

Package ‘shrinkGPR’

August 19, 2025

Type Package

Title Scalable Gaussian Process Regression with Hierarchical Shrinkage Priors

Version 1.1

Maintainer Peter Knaus <peter.knaus@wu.ac.at>

Description Efficient variational inference methods for fully Bayesian Gaussian Process Regression (GPR) models with hierarchical shrinkage priors, including the triple gamma prior for effective variable selection and covariance shrinkage in high-dimensional settings. The package leverages normalizing flows to approximate complex posterior distributions. For details on implementation, see Knaus (2025) <[doi:10.48550/arXiv.2501.13173](https://doi.org/10.48550/arXiv.2501.13173)>.

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports gsl, progress, rlang, utils, methods, torch

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), shrinkTVP, plotly

Config/testthat/edition 3

SystemRequirements torch backend, for installation guide see <https://cran.r-project.org/web/packages/torch/vignettes/installation.html>

NeedsCompilation no

Author Peter Knaus [aut, cre] (ORCID: <<https://orcid.org/0000-0001-6498-7084>>)

Repository CRAN

Date/Publication 2025-08-19 12:20:02 UTC

Contents

calc_pred_moments	2
eval_pred_dens	3
gen_marginal_samples	4

gen_posterior_samples	6
kernel_functions	7
LPDS	9
plot.shrinkGPR	10
plot.shrinkGPR_marg_samples_1D	11
plot.shrinkGPR_marg_samples_2D	13
plot.shrinkTPR	14
predict.shrinkGPR	15
predict.shrinkTPR	16
shrinkGPR	18
shrinkTPR	22
simGPR	27
sylvester	29
Index	31

calc_pred_moments	<i>Calculate Predictive Moments</i>
-------------------	-------------------------------------

Description

calc_pred_moments calculates the predictive means and variances for a fitted shrinkGPR model at new data points.

Usage

```
calc_pred_moments(object, newdata, nsamp = 100)
```

Arguments

object	A shrinkGPR object representing the fitted Gaussian process regression model.
newdata	<i>Optional</i> data frame containing the covariates for the new data points. If missing, the training data is used.
nsamp	Positive integer specifying the number of posterior samples to use for the calculation. Default is 100.

Details

This function computes predictive moments by marginalizing over posterior samples from the fitted model. If the mean equation is included in the model, the corresponding covariates are used.

Value

- A list with two elements:
- means: A matrix of predictive means for each new data point, with the rows being the samples and the columns the data points.
 - vars: An array of covariance matrices, with the first dimension corresponding to the samples and second and third dimensions to the data points.

Examples

```

if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)

  # Calculate predictive moments
  momes <- calc_pred_moments(res, nsamp = 100)
}

```

eval_pred_dens

*Evaluate Predictive Densities***Description**

eval_pred_dens evaluates the predictive density for a set of points based on a fitted shrinkGPR model.

Usage

```
eval_pred_dens(x, mod, data_test, nsamp = 100, log = FALSE)
```

Arguments

x	Numeric vector of points for which the predictive density is to be evaluated.
mod	A shrinkGPR object representing the fitted Gaussian process regression model.
data_test	Data frame with one row containing the covariates for the test set. Variables in data_test must match those used in model fitting.
nsamp	Positive integer specifying the number of posterior samples to use for the evaluation. Default is 100.
log	Logical; if TRUE, returns the log predictive density. Default is FALSE.

Details

This function computes predictive densities by marginalizing over posterior samples drawn from the fitted model. If the mean equation is included in the model, the corresponding covariates are incorporated.

Value

A numeric vector containing the predictive densities (or log predictive densities) for the points in x .

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)

  # Create point at which to evaluate predictive density
  data_test <- data.frame(x1 = 0.8, x2 = 0.5)
  eval_points <- c(-1.2, -1, 0)

  eval_pred_dens(eval_points, res, data_test)

  # Is vectorized, can also be used in functions like curve
  curve(eval_pred_dens(x, res, data_test), from = -1.5, to = -0.5)
  abline(v = sin(2 * pi * 0.8), col = "red")
}
```

gen_marginal_samples *Generate Marginal Samples of Predictive Distribution*

Description

gen_marginal_samples() generates model predictions over a grid of values for one or two specified covariates, while filling in the remaining covariates either by drawing from the training data (if fixed_x is not provided) or by using a fixed values for the remaining covariates (if fixed_x is provided). The result is a set of conditional predictions that can be used to visualize the marginal effect of the selected covariates under varying input configurations.

Usage

```
gen_marginal_samples(
  mod,
  to_eval,
  nsamp = 200,
  fixed_x,
  n_eval_points,
  eval_range,
```

