Package 'deepgp'

April 8, 2022

Type Package

Title Deep Gaussian Processes using MCMC

Version 1.0.0

Date 2022-4-04

Author Annie Sauer <anniees@vt.edu>

Maintainer Annie Sauer <anniees@vt.edu>

Depends R (>= 3.6)

Description Performs posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2020) <arXiv:2012.08015>. Models are trained through MCMC including elliptical slice sampling of latent Gaussian layers and Metropolis-Hastings sampling of kernel hyperparameters. Vecchia-approximation for faster computation is implemented following Sauer, Cooper, and Gramacy (2022) <arXiv:2204.02904>. Downstream tasks include sequential design through active learning Cohn/integrated mean squared error (ALC/IMSE; Sauer, Gramacy, and Higdon, 2020) and optimization through expected improvement (EI; Gramacy, Sauer, and Wycoff, 2021 <arXiv:2112.07457>). 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. 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, GpGp, Matrix, Rcpp, mvtnorm, FNN

LinkingTo Rcpp, RcppArmadillo,

Suggests akima, knitr

RoxygenNote 7.1.2

Repository CRAN

Date/Publication 2022-04-08 14:12:32 UTC

R topics documented:

---

deepgp-package *Package deepgp*

Description

Performs posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2020) <arXiv:2012.08015>. Models are trained through MCMC including elliptical slice sampling of latent Gaussian layers and Metropolis-Hastings sampling of kernel hyperparameters. Vecchia-approximation for faster computation is implemented following Sauer, Cooper, and Gramacy (2022) <arXiv:2204.02904>. Downstream tasks include sequential design through active learning Cohn/integrated mean squared error (ALC/IMSE; Sauer, Gramacy, and Higdon, 2020) and optimization through expected improvement (EI; Gramacy, Sauer, and Wycoff, 2021 <arXiv:2112.07457>). 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. 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

Sauer, A, RB Gramacy, and D Higdon. 2021. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics*, (just-accepted), 1-39.

Sauer, A, A Cooper, and RB Gramacy. 2022. "Vecchia-approximated Deep Gaussian Processes for Computer Experiments." *pre-print on arXiv:2204.02904*

Katzfuss, M, J Guinness, W Gong, and D Zilber. 2020. "Vecchia aproximations of Gaussian-process predictions." *Journal of Agricultural, Biological, and Environmental Statistics 25*, 383-414.

Binois, M, J Huang, RB Gramacy, and M Ludkovski. 2019. Replication or Exploration? Sequential Design for Stochastic Simulation Experiments. *Technometrics 61*, 7-23. Taylor & Francis. doi:10.1080/00401706.2018.1469433.

Gramacy, RB. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman Hall, 2020.

Jones, DR, M Schonlau, and WJ Welch. 1998. "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization 13*, 455-492. doi:10.1023/A:1008306431147.

Murray, I, RP Adams, and D MacKay. 2010. "Elliptical slice sampling." *Journal of Machine Learning Research 9*, 541-548.

Seo, S, M Wallat, T Graepel, and K Obermayer. 2000. Gaussian Process Regression: Active Data Selection and Test Point Rejection. In Mustererkennung 2000, 27-34. New York, NY: Springer Verlag.

Examples

```
# See "fit_one_layer", "fit_two_layer", "fit_three_layer",
# "ALC", or "IMSE" for examples
# Examples of real-world implementations are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/
```

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

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

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

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

## S3 method for class 'dgp3'
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

Not yet implemented for Vecchia-approximated fits.

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

list with elements:

- `value`: vector of ALC values, indices correspond to `x_new`
- `time`: computation time in seconds

References

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear;* arXiv:2012.08015.

Seo, S, M Wallat, T Graepel, and K Obermayer. 2000. Gaussian Process Regression: Active Data Selection and Test Point Rejection. In Mustererkennung 2000, 2734. New York, NY: SpringerVerlag.

