

Package: gaussianprocesses (via r-universe)

October 22, 2026

Type Package

Title Gaussian Processes from First Principles

Version 1.0.0.9000

Description Gaussian process regression, classification, and count models with explicit covariance kernels, exact Gaussian inference, Laplace approximations for non-Gaussian likelihoods, sparse approximations, marginal-likelihood optimization, derivatives, posterior sampling, calibration diagnostics, and time-series workflows. The implementation follows the core methodology of Rasmussen and Williams (2006) [doi:10.7551/mitpress/3206.001.0001](https://doi.org/10.7551/mitpress/3206.001.0001) and emphasizes numerical stability, transparent assumptions, and reproducible statistical workflows.

License MIT + file LICENSE

URL <https://gaussian-processes-r-12efed.gitlab.io/>,
<https://gitlab.com/DiogoRibeiro7/gaussian-processes-r>

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports stats

Suggests DiceKriging, gplite, kernlab, knitr, nlme, rmarkdown,
testthat (>= 3.0.0), yaml

VignetteBuilder knitr

Config/testthat/edition 3

Repository <https://diogoribeiro7.r-universe.dev>

Date/Publication 2026-10-22 08:15:07 UTC

RemoteUrl <https://gitlab.com/diogoribeiro7/gaussian-processes-r>

RemoteRef HEAD

RemoteSha 8770c93ab0d47ef341c31442cad45f565f6403c5

Contents

benchmark_exact_gp_scaling	3
benchmark_sparse_gp	4
changepoint_kernel	6
coef.gaussianprocesses_latent_model	7
compare_gp_predictions	8
coregionalization_kernel	9
coregionalization_matrix	11
evaluate_kernel	12
evaluate_likelihood	13
evaluate_mean	14
fit_gp	15
fit_heteroscedastic_gp	17
fit_latent_gp	19
fit_sparse_gp	22
fit_time_series_gp	25
forecast_gp	27
forecast_metrics	28
gaussianprocesses_model	29
gp_calibration_diagnostics	31
gp_classification_scores	32
gp_count_scores	34
gp_diagnostics	36
gp_hyperparameter_uncertainty	37
gp_likelihoods	39
gp_mean_functions	41
gp_model_methods	42
gp_numerical_diagnostics	45
gp_profile_likelihood	47
gp_scores	48
gp_simulation_scenario	51
gp_stack_outputs	52
gp_time_index	53
initialize_coregionalization	54
initialize_spectral_mixture	55
kernel_diagonal	56
kernel_gradient	57
kernel_input_gradient	58
kernel_linear	59
kernel_matern	60
kernel_parameters	61
kernel_periodic	62
kernel_rational_quadratic	63
kernel_rbf	65
kernel_white_noise	66
likelihood_predictive	67
linear_kernel	68

log_marginal_likelihood	69
log_marginal_likelihood_gradient	70
loo_gp	71
matern_kernel_specs	72
mean_coefficients	73
optimize_gp	75
optimize_latent_gp	78
optimize_sparse_gp	81
optimize_time_series_gp	84
parametric_means	85
periodic_kernel	87
predict_gp	88
predict_gradient_gp	90
predict_heteroscedastic_gp	92
predict_latent_gp	93
predict_sparse_gp	95
product_kernel	97
rational_quadratic_kernel	98
rbf_kernel	99
rolling_origin_gp	100
run_gp_benchmark_suite	101
sample_gp_posterior.gaussianprocesses_latent_model	102
sample_gp_prior	106
scale_kernel	108
select_dimensions	109
select_inducing_points	110
simulate_gp_data	111
spectral_mixture_kernel	113
sum_kernel	114
time_series_kernel	115
update_kernel_parameters	117
white_noise_kernel	118

Index**119**

benchmark_exact_gp_scaling

*Benchmark exact-GP scaling***Description**

Measures exact GP fit time and retained model size over increasing training sizes using seeded RBF simulations. This function is intended for local empirical scaling studies rather than CI performance gating.

Usage

```
benchmark_exact_gp_scaling(
  sizes = c(50L, 100L, 200L, 400L),
  repeats = 3L,
  seed = 1729L
)
```

Arguments

sizes	Positive integer vector of training sizes.
repeats	Number of timing repetitions per size.
seed	Base non-negative integer seed.

Value

A data frame with one row per requested size.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
# A deliberately small run; larger sizes show the cubic cost.
benchmark_exact_gp_scaling(sizes = c(20, 40), repeats = 1)
```

benchmark_sparse_gp	<i>Benchmark sparse FITC against exact GP regression</i>
---------------------	--

Description

Runs both models with identical fixed kernel and noise parameters, then compares retained model size, elapsed fitting/prediction time, posterior mean RMSE, and latent-variance MAE. The exact GP is treated as the numerical reference, not as ground truth.

Usage

```
benchmark_sparse_gp(
  x,
  y,
  kernel,
  n_inducing,
  prediction_x = x,
  noise_variance = 1e-06,
  mean = zero_mean(),
  selection = c("farthest", "quantile", "variance"),
  observation_noise_variance = NULL
)
```

Arguments

<code>x</code>	Numeric vector or matrix of training inputs.
<code>y</code>	Numeric response vector.
<code>kernel</code>	Fixed kernel specification shared by both models.
<code>n_inducing</code>	Number of inducing points for FITC.
<code>prediction_x</code>	Optional prediction inputs; defaults to <code>x</code> .
<code>noise_variance</code>	Non-negative observation-noise variance: a scalar, or one value per training observation. Both models use the same noise.
<code>mean</code>	Mean specification shared by both models.
<code>selection</code>	Inducing-point selection method.
<code>observation_noise_variance</code>	Optional noise variance at <code>prediction_x</code> , passed to both predictions. It is required when <code>noise_variance</code> has one value per observation, as in <code>predict_gp()</code> .

Details

The benchmark reports retained R object size, not peak process memory. Elapsed timings are simple wall-clock measurements intended for local comparison, not publication-grade benchmarking.

Value

A list containing a comparison data frame and both prediction objects.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-3, 3, length.out = 200)
y <- sin(x) + 0.1 * cos(7 * x)

benchmark <- benchmark_sparse_gp(
  x,
  y,
  kernel = rbf_kernel(length_scale = 0.8),
  n_inducing = 15,
  noise_variance = 0.01
)
benchmark$comparison
```

changepoint_kernel	<i>Changepoint kernels</i>
--------------------	----------------------------

Description

Switches from one covariance to another along one input column, smoothly around an estimable location: a structural break in a time series, or a regime change in space.

Usage

```
changepoint_kernel(before, after, location = 0, steepness = 1, column = 1L)
```

Arguments

before, after	Kernel specifications that hold before and after the change.
location	Finite location c of the change, in the units of the input column.
steepness	Positive steepness a of the transition, in inverse input units: the transition takes about $4/a$ from 12% to 88%.
column	The input column along which the change happens.

Details

With the sigmoid $s(x) = 1/(1 + e^{-a(x_t - c)})$ along column t ,

$$k(x, x') = (1 - s(x))k_1(x, x')(1 - s(x')) + s(x)k_2(x, x')s(x'),$$

where k_1 is before and k_2 is after. Each term has the form $g(x)k(x, x')g(x')$, so the kernel is positive semidefinite. The two regimes are independent processes: as $a \rightarrow \infty$, the covariance between points on opposite sides of c tends to 0.

location is a real-valued parameter, optimized on the identity scale, and steepness a positive one, optimized on the log scale. Parameter paths are location and steepness for the change itself, and before. and after. for the two kernels, so nested changepoints model several changes. [optimize_gp\(\)](#) bounds every changepoint location by the training range of its column and, because the likelihood is often multimodal in the location, starts it at evenly spaced quantiles of that column.

The kernel is as smooth as its two components, so it supports [kernel_input_gradient\(\)](#), [predict_gradient_gp\(\)](#), and derivative observations when they do.

Value

A composite kernel specification.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[time_series_kernel\(\)](#), whose changepoint argument adds a change to the time-series kernel.

Examples

```
kernel <- changepoint_kernel(
  before = rbf_kernel(variance = 1, length_scale = 2),
  after = rbf_kernel(variance = 0.2, length_scale = 0.5),
  location = 5,
  steepness = 4
)
kernel
names(kernel_parameters(kernel, flatten = TRUE))

# Points on opposite sides of a steep change are nearly uncorrelated.
evaluate_kernel(kernel, c(4, 6))
```

```
coef.gaussianprocesses_latent_model
```

Coefficients of a Gaussian-process mean

Description

coef() returns the coefficients β of a model's mean function, and vcov() their covariance given the hyperparameters.

