Package 'deepgp'

May 8, 2026

Type Package

Title Bayesian Deep Gaussian Processes using MCMC

Version 1.2.1

Date 2026-02-06

Depends R (>= 3.6)

Description Performs Bayesian posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2023, <doi:10.48550/arXiv.2012.08015>). See Sauer (2023, <http://hdl.handle.net/10919/114845>) for comprehensive methodological details and <https://bitbucket.org/gramacylab/deepgp-ex/> for a variety of coding examples. Models are trained through MCMC including elliptical slice sampling of latent Gaussian layers and Metropolis-Hastings sampling of kernel hyperparameters. Gradient-enhancement and gradient predictions are offered following Booth (2025, <doi:10.48550/arXiv.2512.18066>). Vecchia approximation for faster computation is implemented following Sauer, Cooper, and Gramacy (2023, <doi:10.48550/arXiv.2204.02904>). Optional monotonic warpings are implemented following Barnett et al. (2025, <doi:10.48550/arXiv.2408.01540>). Downstream tasks include sequential design through active learning Cohn/integrated mean squared error (ALC/IMSE; Sauer, Gramacy, and Higdon, 2023), optimization through expected improvement (EI; Gramacy, Sauer, and Wycoff, 2022, <doi:10.48550/arXiv.2112.07457>), and contour location through entropy (Booth, Renganathan, and Gramacy, 2025, <doi:10.48550/arXiv.2308.04420>). Models extend up to three layers deep; a one layer model is equivalent to typical Gaussian process regression. Incorporates OpenMP and SNOW parallelization and utilizes C/C++ under the hood.

License LGPL

Encoding UTF-8

NeedsCompilation yes

Imports grDevices, graphics, stats, doParallel, foreach, parallel, GpGp, fields, Matrix, Rcpp, mvtnorm, FNN, abind

LinkingTo Rcpp, RcppArmadillo,

Suggests interp, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.3

Author Annie S. Booth [aut, cre]

Maintainer Annie S. Booth <annie_booth@vt.edu>

Repository CRAN

Date/Publication 2026-02-09 14:50:02 UTC

Contents

`deepgp-package` *Package deepgp*

Description

Performs Bayesian posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2023). See Sauer (2023) for comprehensive methodological details and https://bitbucket.org/gramacylab/deepgp-ex/ for a variety of coding examples. Models are trained through MCMC including elliptical slice sampling of latent Gaussian layers and Metropolis-Hastings sampling of kernel hyperparameters. Gradient-enhancement and gradient predictions are offered following Booth (2025). Vecchia approximation for faster computation is implemented following Sauer, Cooper, and Gramacy (2023). Optional monotonic warpings are implemented following Barnett et al. (2025). Downstream tasks include sequential design through active learning Cohn/integrated mean squared error (ALC/IMSE; Sauer, Gramacy, and Higdon, 2023), optimization through expected improvement (EI; Gramacy, Sauer, and Wycoff, 2022), and contour location through entropy (Booth, Renganathan, and Gramacy, 2025). Models extend up to three layers deep; a one layer model is equivalent to typical Gaussian process regression. Incorporates OpenMP and SNOW parallelization and utilizes C/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 or entropy)
- `plot`: produces trace plots, hidden layer plots, and posterior predictive 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 S. Booth <annie_booth@ncsu.edu>

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.* http://hdl.handle.net/10919/114845

Booth, A. S. (2025). Deep Gaussian processes with gradients. arXiv:2512.18066

Sauer, A., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. arXiv:2012.08015

Sauer, A., Cooper, A., & Gramacy, R. B. (2023). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics, 32*(3), 824-837. arXiv:2204.02904

Gramacy, R. B., Sauer, A. & Wycoff, N. (2022). Triangulation candidates for Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS), 35,* 35933-35945. arXiv:2112.07457

Booth, A., Renganathan, S. A. & Gramacy, R. B. (2025). Contour location for reliability in airfoil simulation experiments using deep Gaussian processes. *Annals of Applied Statistics, 19*(1), 191-211. arXiv:2308.04420

Barnett, S., Beesley, L. J., Booth, A. S., Gramacy, R. B., & Osthus D. (2025). Monotonic warpings for additive and deep Gaussian processes. *Statistics and Computing, 35*(3), 65. arXiv:2408.01540