Gramacy, RB and F Sun. (2016). laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R. *Journal of Statistical Software 72* (1), 1-46. doi:10.18637/jss.v072.i01

Examples

```
# ------------------------------------------------------
# Example 1: toy step function, runs in less than 5 seconds
# ------------------------------------------------------

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 = 100, cov = "exp2")
fit <- trim(fit, 50)
fit <- predict(fit, x_new, cores = 1, store_latent = TRUE)
alc <- ALC(fit)


# ------------------------------------------------------
# Example 2: damped sine wave
# ------------------------------------------------------

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)

plot(xx, yy, type = "l")
points(x, y, col = 2)

# Conduct MCMC (can replace fit_two_layer with fit_one_layer/fit_three_layer)
fit <- fit_two_layer(x, y, D = 1, nmcmc = 2000, cov = "exp2")
plot(fit)
fit <- trim(fit, 1000, 2)

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

# Option 2 - calculate ALC after predictions
fit <- predict(fit, xx, cores = 1, 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)]
```

`continue` *Continues MCMC sampling*

Description

Acts on a `gp`, `gpvec`, `dgp2`, `dgp2vec`, `dgp3`, or `dgp3vec` object. Continues MCMC sampling of hyperparameters and hidden layers using settings from the original object. Appends new samples to existing samples. When `vecchia = TRUE`, this function provides the option to update Vecchia ordering/conditioning sets based on latent layer warpings through the specification of `re_approx = TRUE`.

Usage

```
continue(object, new_mcmc, verb, re_approx, ...)
```

```
## 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, ...)

## S3 method for class 'gpvec'
continue(object, new_mcmc = 1000, verb = TRUE, re_approx = FALSE, ...)

## S3 method for class 'dgp2vec'
continue(object, new_mcmc = 1000, verb = TRUE, re_approx = FALSE, ...)

## S3 method for class 'dgp3vec'
continue(object, new_mcmc = 1000, verb = TRUE, re_approx = FALSE, ...)
```

Arguments

object	object from fit_one_layer, fit_two_layer, or fit_three_layer
new_mcmc	number of new MCMC iterations to conduct and append
verb	logical indicating whether to print iteration progress
re_approx	logical indicating whether to re-randomize the ordering and update Vecchia nearest-neighbor conditioning sets (only for fits with vecchia = TRUE)
...	N/A

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.

Specifying re_approx = TRUE updates random orderings and nearest-neighbor conditioning sets (only for vecchia = TRUE fits). In one-layer, there is no latent warping but the Vecchia approximation is still re-randomized and nearest-neighbors are adjusted accordingly. In two- and three-layers, the latest samples of hidden layers are used to update nearest-neighbors. If you update the Vecchia approximation, you should later remove previous samples (updating the approximation effectively starts a new chain). When re_approx = FALSE the previous orderings and conditioning sets are used (maintaining the continuity of the previous chain).

Value

object of the same class with the new iterations appended

Examples

```
# See "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

```
fit_one_layer(
  x,
  y,
  nmcmc = 10000,
  verb = TRUE,
  g_0 = 0.01,
  theta_0 = 0.1,
  true_g = NULL,
  settings = NULL,
  cov = c("matern", "exp2"),
  v = 2.5,
  vecchia = FALSE,
  m = min(25, length(y) - 1)
)
```

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
`g_0`	initial value for `g`
`theta_0`	initial value for `theta`
`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 (see details)
`cov`	covariance kernel, either Matern or squared exponential (`"exp2"`)
`v`	Matern smoothness parameter (only used if `cov = "matern"`)
`vecchia`	logical indicating whether to use Vecchia approximation
`m`	size of Vecchia conditioning sets (only used if `vecchia = TRUE`)

Details

Utilizes Metropolis Hastings sampling of the length scale and nugget parameters with proposals and priors controlled by `settings`. When `true_g` is set to a specific value, the nugget is not estimated. When `vecchia = TRUE`, all calculations leverage the Vecchia approximation with specified conditioning set size `m`. Vecchia approximation is only implemented for `cov = "matern"`.

Proposals for `g` and `theta` follow a uniform sliding window scheme, e.g.

`g_star <-runif(1,l * g_t / u,u * g_t / l)`,

with defaults `l = 1` and `u = 2` provided in `settings`. To adjust these, set `settings = list(l = new_l,u = new_u)`. Priors on `g` and `theta` follow Gamma distributions with shape parameters (`alpha`) and rate parameters (`beta`) controlled within the `settings` list object. Defaults are

- `settings$alpha$g <-1.5`
- `settings$beta$g <-3.9`
- `settings$alpha$theta <-1.5`
- `settings$beta$theta <-3.9 / 1.5`

These priors are designed for `x` scaled to [0, 1] and `y` scaled to have mean 0 and variance 1. These may be adjusted using the `settings` input.

The output object of class `gp` is designed for use with `continue`, `trim`, and `predict`.

Value

a list of the S3 class `gp` or `gpvec` with elements:

- `x`: copy of input matrix
- `y`: copy of response vector
- `nmcmc`: number of MCMC iterations
- `settings`: copy of proposal/prior settings
- `v`: copy of Matern smoothness parameter (`v = 999` indicates `cov = "exp2"`)
- `g`: vector of MCMC samples for `g`
- `theta`: vector of MCMC samples for `theta`
- `tau2`: vector of MLE estimates for `tau2` (scale parameter)
- `time`: computation time in seconds

References

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear;* arXiv:2012.08015.

Sauer, A, A Cooper, and RB Gramacy. 2022. "Vecchia-approximated Deep Gaussian Processes for Computer Experiments." *pre-print on arXiv:2204.02904*

Gramacy, RB. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman Hall, 2020.

Examples