Usage

```
## S3 method for class 'gaussianprocesses_latent_model'
coef(object, ...)

## S3 method for class 'gaussianprocesses_model'
coef(object, ...)

## S3 method for class 'gaussianprocesses_model'
vcov(object, ...)
```

Arguments

object	A fitted gaussianprocesses_model or, for coef(), gaussianprocesses_latent_model.
...	Unused.

Details

For estimated coefficients, coef() is the generalized least-squares estimate $\hat{\beta}$ and vcov() its covariance $(H^T C^{-1} H)^{-1}$. For marginalized coefficients they are the posterior mean $\bar{\beta}$ and covariance $(B^{-1} + H^T C^{-1} H)^{-1}$. Fixed coefficients have zero covariance. Both treat the kernel and noise parameters as known. The zero mean has no coefficients.

For latent models (`fit_latent_gp()`), `coef()` returns the fixed coefficients, the estimate that maximizes the approximate marginal likelihood, or, under a Gaussian prior $N(b, B)$, the approximate posterior mean $b + BH^\top a$ with $a = \nabla \log p(y \mid \hat{f})$. They have no `vcov()` method.

Value

`coef()`: a named numeric vector. `vcov()`: a matrix with matching row and column names.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[mean_coefficients](#), [linear_mean\(\)](#).

Examples

```
x <- seq(0, 4, length.out = 25)
model <- fit_gp(x, 1 + 0.5 * x + sin(2 * x), rbf_kernel(length_scale = 0.5),
  noise_variance = 0.01, mean = linear_mean())

coef(model)
vcov(model)
```

compare_gp_predictions

Compare two GP posterior predictions numerically

Description

Compare two GP posterior predictions numerically

Usage

```
compare_gp_predictions(reference, candidate)
```

Arguments

reference	Reference prediction object with mean and latent_variance fields.
candidate	Candidate prediction object with matching lengths.

Value

A named list of numerical accuracy differences.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-3, 3, length.out = 100)
y <- sin(x)
kernel <- rbf_kernel(length_scale = 0.8)

exact <- predict_gp(fit_gp(x, y, kernel, noise_variance = 0.01), x)
sparse <- predict_sparse_gp(
  fit_sparse_gp(x, y, kernel, noise_variance = 0.01, n_inducing = 10),
  x
)

compare_gp_predictions(exact, sparse)
```

coregionalization_kernel

Create a coregionalization kernel for several outputs

Description

A coregionalization kernel acts on one input column that holds an output index $p \in \{1, \dots, P\}$ and gives the covariance between outputs, $k(p, q) = B_{pq}$ with

$$B = WW^\top + \text{diag}(\kappa),$$

which is positive semidefinite by construction. Multiplied by a kernel on the other inputs it gives the intrinsic coregionalization model (ICM), $\text{cov}(f_p(x), f_q(x')) = B_{pq} k(x, x')$; a sum of such products is the linear model of coregionalization (LMC).

Usage

```
coregionalization_kernel(n_outputs, rank = 1L, W = NULL, kappa = NULL)
```

Arguments

n_outputs	Number of outputs P .
rank	Number of columns R of W : the rank of the shared part of B .
W	$P \times R$ matrix, or a vector of its PR values in column-major order. The default, zero, makes the outputs independent.
kappa	Positive output-specific variances, one per output; the default is 1.

Details

The kernel needs exactly one input column, so in a model with other inputs it is restricted to the output-index column with `select_dimensions()`. With inputs in column 1 and the output index in column 2, an ICM kernel is `product_kernel(select_dimensions(rbf_kernel(), 1), select_dimensions(coregionalization_kernel(P), 2))`. `gp_stack_outputs()` builds such inputs from one response column per output, and `initialize_coregionalization()` gives a starting W and κ .

The optimizer works on W directly and on $\log \kappa$. Two cautions apply:

- W is identified only up to rotation and sign: WQ gives the same B for any orthogonal Q . Report B , with `coregionalization_matrix()`, not W .
- $W = 0$ is a stationary point of the likelihood, where the gradient with respect to W is zero, so an optimization started there keeps $W = 0$; start from `initialize_coregionalization()` instead.

The overall scale of an ICM term is shared by B and the variance of the input kernel; fixing the latter at 1 removes the redundancy only if that parameter is not optimized, so expect the two to trade off.

Value

A `gaussianprocesses_kernel`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

- Bonilla, E. V., Chai, K. M. A., and Williams, C. K. I. (2008). Multi-task Gaussian process prediction. *Advances in Neural Information Processing Systems*, 20.
- Alvarez, M. A., Rosasco, L., and Lawrence, N. D. (2012). Kernels for vector-valued functions: a review. *Foundations and Trends in Machine Learning*, 4(3), 195–266.

See Also

[gp_stack_outputs\(\)](#), [initialize_coregionalization\(\)](#), [coregionalization_matrix\(\)](#).

Examples

```
kernel <- coregionalization_kernel(
  n_outputs = 2,
  rank = 1,
  W = c(1, 0.5),
  kappa = c(0.1, 0.2)
)
coregionalization_matrix(kernel)
evaluate_kernel(kernel, matrix(c(1, 2, 2)))
```

```
# An ICM kernel for inputs in column 1 and the output index in column 2.
icm <- product_kernel(
  select_dimensions(rbf_kernel(length_scale = 0.5), 1),
  select_dimensions(kernel, 2)
)
```

codegionalization_matrix

Output covariance matrices of coregionalization kernels

Description

Returns $B = WW^T + \text{diag}(\kappa)$ for every coregionalization kernel in a kernel, such as the fitted kernel of a multi-output model.

Usage

```
coregionalization_matrix(kernel)
```

Arguments

kernel A kernel specification.

Details

B is identified, while W is not: WQ gives the same B for any orthogonal Q . In an ICM term $B \otimes k$, the scale of B also trades off with the variance of the input kernel k ; the correlations `stats::cov2cor(B)` do not.

Value

A named list with one $P \times P$ matrix per coregionalization kernel, named by its parameter path ("" for a coregionalization kernel itself).

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[coregionalization_kernel\(\)](#).

Examples

```
kernel <- sum_kernel(
  product_kernel(
    select_dimensions(rbf_kernel(), 1),
    select_dimensions(coregionalization_kernel(2, W = c(1, 1)), 2)
  ),
  select_dimensions(coregionalization_kernel(2, kappa = c(0.5, 2)), 2)
)
coregionalization_matrix(kernel)
```

evaluate_kernel	<i>Evaluate a kernel specification</i>
-----------------	--

Description

The package offers two ways to compute a covariance matrix. Kernel specifications, such as [rbf_kernel\(\)](#), are objects: they can be combined with [sum_kernel\(\)](#), [product_kernel\(\)](#), and [scale_kernel\(\)](#), their parameters can be read, updated, and optimized, and models store them. [evaluate_kernel\(\)](#) computes their covariance. The covariance functions, such as [kernel_rbf\(\)](#), take the parameter values as arguments and return one matrix directly. Both use the same formulas.

Usage

```
evaluate_kernel(kernel, x, y = NULL)
```

Arguments

kernel	A Gaussian-process kernel specification.
x	Numeric vector or matrix of observations.
y	Optional numeric vector or matrix. If NULL, evaluates the covariance of x with itself.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- sum_kernel(rbf_kernel(length_scale = 0.5), white_noise_kernel(0.1))
x <- c(0, 0.5, 1)

evaluate_kernel(kernel, x)

# Cross-covariance between training and new inputs.
evaluate_kernel(kernel, x, c(0.25, 2))
```

evaluate_likelihood	<i>Evaluate a likelihood and its derivatives</i>
---------------------	--

Description

Computes $\log p(y_i | f_i)$ and its first three derivatives with respect to the latent values f_i , the quantities that Laplace and related approximations need.

Usage

```
evaluate_likelihood(likelihood, y, f, exposure = NULL)
```

Arguments

likelihood	A likelihood specification (gp_likelihooods).
y	Responses, as the likelihood accepts them.
f	Numeric vector of latent values, one per response.
exposure	Optional positive exposure for Poisson likelihoods: one value or one per response.