Examples

```
# See vignette, ?fit_one_layer, ?fit_two_layer, ?fit_three_layer,
# ?ALC, or ?IMSE for examples
# Many more examples including real-world computer experiments 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 for SNOW parallelization

Details

Not yet implemented for Vecchia-approximated fits or Matern kernels.

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., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. 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

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# ------------------------------------------------------------
# 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)
# nugget estimated
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,
  ord = NULL,
  ...
)

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

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

```
)
```

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
ord	optional ordering to be used in Vecchia re-approximation (only for fits with vecchia = TRUE when re_approx = TRUE')

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
```

---

crps *Calculates CRPS*

Description

Calculates continuous ranked probability score (lower CRPS indicate better fits, better uncertainty quantification).

Usage

```
crps(y, mu, s2)
```

Arguments

y	response vector
mu	predicted mean
s2	predicted point-wise variances

References

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

Examples

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/
```

---

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,
  dydx = NULL,
  nmcmc = 10000,
  sep = FALSE,
  verb = TRUE,
  theta_0 = 0.01,
  g_0 = 0.001,
  true_g = NULL,
  v = 2.5,
  settings = NULL,
  cov = c("matern", "exp2"),
  vecchia = FALSE,
  m = NULL,
  ord = NULL,
  cores = NULL
)
```

Arguments

x	vector or matrix of input locations
y	vector of response values
dydx	optional matrix of observed gradients, rows correspond to x locations, columns contain partial derivatives with respect to that input dimension (dim(dy) must match dim(x))
nmcmc	number of MCMC iterations
sep	logical indicating whether to use separable (sep = TRUE) or isotropic (sep = FALSE) lengthscales
verb	logical indicating whether to print iteration progress
theta_0	initial value for theta
g_0	initial value for g (only used if true_g = NULL)
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.
v	Matern smoothness parameter (only used if cov = "matern")
settings	hyperparameters for proposals and priors (see details)
cov	covariance kernel, either Matern ("matern") or squared exponential ("exp2")
vecchia	logical indicating whether to use Vecchia approximation
m	size of Vecchia conditioning sets, defaults to the lower of 25 or the maximum available (only used if vecchia = TRUE)
ord	optional ordering for Vecchia approximation, must correspond to rows of x, defaults to random
cores	number of cores to use for OpenMP parallelization (vecchia = TRUE only). Defaults to min(4, maxcores - 1) where maxcores is the number of detectable available cores.

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.

NOTE on OpenMP: The Vecchia implementation relies on OpenMP parallelization for efficient computation. This function will produce a warning message if the package was installed without OpenMP (this is the default for CRAN packages installed on Apple machines). To set up OpenMP parallelization, download the package source code and install using the gcc/g++ compiler.

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. Default priors differ for noisy/deterministic settings. All default values are visible in the internal deepgp:::check_settings function. 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, plot, 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")
- dydx: copy of dydx (if not NULL)
- grad_indx: stacked partial derivative indices (only if dydx is provided)
- g: vector of MCMC samples for g
- theta: vector of MCMC samples for theta
- tau2: vector of MLE estimates for tau2 (scale parameter)
- x_approx: Vecchia approximation object (vecchia = TRUE only)
- ll: vector of MVN log likelihood for each Gibbs iteration
- time: computation time in seconds

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.*

Sauer, A., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. arXiv:2012.08015

Booth, A. S. (2025). Deep Gaussian processes with gradients. arXiv:2512.18066

Sauer, A., Cooper, A., & Gramacy, R. B. (2023). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics, 32*(3), 824-837. arXiv:2204.02904

Examples

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# Booth function (inspired by the Higdon function)
f <- function(x) {
```

```
  i <- which(x <= 0.58)
  x[i] <- sin(pi * x[i] * 6) + cos(pi * x[i] * 12)
  x[-i] <- 5 * x[-i] - 4.9
  return(x)
}

# Training data
x <- seq(0, 1, length = 25)
y <- f(x)

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

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

# Example 1: nugget fixed, calculating EI
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, 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 2: convert fit to Vecchia object before predicting
# (this is faster if the training data set is large)
fit <- to_vec(fit)
fit <- predict(fit, xx, cores = 1)
plot(fit)

# Example 3: using Vecchia for training and testing
fit <- fit_one_layer(x, y, nmcmc = 2000, true_g = 1e-6, 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.