```
# Examples of real-world implementations are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# G function (https://www.sfu.ca/~ssurjano/gfunc.html)
f <- function(xx, a = (c(1:length(xx)) - 1) / 2) {
    new1 <- abs(4 * xx - 2) + a
    new2 <- 1 + a
    prod <- prod(new1 / new2)
    return((prod - 1) / 0.86)
}

# Training data
d <- 1
n <- 20
x <- matrix(runif(n * d), ncol = d)
y <- apply(x, 1, f)

# Testing data
n_test <- 100
xx <- matrix(runif(n_test * d), ncol = d)
yy <- apply(xx, 1, f)

plot(xx[order(xx)], yy[order(xx)], type = "l")
points(x, y, col = 2)

# Example 1: full model (nugget fixed)
fit <- fit_one_layer(x, y, nmcmc = 2000, true_g = 1e-6)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)

# Example 2: full model (nugget estimated, EI calculated)
fit <- fit_one_layer(x, y, nmcmc = 2000)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1, EI = TRUE)
plot(fit)
par(new = TRUE) # overlay EI
plot(xx[order(xx)], fit$EI[order(xx)], type = 'l', lty = 2,
      axes = FALSE, xlab = '', ylab = '')

# Example 3: Vecchia approximated model
fit <- fit_one_layer(x, y, nmcmc = 2000, vecchia = TRUE, m = 10)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)
```

`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 = NULL,
  z_0 = NULL,
  g_0 = 0.01,
  theta_y_0 = 0.1,
  theta_w_0 = 0.1,
  theta_z_0 = 0.1,
  true_g = NULL,
  settings = NULL,
  cov = c("matern", "exp2"),
  v = 2.5,
  vecchia = FALSE,
  m = min(25, length(y) - 1)
)
```

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`
`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.
`z_0`	initial value for hidden layer `z` (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 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 (see details)
cov	covariance kernel, either Matern or squared exponential ("exp2")
v	Matern smoothness parameter (only used if cov = "matern")
vecchia	logical indicating whether to use Vecchia approximation
m	size of Vecchia conditioning sets (only used if vecchia = TRUE)

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. When true_g is set to a specific value, the nugget is not estimated. When vecchia = TRUE, all calculations leverage the Vecchia approximation with specified conditioning set size m. Vecchia approximation is only implemented for cov = "matern".

Proposals for g, theta_y, theta_w, and theta_z follow a uniform sliding window scheme, e.g.

g_star <-runif(1,l * g_t / u,u * g_t / l),

with defaults l = 1 and u = 2 provided in settings. To adjust these, set settings = list(l = new_l,u = new_u). Priors on g, theta_y, theta_w, and theta_z follow Gamma distributions with shape parameters (alpha) and rate parameters (beta) controlled within the settings list object. Defaults are

- settings$alpha$g <-1.5
- settings$beta$g <-3.9
- settings$alpha$theta_z <-1.5
- settings$beta$theta_z <-3.9 / 4
- settings$alpha$theta_w <-1.5
- settings$beta$theta_w <-3.9 / 12
- settings$alpha$theta_y <-1.5
- settings$beta$theta_y <-3.9 / 6

These priors are designed for x scaled to [0, 1] and y scaled to have mean 0 and variance 1. These may be adjusted using the settings input.