```

    display_progress = TRUE
  )

```

Arguments

<code>mod</code>	A shrinkGPR or shrinkTPR object representing the fitted Gaussian/t process regression model.
<code>to_eval</code>	A character vector specifying the names of the covariates to evaluate. Can be one or two variables.
<code>nsamp</code>	Positive integer specifying the number of posterior samples to generate. Default is 200.
<code>fixed_x</code>	<i>optional</i> data frame specifying a fixed covariate configuration. If provided, this configuration is used for all nonswept covariates. If omitted, covariates are sampled from the training data.
<code>n_eval_points</code>	Positive integer specifying the number of evaluation points along each axis. If missing, defaults to 100 for 1D and 30 for 2D evaluations.
<code>eval_range</code>	<i>optional</i> numeric vector (1D) or list of two numeric vectors (2D) specifying the range over which to evaluate the covariates in <code>to_eval</code> . If omitted, the range is set to the range of the swept covariate(s) in the training data.
<code>display_progress</code>	logical value indicating whether to display progress bars and messages during training. The default is TRUE.

Details

This function generates conditional predictive surfaces by evaluating the fitted model across a grid of values for one or two selected covariates. For each of the `nsamp` draws, the remaining covariates are either held fixed (if `fixed_x` is provided) or filled in by sampling a single row from the training data. The selected covariates in `to_eval` are then varied across a regular grid defined by `n_eval_points` and `eval_range`, and model predictions are computed using [calc_pred_moments](#).

The resulting samples represent conditional predictions across different covariate contexts, and can be used to visualize marginal effects, interaction surfaces, or predictive uncertainty.

Note that computational and memory requirements increase rapidly with grid size. In particular, for two-dimensional evaluations, the kernel matrix scales quadratically with the number of evaluation points per axis. Large values of `n_eval_points` may lead to high memory usage during prediction, especially when using a GPU. If memory constraints arise, consider reducing `n_eval_points`.

Value

A list containing posterior predictive summaries over the evaluation grid:

- `mean_pred`: A matrix (1D case) or array (2D case) of predicted means for each evaluation point and posterior sample.
- `grid`: The evaluation grid used to generate predictions. A numeric vector (1D) or a named list of two vectors (`grid1`, `grid2`) for 2D evaluations.

Examples

```

if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)

  # Generate posterior samples
  samps <- gen_posterior_samples(res, nsamp = 1000)

  # Plot the posterior samples
  boxplot(samps$thetas)
}

```

gen_posterior_samples *Generate Posterior Samples*

Description

gen_posterior_samples generates posterior samples of the model parameters from a fitted shrinkGPR or shrinkTPR model.

Usage

```
gen_posterior_samples(mod, nsamp = 1000)
```

Arguments

mod	A shrinkGPR object representing the fitted Gaussian process regression model.
nsamp	Positive integer specifying the number of posterior samples to generate. Default is 1000.

Details

This function draws posterior samples from the latent space and transforms them into the parameter space of the model. These samples can be used for posterior inference or further analysis.

Value

A list containing posterior samples of the model parameters:

- `thetas`: A matrix of posterior samples for the inverse lengthscale parameters.
- `sigma2`: A matrix of posterior samples for the noise variance.
- `lambda`: A matrix of posterior samples for the global shrinkage parameter.
- `betas` (optional): A matrix of posterior samples for the mean equation parameters (if included in the model).
- `lambda_mean` (optional): A matrix of posterior samples for the mean equation's global shrinkage parameter (if included in the model).

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)

  # Generate posterior samples
  samps <- gen_posterior_samples(res, nsamp = 1000)

  # Plot the posterior samples
  boxplot(samps$thetas)
}
```

Description

A set of kernel functions for Gaussian processes, including the squared exponential (SE) kernel and Matérn kernels with smoothness parameters $1/2$, $3/2$, and $5/2$. These kernels compute the covariance structure for Gaussian process regression models and are designed for compatibility with the `shrinkGPR` function.

Usage

```
kernel_se(thetas, tau, x, x_star = NULL)

kernel_matern_12(thetas, tau, x, x_star = NULL)

kernel_matern_32(thetas, tau, x, x_star = NULL)

kernel_matern_52(thetas, tau, x, x_star = NULL)
```

Arguments

thetas	A torch_tensor of dimensions $n_{\text{latent}} \times d$, representing the latent length-scale parameters.
tau	A torch_tensor of length n_{latent} , representing the latent scaling factors.
x	A torch_tensor of dimensions $N \times d$, containing the input data points.
x_star	Either NULL or a torch_tensor of dimensions $N_{\text{new}} \times d$. If NULL, the kernel is computed for x against itself. Otherwise, it computes the kernel between x and x_star.

Details

These kernel functions are used to define the covariance structure in Gaussian process regression models. Each kernel implements a specific covariance function:

- `kernel_se`: Squared exponential (SE) kernel, also known as the radial basis function (RBF) kernel. It assumes smooth underlying functions.
- `kernel_matern_12`: Matérn kernel with smoothness parameter $\nu = 1/2$, equivalent to the absolute exponential kernel.
- `kernel_matern_32`: Matérn kernel with smoothness parameter $\nu = 3/2$.
- `kernel_matern_52`: Matérn kernel with smoothness parameter $\nu = 5/2$.

The `sqdist` helper function is used internally by these kernels to compute squared distances between data points.

Note that these functions perform no input checks, as to ensure higher performance. Users should ensure that the input tensors are of the correct dimensions.

Value

A torch_tensor containing the batched covariance matrices (one for each latent sample):

- If `x_star = NULL`, the output is of dimensions $n_{\text{latent}} \times N \times N$, representing pairwise covariances between all points in x.
- If `x_star` is provided, the output is of dimensions $n_{\text{latent}} \times N_{\text{new}} \times N$, representing pairwise covariances between x_star and x.

Examples

```

if (torch::torch_is_installed()) {
  # Example inputs
  torch::torch_manual_seed(123)
  n_latent <- 3
  d <- 2
  N <- 5
  thetas <- torch::torch_randn(n_latent, d)$abs()
  tau <- torch::torch_randn(n_latent)$abs()
  x <- torch::torch_randn(N, d)

  # Compute the SE kernel
  K_se <- kernel_se(thetas, tau, x)
  print(K_se)

  # Compute the Matérn 3/2 kernel
  K_matern32 <- kernel_matern_32(thetas, tau, x)
  print(K_matern32)

  # Compute the Matérn 5/2 kernel with x_star
  x_star <- torch::torch_randn(3, d)
  K_matern52 <- kernel_matern_52(thetas, tau, x, x_star)
  print(K_matern52)
}

```

LPDS

Log Predictive Density Score

Description

LPDS calculates the log predictive density score for a fitted shrinkGPR model using a test dataset.