Currently, there are no pmx, monowarp, or dydx options.

Usage

```
fit_three_layer(
  x,
  y,
  nmcmc = 10000,
  D = ifelse(is.matrix(x), ncol(x), 1),
  verb = TRUE,
  w_0 = NULL,
  z_0 = NULL,
  theta_y_0 = 0.01,
  theta_w_0 = 0.1,
  theta_z_0 = 0.1,
  g_0 = 0.001,
  true_g = NULL,
  v = 2.5,
  settings = NULL,
  cov = c("matern", "exp2"),
  vecchia = FALSE,
  m = NULL,
  ord = NULL,
  cores = NULL
)
```

Arguments

x	vector or matrix of input locations
y	vector of response values
nmcmc	number of MCMC iterations
D	integer designating dimension of hidden layers, defaults to dimension of x
verb	logical indicating whether to print iteration progress
w_0	initial value for hidden layer w (rows must correspond to rows of x, requires ncol(w_0) = D. Defaults to the identity mapping. If nrow(w_0) < nrow(x), missing initial values are filled-in with the GP posterior mean.
z_0	initial value for hidden layer z (rows must correspond to rows of x, requires ncol(z_0) = D. Defaults to the identity mapping. If nrow(z_0) < nrow(x), missing initial values are filled-in with the GP posterior mean.
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

g_0	initial value for g
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.
v	Matern smoothness parameter (only used if cov = "matern")
settings	hyperparameters for proposals and priors (see details)
cov	covariance kernel, either Matern ("matern") or squared exponential ("exp2")
vecchia	logical indicating whether to use Vecchia approximation
m	size of Vecchia conditioning sets, defaults to the lower of 25 or the maximum available (only used if vecchia = TRUE)
ord	optional ordering for Vecchia approximation, must correspond to rows of x, defaults to random, is applied to x, w, and z
cores	number of cores to use for OpenMP parallelization (vecchia = TRUE only). Defaults to min(4, maxcores - 1) where maxcores is the number of detectable available cores.

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.

NOTE on OpenMP: The Vecchia implementation relies on OpenMP parallelization for efficient computation. This function will produce a warning message if the package was installed without OpenMP (this is the default for CRAN packages installed on Apple machines). To set up OpenMP parallelization, download the package source code and install using the gcc/g++ compiler.

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. Default priors differ for noisy/deterministic settings. All default values are visible in the internal deepgp:::check_settings function. 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 scale on the latent layers (tau2_z and tau2_w) may also be specified in settings. Defaults to 1.

When w_0 = NULL and/or z_0 = NULL, the hidden layers are initialized at x (i.e., the identity mapping). If w_0 and/or z_0 is of dimension nrow(x) - 1 by D, the final row is filled-in using the GP posterior mean. 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
- tau2_y: vector of MLE estimates for tau2 on outer layer
- 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
- w_approx: Vecchia approximation object for outer layer (vecchia = TRUE only)
- z_approx: Vecchia approximation object for middle layer (vecchia = TRUE only)
- x_approx: Vecchia approximation object for inner layer (vecchia = TRUE only)
- ll: vector of MVN log likelihood of the outer layer for reach Gibbs iteration
- time: computation time in seconds

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.* http://hdl.handle.net/10919/114845

Sauer, A., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. arXiv:2012.08015

Sauer, A., Cooper, A., & Gramacy, R. B. (2023). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics, 32*(3), 824-837. arXiv:2204.02904

Examples

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# G function in 2 dimensions (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 <- 500
xx <- matrix(runif(n_test * d), ncol = d)
yy <- apply(xx, 1, f)

i <- interp::interp(xx[, 1], xx[, 2], yy)
image(i, col = heat.colors(128))
contour(i, add = TRUE)
contour(i, level = -0.5, col = 4, add = TRUE) # potential failure limit
points(x)

# Example 1: nugget fixed, calculating entropy
fit <- fit_three_layer(x, y, nmcmc = 2000, true_g = 1e-6)
plot(fit)
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, entropy_limit = -0.5, cores = 1)
plot(fit)
i <- interp::interp(xx[, 1], xx[, 2], fit$entropy)
image(i, col = heat.colors(128), main = "Entropy")

