a small number of MCMC iterations in order to run quickly
# More iterations are required to get appropriate fits
# Function defaults are recommended (see additional example below)

f <- function(x) {
  if (x <= 0.4) return(-1)
  if (x >= 0.6) return(1)
  if (x > 0.4 & x < 0.6) return(10 * (x - 0.5))
}
x <- seq(0.05, 0.95, length = 7)
y <- sapply(x, f)
x_new <- seq(0, 1, length = 100)

# Fit model and calculate EI
fit <- fit_one_layer(x, y, nmcmc = 500)
fit <- trim(fit, 400)
fit <- predict(fit, x_new, EI = TRUE)

# One Layer and EI ----------------------------------------------------------

f <- function(x) {
  sin(5 * pi * x) / (2 * x) + (x - 1) ^ 4
}

# Training data
x <- seq(0.5, 2, length = 30)
y <- f(x) + rnorm(30, 0, 0.01)

# Testing data
xx <- seq(0.5, 2, length = 100)
yy <- f(xx)

# Standardize inputs and outputs
xx <- (xx - min(x)) / (max(x) - min(x))
x <- (x - min(x)) / (max(x) - min(x))
yy <- (yy - mean(y)) / sd(y)
y <- (y - mean(y)) / sd(y)

# Conduct MCMC
fit <- fit_one_layer(x, y, nmcmc = 10000)
plot(fit)  # investigate trace plots
fit <- trim(fit, 8000, 2)

# Predict and calculate EI
fit <- predict(fit, xx, EI = TRUE)

# Visualize Fit
plot(fit)
par(new = TRUE)  # overlay EI
fit_three_layer

plot(xx, fit$EI, type = 'l', lty = 2, axes = FALSE, xlab = '', ylab = '')

# Select next design point
x_new <- xx[which.max(fit$EI)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

fit_three_layer

MCMC sampling for three layer deep GP

Description

Conducts MCMC sampling of hyperparameters, hidden layer z, and hidden layer w for a three layer deep GP. Separate length scale parameters theta_z, theta_w, and theta_y govern the correlation strength of the inner layer, middle layer, and outer layer respectively. Nugget parameter g governs noise on the outer layer. In Matern covariance, v governs smoothness.

Usage

fit_three_layer(
  x, 
  y,  
  D = ifelse(is.matrix(x), ncol(x), 1), 
  nmcmc = 10000, 
  verb = TRUE,  
  w_0 = suppressWarnings(matrix(x, nrow = length(y), ncol = D)), 
  z_0 = suppressWarnings(matrix(x, nrow = length(y), ncol = D)), 
  g_0 = 0.01,  
  theta_y_0 = 0.1,  
  theta_w_0 = 0.1,  
  theta_z_0 = 0.1, 
  true_g = NULL,  
  settings = list(l = 1, u = 2, alpha = list(g = 1.5, theta_z = 1.5, theta_w = 1.5, theta_y = 1.5), beta = list(g = 3.9, theta_z = 3.9/4, theta_w = 3.9/12, theta_y = 3.9/6)), 
  cov = c("matern", "exp2"),  
  v = 2.5  
)

Arguments

x vector or matrix of input locations
y vector of response values
D integer designating dimension of hidden layers, defaults to dimension of x
```r
nmcmc  number of MCMC iterations
verb   logical indicating whether to print iteration progress
w_0    initial value for hidden layer \( w \) (must be matrix of dimension \( \text{nrow}(x) \) by \( D \) or dimension \( \text{nrow}(x) -1 \) by \( D \)). Defaults to the identity mapping.
z_0    initial value for hidden layer \( z \) (must be matrix of dimension \( \text{nrow}(x) \) by \( D \) or dimension \( \text{nrow}(x) -1 \) by \( D \)). Defaults to the identity mapping.
g_0    initial value for \( g \)
theta_y_0 initial value for \( \theta_y \) (length scale of outer layer)
theta_w_0 initial value for \( \theta_w \) (length scale of middle layer), may be single value or vector of length \( D \)
theta_z_0 initial value for \( \theta_z \) (length scale of inner layer), may be single value or vector of length \( D \)
true_g if true nugget is known it may be specified here (set to a small value to make fit deterministic). Note - values that are too small may cause numerical issues in matrix inversions.
settings hyperparameters for proposals and priors on \( g \), \( \theta_y \), \( \theta_w \), and \( \theta_z \)
cov    covariance kernel, either Matern or squared exponential (\text{exp2})
v      Matern smoothness parameter (only used if cov = "matern")
```