Usage

```
LPDS(mod, data_test, nsamp = 100)
```

Arguments

<code>mod</code>	A shrinkGPR object representing the fitted Gaussian process regression model.
<code>data_test</code>	Data frame with one row containing the covariates for the test set. Variables in <code>data_test</code> must match those used in model fitting.
<code>nsamp</code>	Positive integer specifying the number of posterior samples to use for the evaluation. Default is 100.

Details

The log predictive density score is a measure of model fit that evaluates how well the model predicts unseen data. It is computed as the log of the marginal predictive density of the observed responses.

Value

A numeric value representing the log predictive density score for the test dataset.

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)

  # Calculate true y value and calculate LPDS at specific point
  x1_new <- 0.8
  x2_new <- 0.5
  y_true <- sin(2 * pi * x1_new)
  data_test <- data.frame(y = y_true, x1 = x1_new, x2 = x2_new)
  LPDS(res, data_test)
}
```

plot.shrinkGPR

Graphical summary of posterior of theta

Description

plot.shrinkGPR generates a boxplot visualizing the posterior distribution of theta obtained from a fitted shrinkGPR object.

Usage

```
## S3 method for class 'shrinkGPR'
plot(x, nsamp = 1000, ...)
```

Arguments

x	a shrinkGPR object.
nsamp	a positive integer specifying the number of posterior samples to draw for plotting. The default is 1000.
...	further arguments passed to the internal boxplot function, such as axis labeling or plotting options. By default, las = 2 is used unless explicitly overridden by the user.

Value

Called for its side effects. Returns invisible(NULL).

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

See Also

Other plotting functions: [plot.shrinkGPR_marg_samples_1D\(\)](#), [plot.shrinkGPR_marg_samples_2D\(\)](#), [plot.shrinkTPR\(\)](#)

Examples

```
# Simulate and fit a shrinkGPR model, then plot:
sim <- simGPR()
mod <- shrinkGPR(y ~ ., data = sim$data)
plot(mod)

## Change axis label orientation
plot(mod, las = 1)
```

```
plot.shrinkGPR_marg_samples_1D
```

Plot method for 1D marginal predictions

Description

Generates a plot of 1D conditional predictive samples produced by [gen_marginal_samples](#) with a single covariate.

Usage

```
## S3 method for class 'shrinkGPR_marg_samples_1D'
plot(x, ...)
```

Arguments

x	An object of class "shrinkGPR_marg_samples_1D", typically returned by gen_marginal_samples when providing a single covariate to sweep over.
...	Additional arguments passed to plot.mcmc.tvp for customizing the plot, such as axis labels or plotting options.

Details

By default, the function visualizes the posterior predictive median and 95% and 50% credible intervals for the selected covariate across a grid of evaluation points. Axis labels are automatically inferred if not explicitly provided.

Note: The **shrinkTVP** package must be installed to use this function.

Value

Called for its side effects. Returns `invisible(NULL)`.

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

See Also

[gen_marginal_samples](#)

Other plotting functions: [plot.shrinkGPR\(\)](#), [plot.shrinkGPR_marg_samples_2D\(\)](#), [plot.shrinkTPR\(\)](#)

Examples

```
# Simulate data
set.seed(123)
torch::torch_manual_seed(123)
n <- 100
x <- matrix(runif(n * 2), n, 2)
y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

# Fit GPR model
res <- shrinkGPR(y ~ x1 + x2, data = data)

# Generate marginal samples
marginal_samps_x1 <- gen_marginal_samples(res, to_eval = "x1", nsamp = 100)
marginal_samps_x2 <- gen_marginal_samples(res, to_eval = "x2", nsamp = 100)

# Plot marginal predictions
plot(marginal_samps_x1)
plot(marginal_samps_x2)

# Customize plot appearance (see plot.mcmc.tvp from shrinkTVP package for more options)
plot(marginal_samps_x2, shaded = FALSE, quantlines = TRUE, quantcol = "red")
```

`plot.shrinkGPR_marg_samples_2D`*Plot method for 2D marginal predictions from shrinkGPR*

Description

Generates a 3D surface plot of 2D conditional predictive samples produced by [gen_marginal_samples](#).

Usage

```
## S3 method for class 'shrinkGPR_marg_samples_2D'  
plot(x, ...)
```

Arguments

<code>x</code>	An object of class "shrinkGPR_marg_samples_2D", typically returned by gen_marginal_samples when evaluating two covariates.
<code>...</code>	Additional arguments passed to add_surface for customizing the appearance of the surface plot (e.g., opacity, colorscale, showscale).

Details

The function visualizes the posterior predictive mean across a 2D grid of evaluation points for two covariates. Interactive plotting is handled via the **plotly** package. Axis labels are automatically inferred from the object.

Note: The **plotly** package must be installed to use this function.

Value

A plotly plot object (interactive 3D surface). Can be further customized via pipes and plotly functions.

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

See Also

[gen_marginal_samples](#), [plot_ly](#), [add_surface](#)

Other plotting functions: [plot.shrinkGPR\(\)](#), [plot.shrinkGPR_marg_samples_1D\(\)](#), [plot.shrinkTPR\(\)](#)

Examples

```
# Simulate data
set.seed(123)
torch::torch_manual_seed(123)
n <- 100
x <- matrix(runif(n * 2), n, 2)
y <- sin(2 * pi * x[, 1]) * cos(2 * pi * x[, 2]) + rnorm(n, sd = 0.1)
data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

# Fit GPR model
res <- shrinkGPR(y ~ x1 + x2, data = data)

# Generate marginal predictions for x1 and x2 jointly
marginal_samps_both <- gen_marginal_samples(res, to_eval = c("x1", "x2"), nsamp = 100)