When w_0 = NULL and/or z_0 = NULL, the hidden layers are initialized at x (i.e. the identity mapping). The default prior mean of the hidden layer is zero, but may be adjusted to x using settings = list(w_prior_mean = x,z_prior_mean = x). If w_0 and/or z_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.

The output object of class dgp3 or dgp3vec is designed for use with continue, trim, and predict.

Value

a list of the S3 class `dgp3` or `dgp3vec` with elements:

- `x`: copy of input matrix
- `y`: copy of response vector
- `nmcmc`: number of MCMC iterations
- `settings`: copy of proposal/prior settings
- `v`: copy of Matern smoothness parameter (`v = 999` indicates `cov = "exp2"`)
- `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)
- `tau2`: vector of MLE estimates for `tau2` (scale parameter of outer 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

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear;* arXiv:2012.08015.

Sauer, A, A Cooper, and RB Gramacy. 2022. "Vecchia-approximated Deep Gaussian Processes for Computer Experiments." *pre-print on arXiv:2204.02904*

Murray, I, RP Adams, and D MacKay. 2010. "Elliptical slice sampling." *Journal of Machine Learning Research 9*, 541-548.

Examples

```
# Examples of real-world implementations are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# G function (https://www.sfu.ca/~ssurjano/gfunc.html)
f <- function(xx, a = (c(1:length(xx)) - 1) / 2) {
    new1 <- abs(4 * xx - 2) + a
    new2 <- 1 + a
    prod <- prod(new1 / new2)
    return((prod - 1) / 0.86)
}

# Training data
d <- 2
n <- 30
x <- matrix(runif(n * d), ncol = d)
y <- apply(x, 1, f)
```

```
# Testing data
n_test <- 100
xx <- matrix(runif(n_test * d), ncol = d)
yy <- apply(xx, 1, f)

i <- akima::interp(xx[, 1], xx[, 2], yy)
image(i, col = heat.colors(128))
contour(i, add = TRUE)
points(x)

# Example 1: full model (nugget estimated)
fit <- fit_three_layer(x, y, nmcmc = 2000)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)

# Example 2: Vecchia approximated model (nugget fixed)
# (Vecchia approximation is faster for larger data sizes)
fit <- fit_three_layer(x, y, nmcmc = 2000, vecchia = TRUE,
                       m = 10, true_g = 1e-6)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)
```

`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

```
fit_two_layer(
  x,
  y,
  D = ifelse(is.matrix(x), ncol(x), 1),
  nmcmc = 10000,
  verb = TRUE,
  w_0 = NULL,
  g_0 = 0.01,
```

```
  theta_y_0 = 0.1,
  theta_w_0 = 0.1,
  true_g = NULL,
  settings = NULL,
  cov = c("matern", "exp2"),
  v = 2.5,
  vecchia = FALSE,
  m = min(25, length(y) - 1)
)
```

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 (see details)
cov	covariance kernel, either Matern or squared exponential ("exp2")
v	Matern smoothness parameter (only used if cov = "matern")
vecchia	logical indicating whether to use Vecchia approximation
m	size of Vecchia conditioning sets (only used if vecchia = TRUE)

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. When true_g is set to a specific value, the nugget is not estimated. When vecchia = TRUE, all calculations leverage the Vecchia approximation with specified conditioning set size m. Vecchia approximation is only implemented for cov = "matern".

Proposals for g, theta_y, and theta_w follow a uniform sliding window scheme, e.g.

g_star <-runif(1,l * g_t / u,u * g_t / l),

with defaults l = 1 and u = 2 provided in settings. To adjust these, set settings = list(l = new_l,u = new_u). Priors on g, theta_y, and theta_w follow Gamma distributions with shape

parameters (alpha) and rate parameters (beta) controlled within the settings list object. Defaults are

- settings$alpha$g <-1.5
- settings$beta$g <-3.9
- settings$alpha$theta_w <-1.5
- settings$beta$theta_w <-3.9 / 4
- settings$alpha$theta_y <-1.5
- settings$beta$theta_y <-3.9 / 6

These priors are designed for x scaled to [0, 1] and y scaled to have mean 0 and variance 1. These may be adjusted using the settings input.