Details

Maps inputs \( x \) through hidden layer \( z \) then hidden layer \( w \) to outputs \( y \). Conducts sampling of the hidden layers using Elliptical Slice sampling. Utilizes Metropolis Hastings sampling of the length scale and nugget parameters with proposals and priors controlled by settings. Proposals for \( g \), \( \theta_y \), \( \theta_w \), and \( \theta_z \) follow a uniform sliding window scheme, e.g.
\[
g_{\text{star}} \leftarrow \text{runif}(1, 1 * g_t / u, u * g_t / l),
\]
with defaults \( l = 1 \) and \( u = 2 \) provided in settings. Priors on \( g \), \( \theta_y \), \( \theta_w \), and \( \theta_z \) follow Gamma distributions with shape parameter (\( \alpha \)) and rate parameter (\( \beta \)) provided in settings. These priors are designed for \( x \) scaled to \([0, 1]\) and \( y \) scaled to have mean 0 and variance 1.

The output object of class \text{dgp3} is designed for use with \text{continue}, \text{trim}, and \text{predict}. If \( z_0 \) and \( w_0 \) are of dimension \( \text{nrow}(x) -1 \) by \( D \), the final rows are predicted using kriging. This is helpful in sequential design when adding a new input location and starting the MCMC at the place where the previous MCMC left off.

Value

a list of the S3 class \text{dgp3} with elements:

- \( x \): copy of input matrix
- \( y \): copy of response vector
- \( nmcmc \): number of MCMC iterations
- \( settings \): copy of proposal/prior settings
- \( cov \): copy of covariance kernel setting
• v: copy of Matern smoothness parameter (if cov = "matern")
• g: vector of MCMC samples for g
• theta_y: vector of MCMC samples for theta_y (length scale of outer layer)
• theta_w: matrix of MCMC samples for theta_w (length scale of middle layer)
• theta_z: matrix of MCMC samples for theta_z (length scale of inner layer)
• w: list of MCMC samples for middle hidden layer w
• z: list of MCMC samples for inner hidden layer z
• time: computation time in seconds

References


Examples

# Toy example (runs in less than 5 seconds) --------------------------------
# This example uses a small number of MCMC iterations in order to run quickly
# More iterations are required to get appropriate fits
# Function defaults are recommended (see additional example below)

f <- function(x) {
  if (x <= 0.4) return(-1)
  if (x >= 0.6) return(1)
  if (x > 0.4 & x < 0.6) return(10 * (x - 0.5))
}
x <- seq(0.05, 0.95, length = 7)
y <- sapply(x, f)
x_new <- seq(0, 1, length = 100)

# Fit model and calculate IMSPE
fit <- fit_three_layer(x, y, nmcmc = 500, cov = "exp2")
fit <- trim(fit, 400)
fit <- predict(fit, x_new, store_latent = TRUE)
imse <- IMSE(fit)

# Three Layer and IMSE -----------------------------------------------------

f <- function(x) {
  i <- which(x <= 0.48)
  x[i] <- 2 * sin(pi * x[i] * 4) + 0.4 * cos(pi * x[i] * 16)
  x[-i] <- 2 * x[-i] - 1
  return(x)
}
# Training data
x <- seq(0, 1, length = 30)
y <- f(x) + rnorm(30, 0, 0.05)

