

Package ‘deepgp’

August 7, 2023

Type Package

Title Bayesian Deep Gaussian Processes using MCMC

Version 1.1.1

Date 2023-08-04

Author Annie S. Booth <annie_booth@ncsu.edu>

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

Depends R (>= 3.6)

Description Performs Bayesian posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2023, <[arXiv:2012.08015](https://arxiv.org/abs/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. Vecchia-approximation for faster computation is implemented following Sauer, Cooper, and Gramacy (2022, <[arXiv:2204.02904](https://arxiv.org/abs/2204.02904)>). 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, 2021 <[arXiv:2112.07457](https://arxiv.org/abs/2112.07457)>), and contour location through entropy (Sauer, 2023). 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, Matrix, Rcpp, mvtnorm, FNN

LinkingTo Rcpp, RcppArmadillo,

Suggests interp, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.2

Repository CRAN

Date/Publication 2023-08-07 20:50:05 UTC

R topics documented:

deepgp-package	2
ALC	3
continue	6
crps	8
fit_one_layer	8
fit_three_layer	11
fit_two_layer	15
IMSE	19
plot	21
predict	23
rmse	26
score	27
sq_dist	27
trim	28

Index **30**

deepgp-package	<i>Package deepgp</i>
----------------	-----------------------

Description

Performs Bayesian posterior inference for deep Gaussian processes following Sauer, Gramacy, and Higdon (2023, <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. 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, 2023), optimization through expected improvement (EI; Gramacy, Sauer, and Wycoff, 2021 <arXiv:2112.07457>), and contour location through entrop(Sauer, 2023). 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.*

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. (2022). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics,* 1-14. 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

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

<code>object</code>	object of class <code>gp</code> , <code>dgp2</code> , or <code>dgp3</code>
<code>x_new</code>	matrix of possible input locations, if <code>object</code> has been run through <code>predict</code> the previously stored <code>x_new</code> is used
<code>ref</code>	optional reference grid for ALC approximation, if <code>ref = NULL</code> then <code>x_new</code> is used
<code>cores</code>	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., 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

```
# -----
# 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 <code>fit_one_layer</code> , <code>fit_two_layer</code> , or <code>fit_three_layer</code>
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 <code>vecchia = TRUE</code>)
...	N/A

Details

See `fit_one_layer`, `fit_two_layer`, or `fit_three_layer` for details on MCMC. The resulting object will have `nmc` equal to the previous `nmc` 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	<i>Calculates CRPS</i>
------	------------------------

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.

fit_one_layer	<i>MCMC sampling for one layer GP</i>
---------------	---------------------------------------

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,
  sep = FALSE,
  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
nmc	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
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".

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. 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
- `nmc`: number of MCMC iterations
- `settings`: copy of proposal/prior settings
- `v`: copy of Matern smoothness parameter ($\nu = 999$ 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)
- `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

Sauer, A., Cooper, A., & Gramacy, R. B. (2022). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics,* 1-14. arXiv:2204.02904

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 θ_z , θ_w , and θ_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, ν governs smoothness.

Usage

```
fit_three_layer(
```

```

x,
y,
nmcnc = 10000,
D = ifelse(is.matrix(x), ncol(x), 1),
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
nmcnc	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 (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

`pmx = TRUE` option not yet implemented for three-layer DGP.

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"`.

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 , θ_y , θ_w , and θ_z follow a uniform sliding window scheme, e.g.

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

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 , θ_y , θ_w , and θ_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.

In the current version, the three-layer does not have any equivalent setting for `pmx = TRUE` as in `fit_two_layer`.

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 inner hidden layer z is zero, but may be adjusted to x using `settings = list(z_prior_mean = x)`. The prior mean of the middle hidden layer w is set at zero is is not user adjustable. 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
- ll: vector of MVN log likelihood of the outer layer 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
- Sauer, A., Cooper, A., & Gramacy, R. B. (2022). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics,* 1-14. arXiv:2204.02904

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 <- interp::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 θ_w and θ_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, ν governs smoothness.