When w_0 = NULL, the hidden layer is initialized at x (i.e. the identity mapping). The default prior mean of the hidden layer is zero, but may be adjusted to x using settings = list(w_prior_mean = x). 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.

The output object of class dgp2 or dgp2vec is designed for use with continue, trim, and predict.

Value

a list of the S3 class dgp2 or dgp2vec with elements:

- x: copy of input matrix
- y: copy of response vector
- nmcmc: number of MCMC iterations
- settings: copy of proposal/prior settings
- v: copy of Matern smoothness parameter (v = 999 indicates cov = "exp2")
- 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)
- tau2: vector of MLE estimates for tau2 (scale parameter of outer layer)
- w: list of MCMC samples for hidden layer w
- time: computation time in seconds

References

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear;* arXiv:2012.08015.

Sauer, A, A Cooper, and RB Gramacy. 2022. "Vecchia-approximated Deep Gaussian Processes for Computer Experiments." *pre-print on arXiv:2204.02904*

Murray, I, RP Adams, and D MacKay. 2010. "Elliptical slice sampling." *Journal of Machine Learning Research 9*, 541-548.

Examples

```
# Examples of real-world implementations are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# G function (https://www.sfu.ca/~ssurjano/gfunc.html)
f <- function(xx, a = (c(1:length(xx)) - 1) / 2) {
    new1 <- abs(4 * xx - 2) + a
    new2 <- 1 + a
    prod <- prod(new1 / new2)
    return((prod - 1) / 0.86)
}

# Training data
d <- 1
n <- 20
x <- matrix(runif(n * d), ncol = d)
y <- apply(x, 1, f)

# Testing data
n_test <- 100
xx <- matrix(runif(n_test * d), ncol = d)
yy <- apply(xx, 1, f)

plot(xx[order(xx)], yy[order(xx)], type = "l")
points(x, y, col = 2)

# Example 1: full model (nugget estimated, using continue)
fit <- fit_two_layer(x, y, nmcmc = 1000)
plot(fit)
fit <- continue(fit, 1000)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit, hidden = TRUE)

# Example 2: Vecchia approximated model
# (Vecchia approximation is faster for larger data sizes)
fit <- fit_two_layer(x, y, nmcmc = 2000, vecchia = TRUE, m = 10)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit, hidden = TRUE)

# Example 3: Vecchia approximated model (re-approximated after burn-in)
fit <- fit_two_layer(x, y, nmcmc = 1000, vecchia = TRUE, m = 10)
fit <- continue(fit, 1000, re_approx = TRUE)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit, hidden = TRUE)
```

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

Not yet implemented for Vecchia-approximated fits.

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

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear:* arXiv:2012.08015.

Binois, M, J Huang, RB Gramacy, and M Ludkovski. 2019. "Replication or Exploration? Sequential Design for Stochastic Simulation Experiments." *Technometrics 61*, 7-23. Taylor & Francis. doi:10.1080/00401706.2018.1469433.

Examples

```
# ------------------------------------------------------
# Example 1: toy step function, runs in less than 5 seconds
# ------------------------------------------------------

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 IMSE
fit <- fit_one_layer(x, y, nmcmc = 100, cov = "exp2")
fit <- trim(fit, 50)
fit <- predict(fit, x_new, cores = 1, store_latent = TRUE)
imse <- IMSE(fit)


# ------------------------------------------------------
# Example 2: Higdon function
# ------------------------------------------------------

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)

plot(xx, yy, type = "l")
points(x, y, col = 2)

# Conduct MCMC (can replace fit_three_layer with fit_one_layer/fit_two_layer)
fit <- fit_three_layer(x, y, D = 1, nmcmc = 2000, cov = "exp2")
plot(fit)
fit <- trim(fit, 1000, 2)

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

# Option 2 - calculate IMSE after predictions
fit <- predict(fit, xx, cores = 1, 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)]
```

`plot` *Plots object from* deepgp *package*

Description

Acts on a gp, gpvec, dgp2, dgp2vec, dgp3, or dgp3vec 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

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

## S3 method for class 'gpvec'
```

```
plot(x, trace = NULL, predict = NULL, ...)