# Testing data
xx <- seq(0, 1, length = 100)
yy <- f(xx)

# Conduct MCMC
fit <- fit_three_layer(x, y, D = 1, nmcmc = 10000, cov = "exp2")
plot(fit) # investigate trace plots
fit <- trim(fit, 8000, 2)

# Option 1 - calculate IMSE from only MCMC iterations
imse <- IMSE(fit, xx)

# Option 2 - calculate IMSE after predictions
fit <- predict(fit, xx)
imse <- IMSE(fit)

# Visualize fit
plot(fit)
par(new = TRUE) # overlay IMSPE
plot(xx, imse$value, type = 'l', lty = 2, axes = FALSE, xlab = '', ylab = '')

# Select next design point
x_new <- xx[which.min(imse$value)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

---

**fit_two_layer**

**MCMC sampling for two layer deep GP**

**Description**

Conducts MCMC sampling of hyperparameters and hidden layer \( w \) for a two layer deep GP. Separate length scale parameters \( \theta_w \) and \( \theta_y \) govern the correlation strength of the hidden layer and outer layer respectively. Nugget parameter \( g \) governs noise on the outer layer. In Matern covariance, \( v \) governs smoothness.

**Usage**

```r
fit_two_layer(
  x,
  y,
  D = ifelse(is.matrix(x), ncol(x), 1),
  nmcmc = 10000,
)```
verb = TRUE,
w_0 = suppressWarnings(matrix(x, nrow = length(y), ncol = D)),
g_0 = 0.01,
theta_y_0 = 0.1,
theta_w_0 = 0.1,
true_g = NULL,
settings = list(l = 1, u = 2, alpha = list(g = 1.5, theta_w = 1.5, theta_y = 1.5),
    beta = list(g = 3.9, theta_w = 3.9/4, theta_y = 3.9/6)),
cov = c("matern", "exp2"),
v = 2.5
)

Arguments

- **x**: vector or matrix of input locations
- **y**: vector of response values
- **D**: integer designating dimension of hidden layer, defaults to dimension of x
- **nmcmc**: number of MCMC iterations
- **verb**: logical indicating whether to print iteration progress
- **w_0**: initial value for hidden layer w (must be matrix of dimension nrow(x) by D or dimension nrow(x) -1 by D). Defaults to the identity mapping.
- **g_0**: initial value for g
- **theta_y_0**: initial value for theta_y (length scale of outer layer)
- **theta_w_0**: initial value for theta_w (length scale of inner layer), may be single value or vector of length D
- **true_g**: if true nugget is known it may be specified here (set to a small value to make fit deterministic). Note - values that are too small may cause numerical issues in matrix inversions.
- **settings**: hyperparameters for proposals and priors on g, theta_y, and theta_w
- **cov**: covariance kernel, either Matern or squared exponential (exp2)
- **v**: Matern smoothness parameter (only used if cov = "matern")

Details

Maps inputs x through hidden layer w to outputs y. Conducts sampling of the hidden layer using Elliptical Slice sampling. Utilizes Metropolis Hastings sampling of the length scale and nugget parameters with proposals and priors controlled by settings. Proposals for g, theta_y, and theta_w follow a uniform sliding window scheme, e.g.

\[ g_{\text{star}} \leftarrow \text{runif}(1, l \times g_t / u, u \times g_t / l) \]

with defaults l = 1 and u = 2 provided in settings. Priors on g and theta follow Gamma distributions with shape parameter (alpha) and rate parameter (beta) provided in settings. These priors are designed for x scaled to [0, 1] and y scaled to have mean 0 and variance 1.

The output object of class dgp2 is designed for use with continue, trim, and predict. If w_0 is of dimension nrow(x) -1 by D, the final row is predicted using kriging. This is helpful in sequential design when adding a new input location and starting the MCMC at the place where the previous MCMC left off.
Value

a list of the S3 class dgp2 with elements:

- x: copy of input matrix
- y: copy of response vector
- nmcmc: number of MCMC iterations
- settings: copy of proposal/prior settings
- cov: copy of covariance kernel setting
- v: copy of Matern smoothness parameter (if cov = "matern")
- g: vector of MCMC samples for g
- theta_y: vector of MCMC samples for theta_y (length scale of outer layer)
- theta_w: matrix of MCMC samples for theta_w (length scale of inner layer)