# Plot interactive surface
plot(marginal_samps_both)

# Customize surface appearance
plot(marginal_samps_both, opacity = 0.8, colorscale = "Magma")

# Customize further via plotly
p <- plot(marginal_samps_both, opacity = 0.8, colorscale = "Magma")

p |> plotly::layout(
  scene = list(
    xaxis = list(title = "Covariate 1"),
    yaxis = list(title = "Covariate 2"),
    zaxis = list(title = "Expected value")
  )
)
```

plot.shrinkTPR

Graphical summary of posterior of theta

Description

plot.shrinkTPR generates a boxplot visualizing the posterior distribution of theta obtained from a fitted shrinkTPR object.

Usage

```
## S3 method for class 'shrinkTPR'
plot(x, nsamp = 1000, ...)
```

Arguments

`x` a shrinkTPR object.

`nsamp` a positive integer specifying the number of posterior samples to draw for plotting. The default is 1000.

`...` further arguments passed to the internal `boxplot` function, such as axis labeling or plotting options. By default, `las = 2` is used unless explicitly overridden by the user.

Value

Called for its side effects. Returns `invisible(NULL)`.

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

See Also

Other plotting functions: `plot.shrinkGPR()`, `plot.shrinkGPR_marg_samples_1D()`, `plot.shrinkGPR_marg_samples_2D()`

Examples

```
# Simulate and fit a shrinkTPR model, then plot:
sim <- simGPR()
mod <- shrinkTPR(y ~ ., data = sim$data)
plot(mod)

## Change axis label orientation
plot(mod, las = 1)
```

predict.shrinkGPR	<i>Generate Predictions</i>
-------------------	-----------------------------

Description

`predict.shrinkGPR` generates posterior predictive samples from a fitted `shrinkGPR` model at specified covariates.

Usage

```
## S3 method for class 'shrinkGPR'
predict(object, newdata, nsamp = 100, ...)
```

Arguments

object	A shrinkGPR object representing the fitted Gaussian process regression model.
newdata	<i>Optional</i> data frame containing the covariates for the prediction points. If missing, the training data is used.
nsamp	Positive integer specifying the number of posterior samples to generate. Default is 100.
...	Currently ignored.

Details

This function generates predictions by sampling from the posterior predictive distribution. If the mean equation is included in the model, the corresponding covariates are incorporated.

Value

A matrix containing posterior predictive samples for each covariate combination in newdata.

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)
  # Example usage for in-sample prediction
  preds <- predict(res)

  # Example usage for out-of-sample prediction
  newdata <- data.frame(x1 = runif(10), x2 = runif(10))
  preds <- predict(res, newdata = newdata)
}
```

predict.shrinkTPR	<i>Generate Predictions</i>
-------------------	-----------------------------

Description

predict.shrinkTPR generates posterior predictive samples from a fitted shrinkGPR model at specified covariates.

Usage

```
## S3 method for class 'shrinkTPR'
predict(object, newdata, nsamp = 100, ...)
```

Arguments

object	A shrinkTPR object representing the fitted Gaussian process regression model.
newdata	<i>Optional</i> data frame containing the covariates for the prediction points. If missing, the training data is used.
nsamp	Positive integer specifying the number of posterior samples to generate. Default is 100.
...	Currently ignored.

Details

This function generates predictions by sampling from the posterior predictive distribution. If the mean equation is included in the model, the corresponding covariates are incorporated.

Value

A matrix containing posterior predictive samples for each covariate combination in newdata.

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
  res <- shrinkGPR(y ~ x1 + x2, data = data)
  # Example usage for in-sample prediction
  preds <- predict(res)

  # Example usage for out-of-sample prediction
  newdata <- data.frame(x1 = runif(10), x2 = runif(10))
  preds <- predict(res, newdata = newdata)
}
```

Description

shrinkGPR implements Gaussian process regression (GPR) with a hierarchical shrinkage prior for hyperparameter estimation, incorporating normalizing flows to approximate the posterior distribution. The function facilitates model specification, optimization, and training, including support for early stopping, user-defined kernels, and flow-based transformations.

Usage