Value

A data frame with one row per response: y in the internal coding (0/1 for binary responses), f, log_density, and the derivatives first, second, and third.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
f <- c(-40, -2, 0, 2, 40)
evaluate_likelihood(bernoulli_likelihood("probit"), rep(1, 5), f)
evaluate_likelihood(poisson_likelihood(), c(0, 1, 2, 3, 4), f = log(1:5),
  exposure = 2)
```

evaluate_mean	<i>Evaluate a Gaussian-process mean function</i>
---------------	--

Description

Evaluate a Gaussian-process mean function

Usage

```
evaluate_mean(mean, x)
```

Arguments

mean	A Gaussian-process mean specification.
x	Numeric vector or matrix of observations.

Details

Means with fixed coefficients evaluate to $h(x)^\top \beta$. A mean whose coefficients have a Gaussian prior evaluates to its prior mean $h(x)^\top b$. Estimated coefficients, and those with a vague prior, have no value before a model is fitted; [predict_gp\(\)](#) and [coef\(\)](#) use the fitted values.

Value

A numeric vector with one mean value per observation.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
evaluate_mean(constant_mean(1), matrix(0, nrow = 4, ncol = 2))
evaluate_mean(linear_mean(c(1, 2)), c(0, 0.5, 1))
```

fit_gp

*Fit an exact Gaussian-process regression model***Description**

Fits a Gaussian-process regression model by conditioning a Gaussian prior on observed responses. The implementation uses a Cholesky factorization and triangular solves; it never forms an explicit matrix inverse.

Usage

```
fit_gp(
  x,
  y,
  kernel,
  noise_variance = 1e-06,
  mean = zero_mean(),
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps),
  derivative = NULL
)
```

Arguments

x	Numeric vector or matrix of training inputs. Matrix rows are observations and columns are input dimensions.
y	Numeric response vector with one value per training observation.
kernel	A Gaussian-process kernel specification.
noise_variance	Non-negative scalar observation-noise variance or a non-negative vector with one known variance per training observation.
mean	Gaussian-process mean specification: <code>zero_mean()</code> , <code>constant_mean()</code> , or a parametric mean such as <code>linear_mean()</code> whose coefficients are fixed, estimated, or marginalized (<code>mean_coefficients</code>).
initial_jitter	Non-negative jitter tried first, relative to the covariance scale (see Numerical jitter).
fallback_jitter	Positive relative jitter tried after the first failed factorization.
jitter_multiplier	Multiplicative increase applied to jitter after each failed factorization.
max_attempts	Maximum number of Cholesky attempts.
symmetry_tolerance	Relative tolerance for checking that the covariance matrix is symmetric.

derivative Optional observation types, one per observation: 0 if $y[i]$ observes the function value $f(x_i)$, and d if it observes the derivative $\partial f(x_i)/\partial x_d$. NULL (the default) means every observation is a function value.

Details

A scalar `noise_variance` gives the usual homoscedastic model. A vector adds a known diagonal observation-noise variance for each training point. Observation noise and numerical jitter are always stored separately.

Value

An object of class `gaussianprocesses_model`.

Derivative observations

Differentiation is linear, so derivatives of a Gaussian process are jointly Gaussian with its values, and observing them is exact conditioning. With values at X and derivatives at X' ,

$$\begin{pmatrix} y \\ y' \end{pmatrix} \sim N \left(\begin{pmatrix} m(X) \\ \partial_d m(X') \end{pmatrix}, \begin{pmatrix} K(X, X) & \partial_{x'_d} K(X, X') \\ \partial_{x_d} K(X', X) & \partial_{x_d} \partial_{x'_e} K(X', X') \end{pmatrix} + \Sigma \right),$$

where each derivative row uses its own direction d . The blocks come from the kernel's input derivatives (`kernel_input_gradient()`), so the kernel must be mean-square differentiable; otherwise an error of class `gaussianprocesses_smoothness_error` is raised. Parametric means contribute their input derivatives. `noise_variance` may differ between value and derivative observations.

Conditioning, prediction, `log_marginal_likelihood()`, its gradient, and `optimize_gp()` work unchanged with this covariance; `predict_gp()` and `predict_gradient_gp()` predict values and derivatives at new inputs. Derivative blocks scale with $1/\ell^2$, so `gp_numerical_diagnostics()` reports the conditioning of each observation type separately.

Numerical jitter

If $C = K + \Sigma_\epsilon$ does not factorize, the factorization is retried with $C + \delta I$, where $\delta = \epsilon s$ and s is the covariance scale, the mean of the diagonal of C . The relative jitter ϵ starts at `initial_jitter` (if positive), then takes the values `fallback_jitter`, `fallback_jitter * jitter_multiplier`, and so on, for at most `max_attempts` attempts. Because δ is proportional to C , results do not depend on the units of y : multiplying y by c and every variance parameter by c^2 multiplies the jitter by c^2 . The fitted model reports the absolute jitter δ in `jitter`, and `gp_numerical_diagnostics()` also reports ϵ and s .

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy. The derivative argument (derivative observations) is experimental.

See Also

[gaussianprocesses_model](#) for the structure, versioning, and persistence of fitted models.

Examples

```

x <- seq(-2, 2, length.out = 15)
y <- sin(2 * x)

model <- fit_gp(x, y, kernel = rbf_kernel(length_scale = 0.6), noise_variance = 0.01)
model

# Known, observation-specific noise variances.
fit_gp(
  x,
  y,
  kernel = rbf_kernel(length_scale = 0.6),
  noise_variance = seq(0.01, 0.1, length.out = 15)
)

```

fit_heteroscedastic_gp

Fit an approximate heteroscedastic Gaussian process

Description

Fits two coupled Gaussian processes: one for the latent regression function and one for the logarithm of input-dependent observation-noise variance. Noise targets are constructed from exact leave-one-out residuals of the current mean GP, standardized by their leave-one-out variance so that the latent function's uncertainty is not counted as noise, and corrected for the known mean of a log chi-squared variable with one degree of freedom.

Usage

```

fit_heteroscedastic_gp(
  x,
  y,
  kernel,
  noise_kernel,
  mean = zero_mean(),
  initial_noise_variance = NULL,
  noise_floor = 1e-06,
  noise_ceiling = Inf,
  damping = 0.5,
  max_iterations = 50L,
  convergence_tolerance = 0.001,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps)
)

```

Arguments

x	Numeric vector or matrix of training inputs.
y	Numeric response vector.
kernel	Kernel specification for the latent regression function.
noise_kernel	Kernel specification for log observation-noise variance.
mean	Mean specification for the latent regression function, with fixed coefficients.
initial_noise_variance	Optional positive scalar used for the initial homoscedastic mean fit. If NULL, a small fraction of response variance is used, bounded below by noise_floor.
noise_floor	Strictly positive lower bound for estimated noise variances.
noise_ceiling	Optional upper bound for estimated noise variances.
damping	Value in (0, 1] controlling how strongly each noise update replaces the previous estimate.
max_iterations	Maximum number of mean/noise refinement iterations. In a simulation study every fit that converged did so within 50 iterations, and most within 15.
convergence_tolerance	Convergence tolerance on the maximum absolute log change in training noise variance.
initial_jitter	Non-negative jitter tried first, relative to the covariance scale (the mean of its diagonal).
fallback_jitter	Positive relative jitter tried after the first failed factorization.
jitter_multiplier	Multiplicative jitter escalation factor.
max_attempts	Maximum Cholesky attempts.
symmetry_tolerance	Relative tolerance for checking that covariance matrices are symmetric.

Details

This is a deterministic plug-in approximation inspired by most-likely heteroscedastic GP methods. It is not full joint Bayesian inference over the latent function and noise process.

For a Gaussian residual $e \sim N(0, \sigma^2)$, $\log(e^2)$ has additive log-chi-squared noise with known mean $\psi(1/2) + \log 2$ and variance $\psi_1(1/2)$. The implementation uses these known moments when constructing and fitting the log-noise GP.

A leave-one-out residual e_i has variance $v_i + \sigma_i^2$, where v_i is the leave-one-out variance of the latent function, so $\log(e_i^2)$ alone overestimates the noise. The target is therefore $\log(e_i^2) - \psi(1/2) - \log 2 - \log((v_i + s_i^2)/s_i^2)$, with s_i^2 the current noise estimate, which is unbiased at a fixed point of the iteration. The research note `system.file("notes", "heteroscedastic-gp.md", package = "gaussianprocesses")` derives the correction and reports a simulation study of its bias and convergence.