- w: list of MCMC samples for hidden layer w
- time: computation time in seconds

References


Examples

# Toy example (runs in less than 5 seconds) --------------------------------
# This example uses a small number of MCMC iterations in order to run quickly
# More iterations are required to get appropriate fits
# Function defaults are recommended (see additional example below)

f <- function(x) {
  if (x <= 0.4) return(-1)
  if (x >= 0.6) return(1)
  if (x > 0.4 & x < 0.6) return(10 * (x - 0.5))
}
x <- seq(0.05, 0.95, length = 7)
y <- sapply(x, f)
x_new <- seq(0, 1, length = 100)

# Fit model and calculate ALC
fit <- fit_two_layer(x, y, nmcmc = 500, cov = "exp2")
fit <- trim(fit, 400)
fit <- predict(fit, x_new, store_latent = TRUE)
alc <- ALC(fit)

# Two Layer and ALC --------------------------------------------------------
f <- function(x) {
    exp(-10 * x) * (cos(10 * pi * x - 1) + sin(10 * pi * x - 1)) * 5 - 0.2
}

# Training data
x <- seq(0, 1, length = 30)
y <- f(x) + rnorm(30, 0, 0.05)

# Testing data
xx <- seq(0, 1, length = 100)
yy <- f(xx)

# Conduct MCMC
fit <- fit_two_layer(x, y, D = 1, nmcmc = 9000, cov = "exp2")
fit <- continue(fit, 1000)
plot(fit) # investigate trace plots
fit <- trim(fit, 8000, 2)

# Option 1 - calculate ALC from MCMC iterations
alc <- ALC(fit, xx)

# Option 2 - calculate ALC after predictions
fit <- predict(fit, xx, store_latent = TRUE)
alc <- ALC(fit)

# Visualize fit
plot(fit)
par(new = TRUE) # overlay ALC
plot(xx, alc$value, type = "l", lty = 2, axes = FALSE, xlab = "", ylab = "")

# Select next design point
x_new <- xx[which.max(alc$value)]

# Evaluate fit
rmse(yy, fit$mean) # lower is better

---

**IMSE**  
*Integrated Mean-Squared (prediction) Error for Sequential Design*

**Description**

Acts on a gp, dgp2, or dgp3 object. Current version requires squared exponential covariance (cov = "exp2"). Calculates IMSE over the input locations x_new. Optionally utilizes SNOW parallelization. User should select the point with the lowest IMSE to add to the design.
### Usage

```
IMSE(object, x_new, cores)
```

## S3 method for class 'gp'
IMSE(object, x_new = NULL, cores = 1)

## S3 method for class 'dgp2'
IMSE(object, x_new = NULL, cores = 1)

## S3 method for class 'dgp3'
IMSE(object, x_new = NULL, cores = 1)

### Arguments

- **object**: object of class `gp`, `dgp2`, or `dgp3`
- **x_new**: matrix of possible input locations, if object has been run through `predict` the previously stored `x_new` is used
- **cores**: number of cores to utilize in parallel, by default no parallelization is used

### Details

All iterations in the object are used in the calculation, so samples should be burned-in. Thinning the samples using `trim` will speed up computation. This function may be used in two ways:

- **Option 1**: called on an object with only MCMC iterations, in which case `x_new` must be specified
- **Option 2**: called on an object that has been predicted over, in which case the `x_new` from `predict` is used

In Option 2, it is recommended to set `store_latent = TRUE` for `dgp2` and `dgp3` objects so latent mappings do not have to be re-calculated. Through `predict`, the user may specify a mean mapping (`mean_map = TRUE`) or a full sample from the MVN distribution over `w_new` (`mean_map = FALSE`). When the object has not yet been predicted over (Option 1), the mean mapping is used.

SNOW parallelization reduces computation time but requires more memory storage.

### Value

- list with elements:
  - **value**: vector of IMSE values, indices correspond to `x_new`
  - **time**: computation time in seconds

### References


plot

Plots object from deepgp package

Description

Acts on a gp, dgp2, or dgp3 object. Generates trace plots for length scale and nugget hyperparameters. Generates plots of hidden layers for one-dimensional inputs. Generates plots of the posterior mean and estimated 95% prediction intervals for one-dimensional inputs; generates heat maps of the posterior mean and point-wise variance for two-dimensional inputs.

Usage