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

## S3 method for class 'dgp2vec'
plot(x, trace = NULL, hidden = NULL, predict = NULL, ...)

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

## S3 method for class 'dgp3vec'
plot(x, trace = NULL, hidden = NULL, predict = NULL, ...)
```

Arguments

x	object of class gp, gpvec, dgp2, dgp2vec, dgp3, or dgp3vec
trace	logical indicating whether to generate trace plots (default is TRUE if the object has not been through predict)
predict	logical indicating whether to generate posterior predictive plot (default is TRUE if the object has been through predict)
...	N/A
hidden	logical indicating whether to generate plots of hidden layers (two or three layer only, default is FALSE)

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

```
# See "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

```
## S3 method for class 'gp'
predict(object, x_new, lite = TRUE, EI = FALSE, cores = detectCores() - 1, ...)

## S3 method for class 'dgp2'
predict(
  object,
  x_new,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  EI = FALSE,
  cores = detectCores() - 1,
  ...
)

## S3 method for class 'dgp3'
predict(
  object,
  x_new,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  EI = FALSE,
  cores = detectCores() - 1,
  ...
)

## S3 method for class 'gpvec'
predict(
  object,
  x_new,
  m = object$m,
  lite = TRUE,
  cores = detectCores() - 1,
  ...
)

## S3 method for class 'dgp2vec'
predict(
  object,
  x_new,
  m = object$m,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  cores = detectCores() - 1,
  ...
```

```
)

## S3 method for class 'dgp3vec'
predict(
  object,
  x_new,
  m = object$m,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  cores = detectCores() - 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, defaults to available cores minus one
...	N/A
store_latent	logical indicating whether to store and return mapped values of latent layers (two or three layer models 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 (two or three layer models only), mean_map = FALSE is not yet implemented for fits with vecchia = TRUE
m	size of Vecchia conditioning sets (only for fits with vecchia = TRUE), defaults to the m used for MCMC

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 more memory storage.

Value

object of the same class with the following additional elements:

- x_new: copy of predictive input locations
- mean: predicted posterior mean, indices correspond to x_new locations
- s2: predicted point-wise variances, indices correspond to x_new locations (only returned when lite = TRUE)
- s2_smooth: predicted point-wise variances with g removed, indices correspond to x_new locations (only returned when lite = TRUE)
- Sigma: predicted posterior covariance, indices correspond to x_new locations (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 locations (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 (two or three layer models 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 (three layer models only)

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

References

Sauer, A, RB Gramacy, and D Higdon. 2020. "Active Learning for Deep Gaussian Process Surrogates." *Technometrics, to appear;* arXiv:2012.08015.

Sauer, A, A Cooper, and RB Gramacy. 2022. "Vecchia-approximated Deep Gaussian Processes for Computer Experiments." *pre-print on arXiv:2204.02904*

Gramacy, RB. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman Hall, 2020.

Examples

```
# See "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

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

Gneiting, T, and AE Raftery. 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association 102* (477), 359-378.

sq_dist *Calculates squared pairwise distances*

Description

Calculates squared pairwise euclidean distances using C.

Usage

```
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

Gramacy, RB and F Sun. (2016). laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R. *Journal of Statistical Software 72* (1), 1-46. doi:10.18637/jss.v072.i01

Examples

```
x <- seq(0, 1, length = 10)
d2 <- sq_dist(x)
```

trim *Trim/Thin MCMC iterations*

Description

Acts on a gp, gpvec, dgp2, dgp2vec, dgp3vec, or dgp3 object. Removes the specified number of MCMC iterations (starting at the first iteration). After these samples are removed, the remaining samples are optionally thinned.

Usage

```
trim(object, burn, thin)

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

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

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

## S3 method for class 'dgp2vec'
trim(object, burn, thin = 1)

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

## S3 method for class 'dgp3vec'
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 "fit_one_layer", "fit_two_layer", or "fit_three_layer"
# for an example
```

Index