The approach alternates plug-in estimates rather than integrating jointly over the latent function and noise process. If the estimate has not converged after `max_iterations` iterations, the function warns with class `gaussianprocesses_convergence_warning` and returns the last iterate with `converged = FALSE`.

Value

An object of class `gaussianprocesses_heteroscedastic_model`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[gaussianprocesses_model](#) for the structure, versioning, and persistence of fitted models.

Examples

```
simulation <- gp_simulation_scenario("heteroscedastic_1d", n = 60)

model <- fit_heteroscedastic_gp(
  simulation$x,
  simulation$observed,
  kernel = matern52_kernel(length_scale = 0.2),
  noise_kernel = rbf_kernel(length_scale = 0.4),
  max_iterations = 30
)
model

# The estimated noise variance rises with x, as in the simulation.
rows <- c(1, 20, 40, 60)
cbind(
  x = simulation$x[rows],
  true = simulation$noise_variance[rows],
  estimated = model$training_noise_variance[rows]
)
```

fit_latent_gp

Fit a latent Gaussian-process model

Description

Fits a Gaussian process $f \sim GP(m, k)$ observed through a likelihood $p(y_i \mid f(x_i))$, such as Bernoulli responses for classification or Poisson counts, with the Laplace approximation to the posterior of f . With [gaussian_likelihood\(\)](#) the approximation is exact and the model equals [fit_gp\(\)](#).

Usage

```
fit_latent_gp(
  x,
  y,
```

```

    kernel,
    likelihood,
    mean = zero_mean(),
    exposure = NULL,
    method = "laplace",
    max_iterations = 100L,
    convergence_tolerance = 1e-14,
    initial_jitter = 0,
    fallback_jitter = 1e-10,
    jitter_multiplier = 10,
    max_attempts = 8L,
    symmetry_tolerance = sqrt(.Machine$double.eps)
)

```

Arguments

<code>x</code>	Numeric vector or matrix of training inputs. Matrix rows are observations and columns are input dimensions.
<code>y</code>	Responses, one per row of <code>x</code> , as the likelihood accepts them: numbers for Gaussian likelihoods; 0/1, logical values, or a two-level factor whose second level is 1 for Bernoulli likelihoods; and non-negative integer counts for Poisson likelihoods.
<code>kernel</code>	A Gaussian-process kernel specification.
<code>likelihood</code>	A likelihood specification (gp_likelihooods). It must be log-concave.
<code>mean</code>	Gaussian-process mean specification, with fixed coefficients, coefficients estimated by maximum likelihood, or coefficients with a Gaussian coefficient_prior() .
<code>exposure</code>	Optional positive exposure for Poisson likelihoods: one value or one per observation (see poisson_likelihoood()).
<code>method</code>	The approximation: "laplace".
<code>max_iterations</code>	Maximum number of Newton iterations.
<code>convergence_tolerance</code>	The Newton iterations stop when the Newton decrement is at most this value (see Details).
<code>initial_jitter</code> , <code>fallback_jitter</code> , <code>jitter_multiplier</code> , <code>max_attempts</code>	The jitter policy for the factorization of B , relative to its mean diagonal, as in fit_gp() .
<code>symmetry_tolerance</code>	Relative tolerance for checking that B is symmetric.

Details

The posterior of $f = f(X)$ is proportional to $p(y | f)N(f | m, K)$. The Laplace approximation replaces it by $N(\hat{f}, (K^{-1} + W)^{-1})$, where $\hat{f}$ maximizes

$$\Psi(f) = \log p(y | f) - \frac{1}{2}(f - m)^\top K^{-1}(f - m)$$

and $W = -\nabla\nabla \log p(y \mid \hat{f})$ is diagonal. For a log-concave likelihood Ψ is strictly concave, so the mode is unique; other likelihoods raise an error of class `gaussianprocesses_log_concavity_error`.

Newton's method finds the mode (Rasmussen and Williams, 2006, Algorithm 3.1) in the parameterization $f = m + Ka$. Each iteration factorizes

$$B = I + W^{1/2} K W^{1/2},$$

whose eigenvalues are at least 1, so it is well conditioned even when K is nearly singular; the factorization follows the jitter policy of `fit_gp()`. A step-halving line search on Ψ prevents overshooting. The iterations stop when the Newton decrement

$$\lambda^2 = g^\top (K^{-1} + W)^{-1} g, \quad g = \nabla \Psi(f),$$

which is twice the increase of Ψ that the next Newton step predicts, is at most `convergence_tolerance`. λ is about the distance to the mode measured in posterior standard deviations, so the default of 10^{-14} places the mode within about 10^{-7} standard deviations. `$laplace` records the number of iterations and step halvings, the final decrement, the largest component of g , the last change of Ψ , and whether the iterations converged; if they did not, a warning of class `gaussianprocesses_convergence_warning` is raised.

The approximate log marginal likelihood is

$$\log q(y \mid X, \theta) = \log p(y \mid \hat{f}) - \frac{1}{2}(\hat{f} - m)^\top K^{-1}(\hat{f} - m) - \frac{1}{2} \log |B|,$$

reported by `log_marginal_likelihood()` and `logLik()`, with its gradient in `log_marginal_likelihood_gradient()`.

A mean with fixed coefficients contributes $m(x)$. Coefficients with a Gaussian prior $N(b, B_\beta)$ are integrated out: the prior becomes $m(x) = h(x)^\top b$ with covariance $k(x, x') + h(x)^\top B_\beta h(x')$. Coefficients estimated by maximum likelihood (`estimate_coefficients("ml")`, the default of `linear_mean()`) maximize $\log q(y \mid X, \theta, \beta)$ at the given hyperparameters, by BFGS with the analytical gradient $H^\top a + [(I + KW)^{-1} H]^\top s$, where s is the derivative of $-\frac{1}{2} \log |B|$ in $\hat{f}$, starting from the generalized linear model without the latent process. The estimate is plugged in, like the hyperparameters; `coef()` returns it and `$mean_fit` records it. A constant mean estimated this way is the usual mean level of a count or classification model. Restricted likelihoods and vague priors are not supported.

Value

An object of class `gaussianprocesses_latent_model` with fields `x`, `y` (responses in the internal coding), `response_levels` (the levels of a factor response), `exposure`, `kernel`, `mean`, `likelihood`, `method`, `mode` ($\hat{f}$), `alpha` (a), `curvature` (the diagonal of W), `b_cholesky` (R with $R^\top R = B + \delta I$), `jitter` (δ), `log_marginal_likelihood`, `laplace`, `n_observations`, `n_features`, `schema_version`, and `package_version`. Other fields are internal.

Accuracy of the Laplace approximation

For a few observations the exact marginal likelihood $\log p(y \mid X)$ can be computed by tensor-product Gauss-Hermite quadrature. With an RBF kernel (variance 2, length scale 1) and the first n of the inputs $x = 0, 0.5, 2$ and of the responses below, the error $\log q - \log p$ of the Laplace approximation is:

Likelihood	Responses	$n = 1$	$n = 2$	$n = 3$
logit	1, 1, 0	-0.0171	-0.0192	-0.0387
probit	1, 1, 0	-0.0196	-0.0298	-0.0450
Poisson	0, 2, 7	0.0125	-0.0101	-0.0197
Poisson, mean 5	120, 150, 200	-0.000681	-0.00118	-0.00158

For binary responses and small counts the error is a few hundredths and grows with n ; the approximation can over- or underestimate. For counts in the hundreds, where the likelihood is nearly Gaussian in f , it is ten times smaller. The test suite reproduces these values.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. Chapters 3 and 5.

See Also

[predict_latent_gp\(\)](#), [optimize_latent_gp\(\)](#), [sample_gp_posterior\(\)](#).

Examples

```
x <- seq(-3, 3, length.out = 30)
y <- as.numeric(sin(x) + 0.3 * cos(5 * x) > 0)

model <- fit_latent_gp(x, y, rbf_kernel(variance = 4), bernoulli_likelihood())
model
predict_latent_gp(model, c(-2, 0, 2))$probability
```

fit_sparse_gp

Fit a sparse Gaussian process (FITC or VFE)

Description

Fits a sparse approximation with fixed inducing locations: the Fully Independent Training Conditional (FITC) approximation or the collapsed variational free energy (VFE) of Titsias (2009). Exact GP regression remains the reference model; these provide a lower-cost path when the number of inducing points is much smaller than the number of observations.