```r
## S3 method for class 'gp'
plot(x, trace = NULL, predict = NULL, ...)

## S3 method for class 'dgp2'
plot(x, trace = NULL, hidden = NULL, predict = NULL, ...)

## S3 method for class 'dgp3'
plot(x, trace = NULL, hidden = NULL, predict = NULL, ...)
```

Arguments

- `x` object of class gp, dgp2, or dgp3
- `trace` logical indicating whether to generate trace plots
- `predict` logical indicating whether to generate posterior predictive plot
- `...` N/A
- `hidden` logical indicating whether to generate plots of hidden layers (dgp2 or dgp3 only)

Details

Trace plots are useful in assessing burn-in. Hidden layer plots are colored on a gradient - red lines represent earlier iterations and yellow lines represent later iterations - to help assess burn-in of the hidden layers. These plots are meant to help in model fitting and visualization.

Examples

```r
# See "deepgp-package", "fit_one_layer", "fit_two_layer", or "fit_three_layer" # for an example
```
**predict**

*Predict posterior mean and variance/covariance*

**Description**

Acts on a gp, dgp2, or dgp3 object. Calculates posterior mean and variance/covariance over specified input locations. Optionally calculates expected improvement (EI) over candidate inputs. Optionally utilizes SNOW parallelization.

**Usage**

```r
## S3 method for class 'gp'
predict(object, x_new, lite = TRUE, EI = FALSE, cores = 1, ...)
## S3 method for class 'dgp2'
predict(
  object,
  x_new,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  EI = FALSE,
  cores = 1,
  ...
)
## S3 method for class 'dgp3'
predict(
  object,
  x_new,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  EI = FALSE,
  cores = 1,
  ...
)
```

**Arguments**

- **object**: object from fit_one_layer, fit_two_layer, or fit_three_layer with burn-in already removed
- **x_new**: matrix of predictive input locations
- **lite**: logical indicating whether to calculate only point-wise variances (lite = TRUE) or full covariance (lite = FALSE)
EI logical indicating whether to calculate expected improvement (for minimizing the response)
cores number of cores to utilize in parallel, by default no parallelization is used
... N/A
store_latent logical indicating whether to store and return mapped values of latent layers (dgp2 or dgp3 only)
mean_map logical indicating whether to map hidden layers using conditional mean (mean_map = TRUE) or using a random sample from the full MVN distribution (dgp2 or dgp3 only)

Details

All iterations in the object are used for prediction, so samples should be burned-in. Thinning the samples using `trim` will speed up computation. Posterior moments are calculated using conditional expectation and variance. As a default, only point-wise variance is calculated. Full covariance may be calculated using `lite = FALSE`.

Expected improvement is calculated with the goal of minimizing the response. See Chapter 7 of Gramacy (2020) for details.

SNOW parallelization reduces computation time but requires significantly more memory storage. Use `cores = 1` if memory is limited.

Value

object of the same class with the following additional elements:

- `x_new`: copy of predictive input locations
- `tau2`: vector of tau2 estimates (governing the magnitude of the covariance)
- `mean`: predicted posterior mean, indices correspond to `x_new` location
- `s2`: predicted point-wise variances, indices correspond to `x_new` location (only returned when `lite = TRUE`)
- `s2_smooth`: predicted point-wise variances with g removed, indices correspond to `x_new` location (only returned when `lite = TRUE`)
- `Sigma`: predicted posterior covariance, indices correspond to `x_new` location (only returned when `lite = FALSE`)
- `Sigma_smooth`: predicted posterior covariance with g removed from the diagonal (only returned when `lite = FALSE`)
- `EI`: vector of expected improvement values, indices correspond to `x_new` location (only returned when `EI = TRUE`)
- `w_new`: list of hidden layer mappings (only returned when `store_latent = TRUE`), list index corresponds to iteration and row index corresponds to `x_new` location (dgp2 and dgp3 only)
- `z_new`: list of hidden layer mappings (only returned when `store_latent = TRUE`), list index corresponds to iteration and row index corresponds to `x_new` location (dgp3 only)

Computation time is added to the computation time of the existing object.
References


Examples

# See "deepgp-package", "fit_one_layer", "fit_two_layer", or "fit_three_layer"
# for an example

rmse  
Calculates RMSE

Description

Calculates root mean square error (lower RMSE indicate better fits).

Usage

rmse(y, mu)

Arguments

y  response vector
mu  predicted mean

Examples

# See "deepgp-package", "fit_one_layer", "fit_two_layer", or "fit_three_layer"
# for an example

score  
Calculates score

Description

Calculates score, proportional to the multivariate normal log likelihood. Higher scores indicate better fits. Only applicable to noisy data. Requires full covariance matrix (e.g. predict with lite = FALSE).

Usage

score(y, mu, sigma)
**Arguments**

- `y`: response vector
- `mu`: predicted mean
- `sigma`: predicted covariance

**References**


---

**sq_dist**

*Calculates squared pairwise distances*

---

**Description**

Calculates squared pairwise euclidean distances using C.

**Usage**