# Example 2: using Vecchia
fit <- fit_three_layer(x, y, nmcmc = 2000, true_g = 1e-6, vecchia = TRUE, m = 10)
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,
  dydx = NULL,
  nmcmc = 10000,
  D = ifelse(is.matrix(x), ncol(x), 1),
  monowarp = FALSE,
  pmx = FALSE,
  verb = TRUE,
  w_0 = NULL,
  theta_y_0 = 0.01,
  theta_w_0 = 0.1,
  g_0 = 0.001,
  true_g = NULL,
  v = 2.5,
  settings = NULL,
  cov = c("matern", "exp2"),
  vecchia = FALSE,
  m = NULL,
  ord = NULL,
  cores = NULL
)
```

Arguments

x	vector or matrix of input locations
y	vector of response values
dydx	optional matrix of observed gradients, rows correspond to x locations, columns contain partial derivatives with respect to that input dimension (dim(dy) must match dim(x))
nmcmc	number of MCMC iterations
D	integer designating dimension of hidden layer, defaults to dimension of x
monowarp	logical or numeric. If FALSE, warpings are not forced to be monotonic. If TRUE, each input dimension is individually monotonically warped with a default grid size of 50. If numeric, triggers monotonic warpings with the provided grid size.
pmx	"prior mean x", logical indicating whether w should have prior mean of x (TRUE, requires D = ncol(x)) or prior mean zero (FALSE). pmx = TRUE is recommended for higher dimensions.
verb	logical indicating whether to print iteration progress
w_0	initial value for hidden layer w (rows must correspond to rows of x, requires ncol(w_0) = D. Defaults to the identity mapping. If nrow(w_0) < nrow(x), missing initial values are filled-in with the GP posterior mean.
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

g_0	initial value for g (only used if true_g = NULL)
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.
v	Matern smoothness parameter (only used if cov = "matern")
settings	hyperparameters for proposals and priors (see details)
cov	covariance kernel, either Matern ("matern") or squared exponential ("exp2")
vecchia	logical indicating whether to use Vecchia approximation
m	size of Vecchia conditioning sets, defaults to the lower of 25 or the maximum available (only used if vecchia = TRUE)
ord	optional ordering for Vecchia approximation, must correspond to rows of x, defaults to random, is applied to both x and w
cores	number of cores to use for OpenMP parallelization (vecchia = TRUE only). Defaults to min(4, maxcores - 1) where maxcores is the number of detectable available cores.

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.

When monowarp = TRUE, each input dimension is warped separately and monotonically. This requires D = ncol(x) with x scaled to the unit cube. New in version 1.2.0 - monotonic warpings estimate separate scale parameters (tau2_w) on each latent node and use an isotropic lengthscale on the outer layer. As a default, monotonic warpings use the reference grid: seq(0, 1, length = 50). The grid size may be controlled by passing a numeric integer to monowarp (i.e., monowarp = 100 uses the grid seq(0, 1, length = 100)).

When pmx = TRUE, the prior on the latent layer is set at x (rather than the default of zero). This requires D = ncol(x).

NOTE on OpenMP: The Vecchia implementation relies on OpenMP parallelization for efficient computation. This function will produce a warning message if the package was installed without OpenMP (this is the default for CRAN packages installed on Apple machines). To set up OpenMP parallelization, download the package source code and install using the gcc/g++ compiler.