Usage

```
fit_sparse_gp(
  x,
  y,
  kernel,
  noise_variance = 1e-06,
  mean = zero_mean(),
  n_inducing = NULL,
  inducing_points = NULL,
  selection = c("farthest", "quantile", "variance"),
  method = c("fitc", "vfe"),
  fitc_diagonal_floor = 1e-10,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

<code>x</code>	Numeric vector or matrix of training inputs.
<code>y</code>	Numeric response vector.
<code>kernel</code>	Gaussian-process kernel specification.
<code>noise_variance</code>	Non-negative scalar or one non-negative value per training observation.
<code>mean</code>	Gaussian-process mean specification with fixed coefficients.
<code>n_inducing</code>	Optional number of inducing points selected from <code>x</code> .
<code>inducing_points</code>	Optional explicit inducing locations. If supplied, <code>n_inducing</code> is ignored.
<code>selection</code>	Inducing-point selection method used when explicit points are not supplied: "farthest", "quantile", or "variance" (greedy conditional-variance selection under kernel; see select_inducing_points()).
<code>method</code>	The approximation: "fitc" (the default) or "vfe". VFE needs positive noise variances.
<code>fitc_diagonal_floor</code>	Strictly positive numerical floor applied to the FITC diagonal term before inversion.
<code>initial_jitter</code>	Non-negative jitter tried first in the inducing-system Cholesky factorizations, relative to the scale of each factorized matrix (the mean of its diagonal).
<code>fallback_jitter</code>	Positive relative jitter tried after the first failed factorization.
<code>jitter_multiplier</code>	Multiplicative jitter escalation factor.
<code>max_attempts</code>	Maximum Cholesky attempts.
<code>symmetry_tolerance</code>	Relative tolerance for checking that covariance matrices are symmetric.

Details

Let m be the number of inducing points and n the number of training observations. FITC replaces the full training covariance with a low-rank inducing covariance plus a diagonal correction. The dominant training cost is $O(nm^2 + m^3)$ rather than the exact GP's $O(n^3)$, with $m \ll n$. The implementation stores an $m \times n$ cross covariance rather than an $n \times n$ training covariance.

FITC preserves each training point's prior marginal variance through the diagonal correction, unlike simpler deterministic training conditional approximations. It is still an approximation: inducing locations and their number can affect posterior means, variances, and the approximate marginal likelihood.

If the inducing covariance needs numerical jitter, the jittered matrix is used consistently in the low-rank term, the posterior, and the approximate log marginal likelihood. The inducing posterior is factorized in whitened form, whose eigenvalues are bounded below by one.

Value

An object of class `gaussianprocesses_sparse_model`.

VFE

With $Q_{ff} = K_{fu}K_{uu}^{-1}K_{uf}$ and noise $\Sigma = \text{diag}(\sigma_i^2)$, the collapsed variational bound of Titsias (2009) is

$$\mathcal{F} = \log N(y \mid m(X), Q_{ff} + \Sigma) - \frac{1}{2} \sum_i \frac{k(x_i, x_i) - [Q_{ff}]_{ii}}{\sigma_i^2} \leq \log p(y).$$

It is a lower bound on the exact log marginal likelihood, reported as `log_marginal_likelihood`, so maximizing it over the hyperparameters cannot overfit the way FITC can: FITC's diagonal correction $\text{diag}(K_{ff} - Q_{ff})$ can absorb noise, and FITC tends to underestimate the noise variance (Bauer, van der Wilk, and Rasmussen, 2016). VFE predicts with the deterministic training conditional, which has the same form as the FITC prediction with $\Lambda = \Sigma$.

Both models report the trace term $t = \sum_i (k(x_i, x_i) - [Q_{ff}]_{ii}) / \sigma_i^2$ in `trace_term`: the prior variance that the inducing points do not explain, in units of noise variance. A trace term much larger than 1 means that the inducing points do not cover the data well: VFE's bound is loose by at least $t/2$ and FITC leans on its diagonal correction. More or better placed inducing points reduce it.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

- Titsias, M. (2009). Variational learning of inducing variables in sparse Gaussian processes. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, 567–574.
- Bauer, M., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse Gaussian process approximations. *Advances in Neural Information Processing Systems*, 29.

See Also

[gaussianprocesses_model](#) for the structure, versioning, and persistence of fitted models.

Examples

```
x <- seq(-3, 3, length.out = 200)
y <- sin(x) + 0.1 * cos(7 * x)

model <- fit_sparse_gp(
  x,
  y,
  kernel = rbf_kernel(length_scale = 0.8),
  noise_variance = 0.01,
  n_inducing = 15
)
model
```

fit_time_series_gp	<i>Fit a GP to ordered time-series observations</i>
--------------------	---

Description

Fit a GP to ordered time-series observations

Usage

```
fit_time_series_gp(
  time,
  y,
  kernel,
  unit = "auto",
  method = c("exact", "state_space"),
  ...
)
```

Arguments

time	Numeric, Date, or POSIXt vector in strictly increasing order.
y	Numeric response vector.
kernel	Gaussian-process kernel specification.
unit	Time unit passed to gp_time_index() .
method	"exact" conditions a Gaussian prior on the observations with fit_gp() . "state_space" computes the same posterior and log marginal likelihood in $O(n)$ time by Kalman filtering and smoothing; it needs a Matérn-1/2, 3/2, or 5/2 kernel, a sum of such kernels, or scaled versions of them (see State-space inference).
...	Additional arguments passed to fit_gp() . With method = "state_space" only noise_variance (positive) and mean (with fixed coefficients) apply.

Details

This function was called `fit_gp_time()` in version 0.1.0.

Value

A fitted model containing explicit time-index metadata used by `forecast_gp()`: a `gaussianprocesses_model` for `method = "exact"` and a `gaussianprocesses_state_space_model` for `method = "state_space"`.

State-space inference

On one input, a Matérn process with $\nu = p + 1/2$ is the first component of the state $x = (f, f', \dots, f^{(p)})$ of the linear stochastic differential equation $dx = Fx dt + L dW$, with F the companion matrix of $(s + \lambda)^{p+1}$, $\lambda = \sqrt{2\nu}/\ell$, and white noise W of spectral density q_c (Hartikainen and Särkkä, 2010). The stationary state covariance P_∞ solves $FP_\infty + P_\infty F^\top + Lq_cL^\top = 0$, and $k(\tau) = He^{F\tau}P_\infty H^\top$ for $\tau \geq 0$ with $H = (1, 0, \dots, 0)$. Sums of kernels stack their states, and scaled kernels scale P_∞ and q_c . Between consecutive times the state moves by $A = e^{F\Delta t}$, which has a closed form because $F + \lambda I$ is nilpotent, with process noise $P_\infty - AP_\infty A^\top$.

The Kalman filter (with Joseph-form updates) gives the log marginal likelihood by the prediction-error decomposition, the Rauch-Tung-Striebel smoother gives the posterior at the training times, and `forecast_gp()` interleaves the requested times as unobserved steps. The results equal those of the exact GP up to rounding, at $O(n)$ cost instead of $O(n^3)$, and no numerical jitter is used. `sample_gp_posterior()` draws by forward filtering, backward sampling. Kernels without an exact finite-dimensional state (RBF, rational quadratic, periodic, linear, and products) raise an error of class `gaussianprocesses_state_space_error`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy. `method = "state_space"` is experimental.

References

Hartikainen, J., and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal Gaussian process regression models. *IEEE International Workshop on Machine Learning for Signal Processing*, 379–384.

Särkkä, S., and Solin, A. (2019). *Applied Stochastic Differential Equations*. Cambridge University Press.

See Also

`optimize_time_series_gp()` to estimate the hyperparameters.

Examples

```
time <- as.Date("2026-01-01") + 0:23
y <- sin(2 * pi * (0:23) / 12)
```

```

model <- fit_time_series_gp(
  time,
  y,
  kernel = time_series_kernel(period = 12),
  noise_variance = 0.05
)
model$time[c("origin", "unit")]