```
shrinkGPR(
  formula,
  data,
  a = 0.5,
  c = 0.5,
  formula_mean,
  a_mean = 0.5,
  c_mean = 0.5,
  sigma2_rate = 10,
  kernel_func = kernel_se,
  n_layers = 10,
  n_latent = 10,
  flow_func = sylvester,
  flow_args,
  n_epochs = 1000,
  auto_stop = TRUE,
  cont_model,
  device,
  display_progress = TRUE,
  optim_control
)
```

Arguments

formula	object of class "formula": a symbolic representation of the model for the covariance equation, as in lm . The response variable and covariates are specified here.
data	<i>optional</i> data frame containing the response variable and the covariates. If not found in data, the variables are taken from <code>environment(formula)</code> . No NAs are allowed in the response variable or covariates.
a	positive real number controlling the behavior at the origin of the shrinkage prior for the covariance structure. The default is 0.5.
c	positive real number controlling the tail behavior of the shrinkage prior for the covariance structure. The default is 0.5.

formula_mean	<i>optional</i> formula for the linear mean equation. If provided, the covariates for the mean structure are specified separately from the covariance structure. A response variable is not required in this formula.
a_mean	positive real number controlling the behavior at the origin of the shrinkage for the mean structure. The default is 0.5.
c_mean	positive real number controlling the tail behavior of the shrinkage prior for the mean structure. The default is 0.5.
sigma2_rate	positive real number controlling the prior rate parameter for the residual variance. The default is 10.
kernel_func	function specifying the covariance kernel. The default is kernel_se , a squared exponential kernel. For guidance on how to provide a custom kernel function, see Details.
n_layers	positive integer specifying the number of flow layers in the normalizing flow. The default is 10.
n_latent	positive integer specifying the dimensionality of the latent space for the normalizing flow. The default is 10.
flow_func	function specifying the normalizing flow transformation. The default is sylvester . For guidance on how to provide a custom flow function, see Details.
flow_args	<i>optional</i> named list containing arguments for the flow function. If not provided, default arguments are used. For guidance on how to provide a custom flow function, see Details.
n_epochs	positive integer specifying the number of training epochs. The default is 1000.
auto_stop	logical value indicating whether to enable early stopping based on convergence. The default is TRUE.
cont_model	<i>optional</i> object returned from a previous shrinkGPR call, enabling continuation of training from the saved state.
device	<i>optional</i> device to run the model on, e.g., <code>torch_device("cuda")</code> for GPU or <code>torch_device("cpu")</code> for CPU. Defaults to GPU if available; otherwise, CPU.
display_progress	logical value indicating whether to display progress bars and messages during training. The default is TRUE.
optim_control	<i>optional</i> named list containing optimizer parameters. If not provided, default settings are used.

Details

This implementation provides a computationally efficient framework for GPR, enabling flexible modeling of mean and covariance structures. Users can specify custom kernel functions, flow transformations, and hyperparameter configurations to adapt the model to their data.

The shrinkGPR function combines Gaussian process regression with shrinkage priors and normalizing flows for efficient and flexible hyperparameter estimation. It supports custom kernels, hierarchical shrinkage priors for mean and covariance structures, and flow-based posterior approximations. The `auto_stop` option allows early stopping based on lack of improvement in ELBO.

Custom Kernel Functions

Users can define custom kernel functions for the covariance structure of the Gaussian process by passing them to the `kernel_func` argument. A valid kernel function must follow the same structure as the provided `kernel_se` (squared exponential kernel). The function should:

1. Accept the following arguments:

- `thetas`: A `torch_tensor` of dimensions $n_{\text{latent}} \times d$, representing latent length-scale parameters.
- `tau`: A `torch_tensor` of length n_{latent} , representing latent scaling factors.
- `x`: A `torch_tensor` of dimensions $N \times d$, containing the input data points.
- `x_star`: Either `NULL` or a `torch_tensor` of dimensions $N_{\text{new}} \times d$. If `NULL`, the kernel is computed for `x` against itself. Otherwise, it computes the kernel between `x` and `x_star`.

2. Return:

- If `x_star = NULL`, the function must return a `torch_tensor` of dimensions $n_{\text{latent}} \times N \times N$, representing pairwise covariances between all points in `x`.
- If `x_star` is provided, the function must return a `torch_tensor` of dimensions $n_{\text{latent}} \times N_{\text{new}} \times N$, representing pairwise covariances between `x_star` and `x`.

3. Requirements:

- The kernel must compute a valid positive semi-definite covariance matrix.
- It should use efficient tensor operations from the Torch library (e.g., `torch_bmm`, `torch_sum`) to ensure compatibility with GPUs or CPUs.

Testing a Custom Kernel Function

To test a custom kernel function:

1. Verify Dimensions:

- When `x_star = NULL`, ensure the output is $n_{\text{latent}} \times N \times N$.
- When `x_star` is provided, ensure the output is $n_{\text{latent}} \times N_{\text{new}} \times N$.

2. Check Positive Semi-Definiteness: Validate that the kernel produces a positive semi-definite covariance matrix for valid inputs.

3. Integrate: Use the custom kernel with `shrinkGPR` to confirm its compatibility.

Examples of kernel functions can be found in the `kernel_funcs.R` file in the package source code, which are documented in the [kernel_functions](#) help file.

Custom Flow Functions

Users can define custom flow functions for use in Gaussian process regression models by following the structure and conventions of the provided `sylvester` function. A valid flow function should be implemented as a `nn_module` in `torch` and must meet the following requirements:

Structure of a Custom Flow Function

1. Initialization (initialize):

- Include all required parameters as `nn_parameter` or `nn_buffer`, and initialize them appropriately.
- Parameters may include matrices for transformations (e.g., triangular matrices), biases, or other learnable components.

2. Forward Pass (forward):

- The forward method should accept an input tensor z of dimensions $n_{\text{latent}} \times D$.
- The method must:
 - Compute the transformed tensor z .
 - Compute the log determinant of the Jacobian ($\log|\det J|$).
- The method should return a list containing:
 - z_k : The transformed samples after applying the flow ($n_{\text{latent}} \times D$).
 - $\log_diag_j$: A tensor of size n_{latent} containing the log determinant of the Jacobian for each sample.

3. Output Dimensions:

- Input tensor z : $n_{\text{latent}} \times D$.
- Outputs:
 - z_k : $n_{\text{latent}} \times D$.
 - $\log_diag_j$: n_{latent} .

An example of a flow function can be found in the `sylvester.R` file in the package source code, which is documented in the [sylvester](#) help file.

Value

A list object of class `shrinkGPR`, containing:

<code>model</code>	The best-performing trained model.
<code>loss</code>	The best loss value (ELBO) achieved during training.
<code>loss_stor</code>	A numeric vector storing the ELBO values at each training iteration.
<code>last_model</code>	The model state at the final iteration.
<code>optimizer</code>	The optimizer object used during training.
<code>model_internals</code>	Internal objects required for predictions and further training, such as model matrices and formulas.

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit GPR model
```

```

res <- shrinkGPR(y ~ x1 + x2, data = data)

# Check convergence
plot(res$loss_stor, type = "l", main = "Loss Over Iterations")

# Check posterior
samps <- gen_posterior_samples(res, nsamp = 1000)
boxplot(samps$thetas) # Second theta is pulled towards zero

# Predict
x1_new <- seq(from = 0, to = 1, length.out = 100)
x2_new <- runif(100)
y_new <- predict(res, newdata = data.frame(x1 = x1_new, x2 = x2_new), nsamp = 2000)

# Plot
quants <- apply(y_new, 2, quantile, c(0.025, 0.5, 0.975))
plot(x1_new, quants[2, ], type = "l", ylim = c(-1.5, 1.5),
     xlab = "x1", ylab = "y", lwd = 2)
polygon(c(x1_new, rev(x1_new)), c(quants[1, ], rev(quants[3, ])),
       col = adjustcolor("skyblue", alpha.f = 0.5), border = NA)
points(x[,1], y)
curve(sin(2 * pi * x), add = TRUE, col = "forestgreen", lwd = 2, lty = 2)

# Add mean equation
res2 <- shrinkGPR(y ~ x1 + x2, formula_mean = ~ x1, data = data)
}