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. Default priors differ for noisy/deterministic settings and depend on whether monowarp = TRUE. All default values are visible in the internal deepgp:::check_settings function. 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 scale on the latent layer (tau2_w) may also be specified in settings. Defaults to 1.

When w_0 = NULL, the hidden layer is initialized at x (i.e., the identity mapping). If w_0 is of dimension nrow(x) - 1 by D, the final row is filled-in using the GP posterior mean. 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")
- x_grid: grid used for monotonic warpings (monowarp = TRUE only)
- dydx: copy of dydx (if not NULL)
- grad_indx: stacked partial derivative indices (only if dydx is provided)
- g: vector of MCMC samples for g
- tau2_y: vector of MLE estimates for tau2 on the outer layer
- theta_y: vector of MCMC samples for theta_y (length scale of outer layer)
- tau2_w: matrix of MLE estimates for tau2 on inner layer (only returned if monowarp = TRUE, otherwise this is fixed in settings)
- theta_w: matrix of MCMC samples for theta_w (length scale of inner layer)
- w: list of MCMC samples for hidden layer w
- w_grid: w values at x_grid locations (monowarp = TRUE only)
- w_approx: Vecchia approximation object for outer layer (vecchia = TRUE only)
- x_approx: Vecchia approximation object for inner layer (vecchia = TRUE only)
- ll: vector of MVN log likelihood of the outer layer for reach Gibbs iteration
- time: computation time in seconds

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.* http://hdl.handle.net/10919/114845

Booth, A. S. (2025). Deep Gaussian processes with gradients. arXiv:2512.18066

Sauer, A., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. arXiv:2012.08015

Sauer, A., Cooper, A., & Gramacy, R. B. (2023). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics, 32*(3), 824-837. arXiv:2204.02904

Barnett, S., Beesley, L. J., Booth, A. S., Gramacy, R. B., & Osthus D. (2025). Monotonic warpings for additive and deep Gaussian processes. *Statistics and Computing, 35*(3), 65. arXiv:2408.01540

Examples

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# Booth function (inspired by the Higdon function)
f <- function(x) {
  i <- which(x <= 0.58)
  x[i] <- sin(pi * x[i] * 6) + cos(pi * x[i] * 12)
  x[-i] <- 5 * x[-i] - 4.9
  return(x)
}

# Training data
x <- seq(0, 1, length = 25)
y <- f(x)

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

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

# Example 1: nugget fixed, using continue
fit <- fit_two_layer(x, y, nmcmc = 1000, true_g = 1e-6)
plot(fit)
fit <- continue(fit, 1000)
plot(fit, hidden = TRUE) # trace plots and ESS samples
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)

# Example 2: using Vecchia, re-approximated after burn-in
fit <- fit_two_layer(x, y, nmcmc = 1000, true_g = 1e-6, vecchia = TRUE, m = 10)
fit <- continue(fit, 1000, re_approx = TRUE)
plot(fit, hidden = TRUE) # trace plots and ESS samples
fit <- trim(fit, 1000, 2)
fit <- predict(fit, xx, cores = 1)
plot(fit)

# Example 3: using monotonic warpings
fit <- fit_two_layer(x, y, nmcmc = 2000, true_g = 1e-6, monowarp = TRUE)
plot(fit, hidden = TRUE) # trace plots and ESS samples
fit <- trim(fit, 1000, 2)
```

```
fit <- predict(fit, xx, cores = 1)
plot(fit)
```