Usage

```

fit_two_layer(
  x,
  y,
  nmcmc = 10000,
  D = ifelse(is.matrix(x), ncol(x), 1),
  pmx = FALSE,
  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
nmcmc	number of MCMC iterations
D	integer designating dimension of hidden layer, defaults to dimension of x
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. May be numeric, in which case the specified argument is used as the scale (tau2) in the latent w layer (default is 1). Small values encourage identity mappings.
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"`.

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 , θ_y , and θ_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 , θ_y , and θ_w follow Gamma distributions with shape parameters (α) and rate parameters (β) 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 $w_0 = \text{NULL}$, 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 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 ($\nu = 999$ indicates `cov = "exp2"`)
- `g`: vector of MCMC samples for g
- `theta_y`: vector of MCMC samples for θ_y (length scale of outer layer)
- `theta_w`: matrix of MCMC samples for θ_w (length scale of inner layer)
- `tau2`: vector of MLE estimates for τ_2 (scale parameter of outer layer)
- `w`: list of MCMC samples for hidden layer w
- `ll`: vector of MVN log likelihood of the outer layer 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
- Sauer, A., Cooper, A., & Gramacy, R. B. (2022). Vecchia-approximated deep Gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics,* 1-14. arXiv:2204.02904

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

<code>object</code>	object of class <code>gp</code> , <code>dgp2</code> , or <code>dgp3</code>
<code>x_new</code>	matrix of possible input locations, if object has been run through <code>predict</code> the previously stored <code>x_new</code> is used
<code>cores</code>	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., 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

```
# -----
# 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)]

```

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 one-

dimensional inputs. 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. 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	<i>Predict posterior mean and variance/covariance</i>
---------	---

Description

Acts on a gp, dgp2, or dgp3 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,
  return_all = FALSE,
  EI = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

## S3 method for class 'dgp2'
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 '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 = object$m,
  lite = TRUE,
  return_all = FALSE,
  entropy_limit = NULL,
  cores = 1,
  ...
)

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

## S3 method for class 'dgp3vec'
predict(
  object,
  x_new,
  m = object$m,
  lite = TRUE,
  store_latent = FALSE,
  mean_map = TRUE,
  return_all = 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

<code>x_new</code>	matrix of predictive input locations
<code>lite</code>	logical indicating whether to calculate only point-wise variances (<code>lite = TRUE</code>) or full covariance (<code>lite = FALSE</code>)
<code>return_all</code>	logical indicating whether to return mean and point-wise variance prediction for ALL samples (only available for <code>lite = TRUE</code>)
<code>EI</code>	logical indicating whether to calculate expected improvement (for minimizing the response)
<code>entropy_limit</code>	optional limit state for entropy calculations (separating passes and failures), default value of NULL bypasses entropy calculations
<code>cores</code>	number of cores to utilize in parallel
<code>...</code>	N/A
<code>store_latent</code>	logical indicating whether to store and return mapped values of latent layers (two or three layer models only)
<code>mean_map</code>	logical indicating whether to map hidden layers using conditional mean (<code>mean_map = TRUE</code>) or using a random sample from the full MVN distribution (two or three layer models only), <code>mean_map = FALSE</code> is not yet implemented for fits with <code>vecchia = TRUE</code>
<code>m</code>	size of Vecchia conditioning sets (only for fits with <code>vecchia = TRUE</code>), defaults to the <code>m</code> 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. 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 (column indices), only returned when `return_all = TRUE`
- `s2_all` predicted point-wise variances for each sample (column indices), only returned when `return-all = TRUE`
- `Sigma`: predicted posterior covariance, indices correspond to `x_new` locations (only returned when `lite = FALSE`)

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

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

<code>y</code>	response vector
<code>mu</code>	predicted mean

score	<i>Calculates score</i>
-------	-------------------------

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	<i>Calculates squared pairwise distances</i>
---------	--

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

Details

The resulting object will have `nmc` equal to the previous `nmc` 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

ALC, [3](#), [3](#)

continue, [3](#), [6](#)

crps, [8](#)

deepgp-package, [2](#)

fit_one_layer, [2](#), [8](#)

fit_three_layer, [3](#), [11](#)

fit_two_layer, [2](#), [15](#)

IMSE, [3](#), [19](#)

plot, [3](#), [21](#)

predict, [3](#), [23](#)

rmse, [26](#)

score, [27](#)

sq_dist, [27](#)

trim, [3](#), [28](#)