```

shrinkTPR

Student-t Process Regression with Shrinkage and Normalizing Flows

Description

shrinkTPR implements Student-t process regression (TPR), as described in Shah et al. (2014), with a hierarchical shrinkage prior for hyperparameter estimation, incorporating normalizing flows to approximate the posterior distribution. The function facilitates model specification, optimization, and training, including support for early stopping, user-defined kernels, and flow-based transformations.

Usage

```

shrinkTPR(
  formula,
  data,
  a = 0.5,
  c = 0.5,
  formula_mean,
  a_mean = 0.5,

```

```

    c_mean = 0.5,
    sigma2_rate = 10,
    nu_alpha = 0.5,
    nu_beta = 2,
    kernel_func = kernel_se,
    n_layers = 10,
    n_latent = 10,
    flow_func = sylvester,
    flow_args,
    n_epochs = 1000,
    auto_stop = TRUE,
    cont_model,
    device,
    display_progress = TRUE,
    optim_control
)

```

Arguments

formula	object of class "formula": a symbolic representation of the model for the covariance equation, as in lm . The response variable and covariates are specified here.
data	<i>optional</i> data frame containing the response variable and the covariates. If not found in data, the variables are taken from <code>environment(formula)</code> . No NAs are allowed in the response variable or covariates.
a	positive real number controlling the behavior at the origin of the shrinkage prior for the covariance structure. The default is 0.5.
c	positive real number controlling the tail behavior of the shrinkage prior for the covariance structure. The default is 0.5.
formula_mean	<i>optional</i> formula for the linear mean equation. If provided, the covariates for the mean structure are specified separately from the covariance structure. A response variable is not required in this formula.
a_mean	positive real number controlling the behavior at the origin of the shrinkage for the mean structure. The default is 0.5.
c_mean	positive real number controlling the tail behavior of the shrinkage prior for the mean structure. The default is 0.5.
sigma2_rate	positive real number controlling the prior rate parameter for the residual variance. The default is 10.
nu_alpha	positive real number controlling the shape parameter of the shifted gamma prior for the degrees of freedom of the Student-t process. The default is 0.5.
nu_beta	positive real number controlling the scale parameter of the shifted gamma prior for the degrees of freedom of the Student-t process. The default is 2.
kernel_func	function specifying the covariance kernel. The default is kernel_se , a squared exponential kernel. For guidance on how to provide a custom kernel function, see Details .

<code>n_layers</code>	positive integer specifying the number of flow layers in the normalizing flow. The default is 10.
<code>n_latent</code>	positive integer specifying the dimensionality of the latent space for the normalizing flow. The default is 10.
<code>flow_func</code>	function specifying the normalizing flow transformation. The default is sylvester . For guidance on how to provide a custom flow function, see Details.
<code>flow_args</code>	<i>optional</i> named list containing arguments for the flow function. If not provided, default arguments are used. For guidance on how to provide a custom flow function, see Details.
<code>n_epochs</code>	positive integer specifying the number of training epochs. The default is 1000.
<code>auto_stop</code>	logical value indicating whether to enable early stopping based on convergence. The default is TRUE.
<code>cont_model</code>	<i>optional</i> object returned from a previous shrinkTPR call, enabling continuation of training from the saved state.
<code>device</code>	<i>optional</i> device to run the model on, e.g., <code>torch_device("cuda")</code> for GPU or <code>torch_device("cpu")</code> for CPU. Defaults to GPU if available; otherwise, CPU.
<code>display_progress</code>	logical value indicating whether to display progress bars and messages during training. The default is TRUE.
<code>optim_control</code>	<i>optional</i> named list containing optimizer parameters. If not provided, default settings are used.

Details

This implementation provides a computationally efficient framework for TPR, enabling flexible modeling of mean and covariance structures. Users can specify custom kernel functions, flow transformations, and hyperparameter configurations to adapt the model to their data.

The shrinkTPR function combines Student-t process regression with shrinkage priors and normalizing flows for efficient and flexible hyperparameter estimation. It supports custom kernels, hierarchical shrinkage priors for mean and covariance structures, and flow-based posterior approximations. The `auto_stop` option allows early stopping based on lack of improvement in ELBO.

Custom Kernel Functions

Users can define custom kernel functions for the covariance structure of the Gaussian process by passing them to the `kernel_func` argument. A valid kernel function must follow the same structure as the provided `kernel_se` (squared exponential kernel). The function should:

1. Accept the following arguments:

- `thetas`: A `torch_tensor` of dimensions $n_{\text{latent}} \times d$, representing latent length-scale parameters.
- `tau`: A `torch_tensor` of length n_{latent} , representing latent scaling factors.
- `x`: A `torch_tensor` of dimensions $N \times d$, containing the input data points.
- `x_star`: Either `NULL` or a `torch_tensor` of dimensions $N_{\text{new}} \times d$. If `NULL`, the kernel is computed for `x` against itself. Otherwise, it computes the kernel between `x` and `x_star`.