# The same posterior by Kalman smoothing, for a Matern kernel.
kernel <- matern32_kernel(length_scale = 3)
exact <- fit_time_series_gp(time, y, kernel, noise_variance = 0.05)
kalman <- fit_time_series_gp(time, y, kernel, noise_variance = 0.05,
  method = "state_space")
c(logLik(exact), logLik(kalman))

```

forecast_gp

*Forecast or interpolate from a time-aware GP model***Description**

Converts future time values using the training model's original time origin and units, then evaluates the GP posterior: exactly, or by Kalman smoothing for state-space models. Each requested point is labelled as interpolation, backward extrapolation, or forward forecast.

Usage

```

forecast_gp(
  model,
  future_time,
  interval_level = 0.95,
  include_covariance = FALSE,
  variance_tolerance = sqrt(.Machine$double.eps),
  observation_noise_variance = NULL
)

```

Arguments

model	A model fitted by fit_time_series_gp() or optimize_time_series_gp() .
future_time	Numeric, Date, or POSIXt values matching the training time type and supplied in strictly increasing order.
interval_level	Probability level for posterior intervals.
include_covariance	If TRUE, include full posterior covariance matrices.
variance_tolerance	Relative tolerance for posterior variances.
observation_noise_variance	Optional non-negative scalar or vector of observation-noise variances at future_time, passed to predict_gp() . Required when the model was fitted with observation-specific training noise.

Value

An object of class `gaussianprocesses_time_forecast`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
time <- as.Date("2026-01-01") + 0:23
model <- fit_time_series_gp(
  time,
  sin(2 * pi * (0:23) / 12),
  kernel = time_series_kernel(period = 12),
  noise_variance = 0.05
)

forecast <- forecast_gp(model, as.Date("2026-01-01") + c(10.5, 25, 30))
data.frame(
  region = forecast$region,
  mean = forecast$mean,
  forecast$prediction_interval
)
```

forecast_metrics

Uncertainty-aware forecast metrics

Description

Summarizes a rolling-origin GP backtest with both point and probabilistic forecast metrics.

Usage

```
forecast_metrics(backtest)
```

Arguments

`backtest` A `gaussianprocesses_backtest` returned by [rolling_origin_gp\(\)](#).

Value

A named list containing RMSE, MAE, standardized RMSE, empirical interval coverage, mean interval width, and mean log predictive density.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
time <- 0:23
backtest <- rolling_origin_gp(
  time,
  sin(2 * pi * time / 12),
  kernel = time_series_kernel(period = 12),
  initial_window = 12,
  horizon = 3,
  step = 3,
  noise_variance = 0.05
)

forecast_metrics(backtest)
```

gaussianprocesses_model

Fitted model objects

Description

The structure, versioning, and persistence of the models returned by [fit_gp\(\)](#), [optimize_gp\(\)](#), [fit_time_series_gp\(\)](#), [fit_sparse_gp\(\)](#), [fit_heteroscedastic_gp\(\)](#), [fit_latent_gp\(\)](#), [optimize_latent_gp\(\)](#), and [optimize_sparse_gp\(\)](#).

Exact models

Objects of class `gaussianprocesses_model`, from [fit_gp\(\)](#), [optimize_gp\(\)](#), and [fit_time_series_gp\(\)](#), document these fields:

`x, y` Training inputs as a matrix and responses.

`derivative` For derivative observations, the type of each observation: 0 for a function value and d for $\partial f / \partial x_d$. NULL when every observation is a function value.

`kernel, mean` The kernel and mean specifications. A polynomial mean records the centring and scaling of its inputs.

`mean_fit` For estimated or marginalized mean coefficients, a list with treatment, method ("ml", "reml", "gaussian", or "vague"), the coefficients $\bar{\beta}$, the basis matrix $\text{basis}(H)$, and coefficient_factor, an upper-triangular G with $G^\top G = H^\top C^{-1} H$ plus B^{-1} under a Gaussian prior. NULL for other means. Use [coef\(\)](#) and [vcov\(\)](#) to read the coefficients.

`noise_variance, training_noise_variance, noise_structure` The noise variance as supplied, one value per training observation, and "homoscedastic" or "known_heteroscedastic".

cholesky Upper-triangular R with $R^\top R = C + \delta I$, where $C = K + \Sigma_\epsilon$ and δ is jitter.

alpha $\alpha = (C + \delta I)^{-1}(y - m(X))$, with $m(X) = H\bar{\beta}$ for estimated or marginalized coefficients.

jitter Diagonal jitter δ that the factorization needed; usually 0. It is a multiple of the mean diagonal of C (see `fit_gp()`).

n_observations, n_features Training size and input dimension.

optimization Optimizer diagnostics, for `optimize_gp()` models.

time Time metadata, for `fit_time_series_gp()` models.

schema_version, package_version See below.

The kernel matrix K and the observation covariance C are not stored, because they can be recomputed exactly: `evaluate_kernel(model$kernel, model$x)` is K , and adding `diag(model$training_noise_variance)` gives C . Storing only R keeps a model with n observations at about $8n^2$ bytes. Other fields, such as `centered_y`, `prior_mean`, `cholesky_attempts`, and `numerical_policy`, are internal and may change without notice.

Sparse and heteroscedastic models

Sparse models (`gaussianprocesses_sparse_model`) document `x`, `y`, `kernel`, `mean`, `noise_variance`, `inducing_points`, `n_inducing`, `method` ("fitc" or "vfe"), `approximation` ("FITC" or "VFE"), `log_marginal_likelihood` (the FITC objective or the VFE bound), `trace_term`, and, for `optimize_sparse_gp()` models, `optimization`; their other fields are internal. Heteroscedastic models (`gaussianprocesses_heteroscedastic_model`) document `x`, `y`, `kernel`, `noise_kernel`, `training_noise_variance`, `converged`, `iterations`, and `history`, and contain two exact models, `mean_model` and `noise_model`. Both record `schema_version` and `package_version`.

Latent models

Latent models (`gaussianprocesses_latent_model`) document the fields listed in `fit_latent_gp()`. Like exact models, they store the Cholesky factor of $B = I + W^{1/2}KW^{1/2}$ but not K , and record `schema_version` and `package_version`; they were added at schema 3.

State-space models

State-space models (`gaussianprocesses_state_space_model`), from `fit_time_series_gp()` with `method = "state_space"`, store the training data, `kernel`, `mean`, `noise`, `log_marginal_likelihood`, and the smoothed fitted means and variances, but no covariance matrix or factor: every prediction filters and smooths again. They were added at schema 4.

Schema versions

`schema_version` identifies the structure of a model, and `package_version` is the version of `gaussianprocesses` that created it. The current schema is 4: schema 2 added `mean_fit`, schema 3 added `derivative`, and schema 4 added the `method` and `trace_term` of sparse models. Schema 4 is the format of version 1.0. Models saved by versions that predate schema versions, which stored K and C , are schema 0, and models saved before parametric means were added are schema 1. Every function that accepts a model first converts older schemas to schema 4; the converted model gives the same results, and a converted sparse model is a FITC model. A converted schema-0 model has `package_version` NA, because the creating version is unknown.

A model with a newer or unknown schema, or without the fields its class requires, raises an error of class `gaussianprocesses_model_format_error`. The condition object has fields `found_schema`, `supported_schemas`, and `package_version`.

Persistence

Models can be saved with `saveRDS()` and restored with `readRDS()`; a restored model gives the same predictions, likelihoods, gradients, diagnostics, and seeded draws as the original.

Every 1.x release reads models saved by any earlier version, back to 0.1.0. A 1.x release may introduce a new schema only together with an upgrade from schema 4, so models saved by version 1.0 stay readable and give the same results. A model saved by a newer version than the one reading it raises `gaussianprocesses_model_format_error` rather than giving wrong results. The tests read models saved by the 0.1.0 release and schema-4 models written before 1.0, and check that they give the results they were saved with. Keeping the training data with the kernel, mean, and noise also reproduces a model exactly: refitting them with the same numerical settings gives the same model.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 10)
model <- fit_gp(x, sin(x), rbf_kernel(), noise_variance = 0.01)

model$schema_version

# C is recomputed from the kernel, and R'R reproduces it.
C <- evaluate_kernel(model$kernel, model$x) +
  diag(model$training_noise_variance)
all.equal(crossprod(model$cholesky), C + diag(model$jitter, nrow(C)))

# A saved model gives identical predictions.
path <- tempfile(fileext = ".rds")
saveRDS(model, path)
identical(predict_gp(readRDS(path), 0.5), predict_gp(model, 0.5))
```

gp_calibration_diagnostics

Summarize Gaussian-process uncertainty calibration

Description

Summarizes exact LOO predictive diagnostics. The returned statistics are descriptive diagnostics rather than a pass/fail calibration test.