```r
sq_dist(X1, X2 = NULL)
```

**Arguments**

- `X1`: matrix of input locations
- `X2`: matrix of second input locations (if NULL, distance is calculated between `X1` and itself)

**Details**

C code derived from the "laGP" package (Robert B Gramacy and Furong Sun).

**Value**

symmetric matrix of squared euclidean distances

**References**


**Examples**

```r
x <- seq(0, 1, length = 10)
d2 <- sq_dist(x)
```
Trim/Thin MCMC iterations

Description

Acts on a gp, dgp2, or dgp3 object. Removes the specified number of MCMC iterations (starting at the first iteration). After these samples are removed, the remaining samples may be thinned.

Usage

trim(object, burn, thin)

## S3 method for class 'gp'
trim(object, burn, thin = 1)

## S3 method for class 'dgp2'
trim(object, burn, thin = 1)

## S3 method for class 'dgp3'
trim(object, burn, thin = 1)

Arguments

object object from fit_one_layer, fit_two_layer, or fit_three_layer
burn integer specifying number of iterations to cut off as burn-in
thin integer specifying amount of thinning (thin = 1 keeps all iterations, thin = 2 keeps every other iteration, thin = 10 keeps every tenth iteration, etc.)

Details

The resulting object will have nmcmc equal to the previous nmcmc minus burn divided by thin. It is recommended to start an MCMC fit then investigate trace plots to assess burn-in. Once burn-in has been achieved, use this function to remove the starting iterations. Thinning reduces the size of the resulting object while accounting for the high correlation between consecutive iterations.

Value

object of the same class with the selected iterations removed

Examples

# See "deepgp-package", "fit_one_layer", "fit_two_layer", or "fit_three_layer"
# for an example
Index

ALC, 2, 5
continue, 2, 7
deepegp-package, 2
fit_one_layer, 2, 8
fit_three_layer, 2, 11
fit_two_layer, 2, 14
IMSE, 2, 17
plot, 2, 19
predict, 2, 20
rmse, 22
score, 22
sq_dist, 23
trim, 2, 24