2. Return:

- If `x_star = NULL`, the function must return a `torch_tensor` of dimensions `n_latent x N`, representing pairwise covariances between all points in `x`.
- If `x_star` is provided, the function must return a `torch_tensor` of dimensions `n_latent x N_new x N`, representing pairwise covariances between `x_star` and `x`.

3. Requirements:

- The kernel must compute a valid positive semi-definite covariance matrix.
- It should use efficient tensor operations from the Torch library (e.g., `torch_bmm`, `torch_sum`) to ensure compatibility with GPUs or CPUs.

Testing a Custom Kernel Function

To test a custom kernel function:

1. Verify Dimensions:

- When `x_star = NULL`, ensure the output is `n_latent x N x N`.
- When `x_star` is provided, ensure the output is `n_latent x N_new x N`.

2. Check Positive Semi-Definiteness:

Validate that the kernel produces a positive semi-definite covariance matrix for valid inputs.

3. Integrate:

Use the custom kernel with `shrinkTPR` to confirm its compatibility.

Examples of kernel functions can be found in the `kernel_funcs.R` file in the package source code, which are documented in the [kernel_functions](#) help file.

Custom Flow Functions

Users can define custom flow functions for use in Gaussian process regression models by following the structure and conventions of the provided `sylvester` function. A valid flow function should be implemented as a `nn_module` in torch and must meet the following requirements:

Structure of a Custom Flow Function

1. Initialization (initialize):

- Include all required parameters as `nn_parameter` or `nn_buffer`, and initialize them appropriately.
- Parameters may include matrices for transformations (e.g., triangular matrices), biases, or other learnable components.

2. Forward Pass (forward):

- The forward method should accept an input tensor `z` of dimensions `n_latent x D`.
- The method must:
 - Compute the transformed tensor `z`.
 - Compute the log determinant of the Jacobian ($\log|\det J|$).
- The method should return a list containing:
 - `zk`: The transformed samples after applying the flow (`n_latent x D`).
 - `log_diag_j`: A tensor of size `n_latent` containing the log determinant of the Jacobian for each sample.

3. Output Dimensions:

- Input tensor `z`: `n_latent x D`.
- Outputs:

- zk: $n_{\text{latent}} \times D$.
- log_diag_j: n_{latent} .

An example of a flow function can be found in the `sylvester.R` file in the package source code, which is documented in the [sylvester](#) help file.

Value

A list object of class `shrinkTPR`, containing:

<code>model</code>	The best-performing trained model.
<code>loss</code>	The best loss value (ELBO) achieved during training.
<code>loss_stor</code>	A numeric vector storing the ELBO values at each training iteration.
<code>last_model</code>	The model state at the final iteration.
<code>optimizer</code>	The optimizer object used during training.
<code>model_internals</code>	Internal objects required for predictions and further training, such as model matrices and formulas.

Author(s)

Peter Knaus <peter.knaus@wu.ac.at>

References

Shah, A., Wilson, A., & Ghahramani, Z. (2014, April). Student-t processes as alternatives to Gaussian processes. In *Artificial intelligence and statistics* (pp. 877-885). PMLR.

Examples

```
if (torch::torch_is_installed()) {
  # Simulate data
  set.seed(123)
  torch::torch_manual_seed(123)
  n <- 100
  x <- matrix(runif(n * 2), n, 2)
  y <- sin(2 * pi * x[, 1]) + rnorm(n, sd = 0.1)
  data <- data.frame(y = y, x1 = x[, 1], x2 = x[, 2])

  # Fit TPR model
  res <- shrinkTPR(y ~ x1 + x2, data = data)

  # Check convergence
  plot(res$loss_stor, type = "l", main = "Loss Over Iterations")

  # Check posterior
  samps <- gen_posterior_samples(res, nsamp = 1000)
  boxplot(samps$thetas) # Second theta is pulled towards zero

  # Predict
```

```

x1_new <- seq(from = 0, to = 1, length.out = 100)
x2_new <- runif(100)
y_new <- predict(res, newdata = data.frame(x1 = x1_new, x2 = x2_new), nsamp = 2000)

# Plot
quants <- apply(y_new, 2, quantile, c(0.025, 0.5, 0.975))
plot(x1_new, quants[2, ], type = "l", ylim = c(-1.5, 1.5),
     xlab = "x1", ylab = "y", lwd = 2)
polygon(c(x1_new, rev(x1_new)), c(quants[1, ], rev(quants[3, ])),
       col = adjustcolor("skyblue", alpha.f = 0.5), border = NA)
points(x[,1], y)
curve(sin(2 * pi * x), add = TRUE, col = "forestgreen", lwd = 2, lty = 2)

# Add mean equation
res2 <- shrinkTPR(y ~ x1 + x2, formula_mean = ~ x1, data = data)
}

```

simGPR

Simulate Data for Gaussian Process Regression

Description

simGPR generates simulated data for Gaussian Process Regression (GPR) models, including the true hyperparameters used for simulation.

Usage

```

simGPR(
  N = 200,
  d = 3,
  d_mean = 0,
  sigma2 = 0.1,
  tau = 2,
  kernel_func = kernel_se,
  perc_spars = 0.5,
  rho = 0,
  theta,
  beta,
  device
)

```

Arguments

N	Positive integer specifying the number of observations to simulate. Default is 200.
---	---

d	Positive integer specifying the number of covariates for the covariance structure. Default is 3.
d_mean	Positive integer specifying the number of covariates for the mean structure. Default is 0.
sigma2	Positive numeric value specifying the noise variance. Default is 0.1.
tau	Positive numeric value specifying the global shrinkage parameter. Default is 2.
kernel_func	Function specifying the covariance kernel. Default is kernel_se.
perc_spars	Numeric value in [0, 1] indicating the proportion of elements in theta and beta to sparsify. Default is 0.5.
rho	Numeric value in [0, 1] indicating the correlation between the covariates. Default is 0.
theta	<i>Optional</i> numeric vector specifying the true inverse length-scale parameters. If not provided, they are randomly generated.
beta	<i>Optional</i> numeric vector specifying the true regression coefficients for the mean structure. If not provided, they are randomly generated.
device	<i>Optional</i> torch_device object specifying whether to run the simulation on CPU or GPU. Defaults to GPU if available.