Usage

```
gp_calibration_diagnostics(model, interval_level = 0.95)
```

Arguments

`model` A fitted `gaussianprocesses_model`.
`interval_level` Probability level for LOO prediction intervals.

Details

PIT values should be assessed as a distribution rather than individually. Under a correctly specified continuous predictive distribution they are uniform in repeated sampling.

Value

A named list of calibration summaries.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 20)
model <- fit_gp(
  x,
  sin(2 * x),
  kernel = rbf_kernel(length_scale = 0.6),
  noise_variance = 0.01
)

gp_calibration_diagnostics(model, interval_level = 0.9)
```

`gp_classification_scores`

Scores of binary probability forecasts

Description

Scores predicted class probabilities against binary responses with proper scoring rules (log loss and Brier score), discrimination (ROC AUC), and calibration (reliability diagram data and the expected calibration error). [gp_holdout_scores\(\)](#) returns these scores for latent models with a Bernoulli likelihood.

Usage

```
gp_classification_scores(
  observed,
  probability,
  n_bins = 10L,
  log_density = NULL
)
```

Arguments

<code>observed</code>	Binary responses: 0/1 numbers, logical values, or a two-level factor whose second level is the event coded 1, as in bernoulli_likelihoood() .
<code>probability</code>	Forecast probabilities $p_i = \Pr(y_i = 1)$, one per response.
<code>n_bins</code>	Number of equal-width probability bins of the reliability diagram and the expected calibration error.
<code>log_density</code>	Optional log predictive probabilities of the observed responses, for the log loss. They default to $\log p_i$ or $\log(1 - p_i)$; gp_holdout_scores() supplies them from the predictive distribution, which keeps their accuracy when p_i is close to 0 or 1.

Details

log loss $-\frac{1}{n} \sum_i \log p(y_i)$, the negative mean log predictive probability. Proper; lower is better; $\log 2 \approx 0.693$ for $p_i = 1/2$.

Brier score $\frac{1}{n} \sum_i (p_i - y_i)^2$. Proper; lower is better; 0.25 for $p_i = 1/2$.

ROC AUC The probability that a random event receives a higher forecast than a random non-event, with ties counted as one half: the Mann-Whitney statistic from mid-ranks. 0.5 for uninformative forecasts; NA without both classes. It measures discrimination only and ignores calibration.

reliability The forecasts are grouped into `n_bins` equal-width bins of $[0, 1]$. For calibrated forecasts the observed frequency of each bin is close to its mean probability, within about $\text{standard_error} = \sqrt{\bar{p}(1 - \bar{p})/n_b}$.

ECE The expected calibration error $\sum_b \frac{n_b}{n} |\bar{p}_b - \bar{y}_b|$. It depends on the binning, and even calibrated forecasts have a positive ECE of the order of the bin standard errors.

Value

An object of class `gaussianprocesses_classification_scores` with

`summary` Named numeric vector: `n`, `log_loss`, `brier_score`, `auc`, `ece`, `base_rate` (the mean response), and `mean_probability`.

`reliability` Data frame with one row per non-empty bin: `bin`, `lower`, `upper`, `n`, `mean_probability`, `observed_frequency`, and `standard_error`.

`pointwise` Data frame with `observed` (0/1), `probability`, `log_density`, and `squared_error`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477), 359–378.

See Also

[gp_scores\(\)](#) for Gaussian predictions, [predict_latent_gp\(\)](#).

Examples

```
set.seed(1)
x <- runif(80, 0, 6)
y <- rbinom(80, 1, plogis(2 * sin(x)))
training <- 1:50

model <- fit_latent_gp(x[training], y[training], rbf_kernel(4, 1),
  bernoulli_likelihood())
scores <- gp_holdout_scores(model, x[-training], y[-training], n_bins = 5)
scores
scores$reliability
```

gp_count_scores	<i>Scores of count forecasts</i>
-----------------	----------------------------------

Description

Scores Poisson-lognormal predictive distributions of counts, those of latent Gaussian-process models with a [poisson_likelihood\(\)](#), with proper scores, a dispersion check, interval coverage, and randomized probability integral transform (PIT) values. [gp_holdout_scores\(\)](#) returns these scores for latent Poisson models.

Usage

```
gp_count_scores(
  observed,
  latent_mean,
  latent_variance,
  exposure = NULL,
  interval_levels = c(0.5, 0.8, 0.9, 0.95),
  seed = NULL,
  quadrature_order = 40L
)
```

Arguments

observed	Non-negative integer counts.
latent_mean, latent_variance	Mean and variance of the latent log rate f at each observation, as returned by predict_latent_gp() (mean and latent_variance).
exposure	Optional positive exposure: one value or one per count.
interval_levels	Levels of the equal-tailed prediction intervals whose coverage is reported.
seed	Optional seed for the randomization of the PIT values. The caller's random state is restored afterwards.
quadrature_order	Number of Gauss-Hermite nodes.

Details

Given $f \sim N(\mu, \sigma^2)$, a count $y \sim \text{Poisson}(Ee^f)$ has the Poisson-lognormal predictive distribution, with mean $Ee^{\mu+\sigma^2/2}$ and probabilities, distribution function, and quantiles computed by Gauss-Hermite quadrature.

log score The mean log predictive probability of the counts. Proper; higher is better.

Dawid-Sebastiani score $(y - m)^2/s^2 + 2 \log s$ with the predictive mean m and standard deviation s ; proper for the first two moments and recommended for counts by Czado, Gneiting, and Held (2009). Lower is better.

dispersion The mean squared Pearson residual $(y - m)^2/s^2$. It is about 1 when the predictive variance is right; clearly larger values mean that the counts are overdispersed relative to the model.

coverage Of the equal-tailed prediction intervals at each level. Counts are discrete, so the intervals cover at least their level: their coverage is conservative, by up to the probability of their end points.

randomized PIT $u = F(y - 1) + v p(y)$ with $v \sim U(0, 1)$: uniform on $[0, 1]$ when the forecasts are calibrated, unlike the plain PIT $F(y)$ of a discrete distribution. The summary reports their mean (1/2), variance (1/12), and the p-value of a Kolmogorov-Smirnov test of uniformity.

Value

An object of class `gaussianprocesses_count_scores` with

`summary` Named numeric vector: `n`, `log_score`, `dawid_sebastiani`, `dispersion`, `pit_mean`, `pit_variance`, and `pit_ks_p_value`.

`calibration` Data frame with one row per interval level: the level, empirical coverage, and mean width (upper minus lower count).

`pointwise` Data frame with `observed`, `predictive_mean`, `predictive_variance`, `log_score`, `dawid_sebastiani`, `pearson_residual`, and `pit`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Czado, C., Gneiting, T., and Held, L. (2009). Predictive model assessment for count data. *Biometrics*, 65(4), 1254–1261.

See Also

[gp_scores\(\)](#), [gp_classification_scores\(\)](#), [predict_latent_gp\(\)](#).

Examples

```
set.seed(1)
x <- seq(0, 10, length.out = 60)
counts <- rpois(60, exp(1 + sin(x)))
training <- seq(1, 60, by = 2)

model <- fit_latent_gp(x[training], counts[training], rbf_kernel(0.5, 1.5),
  poisson_likelihood(), mean = constant_mean(1))
gp_holdout_scores(model, x[-training], counts[-training], seed = 1)
```

gp_diagnostics

Comprehensive diagnostics for an exact Gaussian-process model

Description

Combines exact leave-one-out predictive diagnostics, uncertainty-calibration summaries, and numerical-conditioning diagnostics. The components are kept separate so predictive misfit is not automatically attributed to numerical instability, and numerical instability is not mistaken for model misfit.

Usage

```
gp_diagnostics(
  model,
  interval_level = 0.95,
  condition_tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

model A fitted `gaussianprocesses_model`.

interval_level Probability level for LOO prediction intervals.

condition_tolerance Positive reciprocal-condition threshold used by the numerical diagnostic summary.