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 for SNOW parallelization

Details

Not yet implemented for Vecchia-approximated fits or Matern kernels.

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., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. 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

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/

# ------------------------------------------------------
# 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)
# nugget estimated
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 outer log likelihood, length scale, and nugget hyperparameters. Generates plots of hidden layers for

low dimensions or monotonic warpings. Generates plots of the posterior mean and estimated 90% 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. If there are too many hyperparameters to plot them all, then it is most useful to visualize the log likelihood (e.g., `plot(fit$ll, type = "l")`).

In one dimension, hidden layer plots show 100 evenly distributed samples. In two dimensions (two layer only), hidden layer plots show 3 samples. 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.

Examples

```
# See ?fit_one_layer, ?fit_two_layer, or ?fit_three_layer
# for examples
```

post_sample *Generates joint posterior samples from a trained GP/DGP*

Description

Acts on a gp, gpvec, dgp2, dgp2vec, dgp3, or dgp3vec object. Generates joint samples from the posterior distribution at the provided locations.

Usage

```
post_sample(object, x_new, nper = 1, ...)

## S3 method for class 'gp'
post_sample(object, x_new, nper = 1, grad = FALSE, cores = 1, ...)

## S3 method for class 'gpvec'
post_sample(
  object,
  x_new,
  nper = 1,
  m = NULL,
  ord_new = NULL,
  grad = FALSE,
  cores = 1,
  ...
)

## S3 method for class 'dgp2'
post_sample(
  object,
  x_new,
  nper = 1,
  grad = FALSE,
  mean_map = TRUE,
  cores = 1,
  ...
)

## S3 method for class 'dgp2vec'
post_sample(
  object,
  x_new,
  nper = 1,
  m = NULL,
  ord_new = NULL,
  grad = FALSE,
  mean_map = TRUE,
```

```
  cores = 1,
  ...
)

## S3 method for class 'dgp3'
post_sample(object, x_new, nper = 1, mean_map = TRUE, cores = 1, ...)

## S3 method for class 'dgp3vec'
post_sample(
  object,
  x_new,
  nper = 1,
  m = NULL,
  ord_new = NULL,
  mean_map = TRUE,
  cores = 1,
  ...
)
```

Arguments

object	object from fit_one_layer, fit_two_layer, or fit_three_layer with burn-in already removed
x_new	vector or matrix of predictive input locations
nper	the number of samples to generate from each MCMC iteration. The total number of samples will equal nper*object$nmcmc
...	N/A
grad	logical indicating whether to additionally calculate/return samples of the gradient (one and two layer models only)
cores	number of cores to use for SNOW parallelization
m	size of Vecchia conditioning sets (only for fits with vecchia = TRUE), defaults to the twice the m used for MCMC
ord_new	optional ordering for Vecchia approximation with lite = FALSE, must correspond to rows of x_new, defaults to random, is applied to all layers in deeper models
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)

Details

By default, one sample is generated per each MCMC iteration. This may be increased with the nper argument.

SNOW parallelization reduces computation time but requires more memory storage.

Value

If grad = FALSE, returns matrix of samples. Rows correspond to x_new locations. If grad = TRUE, returns a list with y and dydx containing the respective samples.

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.*

Examples

```
# Simple step function
f <- function(x) {
  return(pnorm((x - 0.5) / 0.065))
}

# Training data
x <- seq(0, 1, length = 5)
y <- f(x)

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

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

# Conduct MCMC
fit <- fit_two_layer(x, y, nmcmc = 2000, true_g = 1e-6, cov = "exp2")
plot(fit, hidden = TRUE)
fit <- trim(fit, 1000, 2)

# Generate posterior samples, including gradients
samples <- post_sample(fit, xx, grad = TRUE, cores = 1)

# Plot samples
par(mfrow = c(1, 2))
matplot(xx, t(samples$y), type = "l")
points(x, y, pch = 20)
matplot(xx, t(samples$dy), type = "l")
```

---

predict *Predict posterior mean and variance/covariance*

Description

Acts on a gp, gpvec, dgp2, dgp2vec, dgp3, or dgp3vec object. Calculates posterior mean and variance/covariance over specified input locations. Optionally calculates expected improvement (EI) or entropy over candidate inputs. Optionally utilizes SNOW parallelization.

Usage

```
## S3 method for class 'gp'
predict(
  object,
  x_new,
  lite = TRUE,
  grad = FALSE,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

## S3 method for class 'dgp2'
predict(
  object,
  x_new,
  lite = TRUE,
  grad = FALSE,
  store_latent = FALSE,
  mean_map = TRUE,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

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

```
## S3 method for class 'gpvec'
predict(
  object,
  x_new,
  m = NULL,
  ord_new = NULL,
  lite = TRUE,
  grad = FALSE,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

## S3 method for class 'dgp2vec'
predict(
  object,
  x_new,
  m = NULL,
  ord_new = NULL,
  lite = TRUE,
  grad = FALSE,
  store_latent = FALSE,
  mean_map = TRUE,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