Details

This function simulates data from a Gaussian Process Regression model. The response variable y is sampled from a multivariate normal distribution with a covariance matrix determined by the specified kernel function, θ , τ , and σ^2 . If $d_mean > 0$, a mean structure is included in the simulation, with covariates x_mean and regression coefficients β .

Value

A list containing:

- **data**: A data frame with y (response variable), x (covariates for the covariance structure), and optionally x_mean (covariates for the mean structure).
- **true_vals**: A list containing the true values used for the simulation:
 - **theta**: The true inverse length-scale parameters.
 - **sigma2**: The true noise variance.
 - **tau**: The true global shrinkage parameter.
 - **beta** (optional): The true regression coefficients for the mean structure.

Examples

```
if (torch::torch_is_installed()) {
  torch::torch_manual_seed(123)

  # Simulate data with default settings
  sim_data <- simGPR()

  # Simulate data with custom settings
```

```

sim_data <- simGPR(N = 100, d = 5, d_mean = 2, perc_spars = 0.3, sigma2 = 0.5)

# Access the simulated data
head(sim_data$data)

# Access the true values used for simulation
sim_data$true_vals
}

```

sylvester

*Sylvester Normalizing Flow***Description**

The `sylvester` function implements Sylvester normalizing flows as described by van den Berg et al. (2018) in "Sylvester Normalizing Flows for Variational Inference." This flow applies a sequence of invertible transformations to map a simple base distribution to a more complex target distribution, allowing for flexible posterior approximations in Gaussian process regression models.

Usage

```
sylvester(d, n_householder)
```

Arguments

<code>d</code>	An integer specifying the latent dimensionality of the input space.
<code>n_householder</code>	An optional integer specifying the number of Householder reflections used to orthogonalize the transformation. Defaults to <code>d - 1</code> .

Details

The Sylvester flow uses two triangular matrices (R_1 and R_2) and Householder reflections to construct invertible transformations. The transformation is parameterized as follows:

$$z = QR_1h(Q^T R_2 z k + b) + z k,$$

where:

- Q is an orthogonal matrix obtained via Householder reflections.
- R_1 and R_2 are upper triangular matrices with learned diagonal elements.
- h is a non-linear activation function (default: `torch_tanh`).
- b is a learned bias vector.

The log determinant of the Jacobian is computed to ensure the invertibility of the transformation and is given by:

$$\log |\det J| = \sum_{i=1}^d \log |\text{diag}_1[i] \cdot \text{diag}_2[i] \cdot h'(RQ^T z k + b) + 1|,$$

where `diag_1` and `diag_2` are the learned diagonal elements of R_1 and R_2 , respectively, and h' is the derivative of the activation function.

Value

An `nn_module` object representing the Sylvester normalizing flow. The module has the following key components:

- `forward(zk)`: The forward pass computes the transformed variable `z` and the log determinant of the Jacobian.
- Internal parameters include matrices `R1` and `R2`, diagonal elements, and Householder reflections used for orthogonalization.

References

van den Berg, R., Hasenclever, L., Tomczak, J. M., & Welling, M. (2018). "Sylvester Normalizing Flows for Variational Inference." *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI 2018)*.

Examples

```
if (torch::torch_is_installed()) {  
  # Example: Initialize a Sylvester flow  
  d <- 5  
  n_householder <- 4  
  flow <- sylvester(d, n_householder)  
  
  # Forward pass through the flow  
  zk <- torch::torch_randn(10, d) # Batch of 10 samples  
  result <- flow(zk)  
  
  print(result$zk)  
  print(result$log_diag_j)  
}
```

Index

* plotting functions

- plot.shrinkGPR, [10](#)
- plot.shrinkGPR_marg_samples_1D, [11](#)
- plot.shrinkGPR_marg_samples_2D, [13](#)
- plot.shrinkTPR, [14](#)

add_surface, [13](#)

boxplot, [10](#), [15](#)

calc_pred_moments, [2](#), [5](#)

eval_pred_dens, [3](#)

gen_marginal_samples, [4](#), [11–13](#)

gen_posterior_samples, [6](#)

kernel_functions, [7](#), [20](#), [25](#)

kernel_matern_12 (kernel_functions), [7](#)

kernel_matern_32 (kernel_functions), [7](#)

kernel_matern_52 (kernel_functions), [7](#)

kernel_se, [19](#), [23](#)

kernel_se (kernel_functions), [7](#)

lm, [18](#), [23](#)

LPDS, [9](#)

plot.mcmc.tvp, [11](#)

plot.shrinkGPR, [10](#), [12](#), [13](#), [15](#)

plot.shrinkGPR_marg_samples_1D, [11](#), [11](#),
[13](#), [15](#)

plot.shrinkGPR_marg_samples_2D, [11](#), [12](#),
[13](#), [15](#)

plot.shrinkTPR, [11–13](#), [14](#)

plot_ly, [13](#)

predict.shrinkGPR, [15](#)

predict.shrinkTPR, [16](#)

shrinkGPR, [18](#)

shrinkTPR, [22](#)

simGPR, [27](#)

sylvester, [19](#), [21](#), [24](#), [26](#), [29](#)