## S3 method for class 'dgp3vec'
predict(
  object,
  x_new,
  m = NULL,
  ord_new = NULL,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)
```

Arguments

object	object from fit_one_layer, fit_two_layer, or fit_three_layer with burn-in already removed
x_new	vector or matrix of predictive input locations
lite	logical indicating whether to calculate only point-wise variances (lite = TRUE) or full covariance (lite = FALSE)
grad	logical indicating whether to additionally calculate/return predictions of the gradient (one and two layer models only)
return_all	logical indicating whether to return mean and point-wise variance prediction for ALL samples (only available for lite = TRUE)
EI	logical indicating whether to calculate expected improvement (for minimizing the response)
entropy_limit	optional limit state for entropy calculations (separating passes and failures), default value of NULL bypasses entropy calculations
cores	number of cores to utilize for SNOW parallelization
...	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)
m	size of Vecchia conditioning sets, defaults to the lower of twice the m used for MCMC or the maximum available (only for fits with vecchia = TRUE),
ord_new	optional ordering for Vecchia approximation with lite = FALSE, must correspond to rows of x_new, defaults to random, is applied to all layers in deeper models

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. Entropy is calculated based on two classes separated by the specified limit. See Sauer (2023, Chapter 3) 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)
- mean_all: predicted posterior mean for each sample (rows correspond to iterations), only returned when return_all = TRUE
- s2_all: predicted point-wise variances for each sample (rows correspond to iterations), only returned when return_all = TRUE
- Sigma: predicted posterior covariance, indices correspond to x_new locations (only returned when lite = FALSE)
- grad_mean: predicted posterior mean of the gradient (rows correspond to x_new, columns correspond to dimension, only returned when grad = TRUE)
- grad_s2: predicted point-wise variances of the gradient (rows correspond to x_new, columns correspond to dimension, only returned when grad = TRUE)
- EI: vector of expected improvement values, indices correspond to x_new locations (only returned when EI = TRUE)
- entropy: vector of entropy values, indices correspond to x_new locations (only returned when entropy_limit is numeric)
- w_new: array of hidden layer mappings, with dimensions corresponding to iteration, then x_new location, then dimension (only returned when store_latent = TRUE)
- z_new: array of hidden layer mappings, with dimensions corresponding to iteration, then x_new location, then dimension (only returned when store_latent = TRUE)

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

References

Sauer, A. (2023). Deep Gaussian process surrogates for computer experiments. *Ph.D. Dissertation, Department of Statistics, Virginia Polytechnic Institute and State University.* http://hdl.handle.net/10919/114845

Booth, A. S. (2025). Deep Gaussian processes with gradients. arXiv:2512.18066

Sauer, A., Gramacy, R.B., & Higdon, D. (2023). Active learning for deep Gaussian process surrogates. *Technometrics, 65,* 4-18. arXiv:2012.08015

Sauer, A., Cooper, A., & Gramacy, R. B. (2023). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics, 32*(3), 824-837. arXiv:2204.02904

Gramacy, R. B., Sauer, A. & Wycoff, N. (2022). Triangulation candidates for Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS), 35,* 35933-35945. arXiv:2112.07457

Booth, A., Renganathan, S. A. & Gramacy, R. B. (2025). Contour location for reliability in airfoil simulation experiments using deep Gaussian processes. *Annals of Applied Statistics, 19*(1), 191-211. arXiv:2308.04420

Barnett, S., Beesley, L. J., Booth, A. S., Gramacy, R. B., & Osthus D. (2025). Monotonic warpings for additive and deep Gaussian processes. *Statistics and Computing, 35*(3), 65. arXiv:2408.01540

Examples

```
# See ?fit_one_layer, ?fit_two_layer, or ?fit_three_layer
# for examples
```

`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

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/
```

`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.

Examples

```
# Additional examples including real-world computer experiments are available at:
# https://bitbucket.org/gramacylab/deepgp-ex/
```

`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)
```

to_vec *Converts non-Vecchia object to its Vecchia version*

Description

Converts an object of class "gp", "dgp2", or "dgp3" to its Vecchia equivalent ("gpvec", "dgp2vec", or "dgp3vec").

Usage

```
to_vec(object, m = NULL, ord = NULL, cores = NULL)
```

Arguments

object	object from fit_one_layer, fit_two_layer, or fit_three_layer with vecchia = FALSE
m	size of Vecchia conditioning sets, defaults to the lower of 25 or maximum available
ord	optional ordering for Vecchia approximation, defaults to random
cores	number of cores to use for OpenMP parallelization. Defaults to min(4, maxcores - 1) where maxcores is the number of detectable available cores.

Details

Creates and appends Vecchia-approximation objects to the provided fit. Defaults to random ordering and nearest neighbors conditioning. Useful for speeding up predictions when the testing size is large but the training size is small.

Value

The same object, with x_approx, w_approx, and z_approx appended (depending on the number of layers).

Examples

```
# See ?fit_one_layer for an example
```

`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 examples
```

Index