Value

An object of class `gaussianprocesses_diagnostics`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy. Its numerical component comes from the experimental [gp_numerical_diagnostics\(\)](#).

Examples

```
x <- seq(-2, 2, length.out = 20)
model <- fit_gp(
  x,
  sin(2 * x),
  kernel = rbf_kernel(length_scale = 0.6),
  noise_variance = 0.01
)

gp_diagnostics(model)
```

gp_hyperparameter_uncertainty

Uncertainty of estimated hyperparameters

Description

Approximates the uncertainty of a model's hyperparameters from the curvature of the log marginal likelihood at its optimum, and flags hyperparameters that the data identify poorly.

Usage

```
gp_hyperparameter_uncertainty(
  model,
  step = 1e-04,
  interval_level = 0.95,
  condition_threshold = 1e+06,
  log_scale_threshold = log(2)
)
```

Arguments

model	A fitted <code>gaussianprocesses_model</code> , normally from optimize_gp() .
step	Step of the central differences of the gradient, on the optimizer's coordinate scale.
interval_level	Level of the approximate intervals.
condition_threshold	Condition number of the observed information above which the hyperparameters are flagged as poorly identified.

log_scale_threshold

Standard deviation, on the log scale, of the least determined combination of log-scale parameters above which they are flagged as poorly identified. The default, log 2, flags a combination known only to within a factor of two at one standard deviation.

Details

The hyperparameters are those `optimize_gp()` estimated (all kernel parameters and the noise for other models), on its coordinate scale η : logarithms of positive parameters and real parameters themselves. The observed information $I = -\nabla^2 \ell(\hat{\eta})$ is computed by central differences of the analytical gradient (`log_marginal_likelihood_gradient()`), one refit per displaced coordinate. Its inverse is the covariance of the Laplace approximation with a flat prior on the coordinate scale:

$$\widehat{\text{Cov}}(\hat{\eta}) \approx I^{-1}.$$

Standard errors and correlations are reported on the coordinate scale, and approximate intervals $\hat{\eta} \pm z \text{ se}$ are mapped back to the natural scale, so that intervals for positive parameters stay positive. Estimated mean coefficients are profiled out (the likelihood is the profile, restricted, or marginal likelihood of the mean), and the approximation conditions on them.

The approximation needs a maximum: gradient reports the gradient at the estimate, which should be close to zero. Some combinations of hyperparameters can be nearly unidentifiable: under infill asymptotics, only $\sigma^2/\ell^{2\nu}$ is consistently estimable for a Matérn kernel of smoothness ν (Zhang, 2004). The likelihood is then nearly flat along a ridge: the information is nearly singular in absolute terms, though not always in relative ones, because the curvature across the ridge can be small too. `poorly_identified` is therefore TRUE when the information is not positive definite, when its condition number exceeds `condition_threshold`, or when the standard deviation along the least determined combination of the log-scale parameters, `weakest_log_scale_sd`, exceeds `log_scale_threshold`. `weakest_direction` is the eigenvector of the smallest eigenvalue, the direction along the ridge; for a Matérn kernel under infill it is close to $(2\nu, 1)/\sqrt{4\nu^2 + 1}$ in (log variance, log length scale). Real-valued parameters, such as changepoint locations, have no common scale with the others and enter only the first two criteria. `gp_profile_likelihood()` shows the likelihood along one parameter without the quadratic approximation.

Value

An object of class `gaussianprocesses_hyperparameter_uncertainty` with

`parameters` Data frame: `parameter`, `estimate`, `coordinate` ("log" or "identity"), `coordinate_estimate`, `standard_error` on the coordinate scale, and `lower` and `upper` on the natural scale.

`covariance`, `correlation` On the coordinate scale; NULL when the information is not positive definite.

`information`, `eigenvalues`, `condition_number`, `weakest_log_scale_sd`, `poorly_identified`, `weakest_direction` The observed information and its diagnostics.

`gradient` The gradient at the estimate.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Zhang, H. (2004). Inconsistent estimation and asymptotically equal interpolations in model-based geostatistics. *Journal of the American Statistical Association*, 99(465), 250–261.

See Also

[gp_profile_likelihood\(\)](#), and the article on hyperparameter uncertainty on the documentation site.

Examples

```
set.seed(1)
x <- seq(0, 10, length.out = 60)
y <- sin(x) + rnorm(60, sd = 0.2)
model <- optimize_gp(x, y, rbf_kernel(), noise_variance = 0.1, n_starts = 1)

uncertainty <- gp_hyperparameter_uncertainty(model)
uncertainty
uncertainty$correlation
```

gp_likelihooods

Likelihoods for Gaussian-process models

Description

Observation models $p(y_i | f_i)$ that link the latent Gaussian process f to the responses. Like kernel specifications, they are plain data with a class, a print method, and constrained parameters.

Usage

```
gaussian_likelihood(variance = 1)

bernoulli_likelihood(link = c("logit", "probit"))

poisson_likelihood(link = "log")
```

Arguments

variance	Positive noise variance of the Gaussian likelihood.
link	Link function: "logit" or "probit" for Bernoulli responses, and "log" for Poisson counts.

Details

`gaussian_likelihood()` $y_i \sim N(f_i, \sigma^2)$, the observation model of `fit_gp()` with noise_variance σ^2 .

`bernoulli_likelihood()` $\Pr(y_i = 1) = \sigma(f_i)$ with the logistic function, $\log p = yf - \log(1 + e^f)$, or $\Phi(f_i)$ with the probit link. Responses may be 0/1 numbers, logical values (TRUE is 1), or a two-level factor whose second level is 1, as in `stats::glm()`. Multi-class (softmax) likelihoods are not supported.

`poisson_likelihood()` $y_i \sim \text{Poisson}(E_i e^{f_i})$ with exposure E_i , so $\log p = y(f + \log E) - E e^f - \log y!$. The exposure is data, supplied where the likelihood is used, not part of the specification. Counts more variable than a Poisson process with a smooth rate allows need negative-binomial or zero-inflated likelihoods, which are not yet supported; `gp_count_scores()` checks for such overdispersion.

Every likelihood provides, vectorized over observations, the log density and its first three derivatives in f_i (`evaluate_likelihood()`), derivatives in its parameters, the predictive distribution of a new response when f_* is Gaussian (`likelihood_predictive()`), and whether the log density is concave in f . All three are log-concave, which makes Laplace approximations well behaved. Gaussian expectations without a closed form use Gauss-Hermite quadrature: nodes from the Golub-Welsch eigenvalue method, weights from Christoffel sums, cached by order.

The probit derivatives use the inverse Mills ratio $r = \varphi(z)/\Phi(z)$ computed in log space, with $z = (2y - 1)f$. For $z < -5$, where r grows like $-z$ and $z + r$ cancels, they use the continued fraction of the Mills ratio instead. The logistic derivatives compute π and $1 - \pi$ separately. All derivatives stay finite and accurate for large $|f|$.

Latent models with any of these likelihoods are fitted with `fit_latent_gp()` and `optimize_latent_gp()`; `fit_gp()` remains the exact Gaussian model.

Value

An object of class `gaussianprocesses_likelihood`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

- Golub, G. H. and Welsch, J. H. (1969). Calculation of Gauss quadrature rules. *Mathematics of Computation*, 23(106), 221–230.
- Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. Chapter 3.

See Also

[evaluate_likelihood\(\)](#), [likelihood_predictive\(\)](#), [fit_latent_gp\(\)](#).

Examples

```

gaussian_likelihood(variance = 0.1)
bernoulli_likelihood("probit")
poisson_likelihood()

evaluate_likelihood(bernoulli_likelihood(), c(TRUE, FALSE), f = c(1, 1))

```

gp_mean_functions	<i>Gaussian-process mean functions</i>
-------------------	--

Description

Mean specifications separate the deterministic part of the model from the covariance kernel. `zero_mean()` is zero everywhere and `constant_mean()` is a constant, fixed or with an estimated or marginalized value. For regression trends see [linear_mean\(\)](#), [polynomial_mean\(\)](#), and [basis_mean\(\)](#).

Usage

```

zero_mean()

constant_mean(value = 0)

```

Arguments

value	A finite number that fixes the constant, or estimate_coefficients() or coefficient_prior() for a constant that is estimated from the data or has a prior (ordinary kriging).
-------	--

Value

An object of class `gaussianprocesses_mean`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```

zero_mean()
constant_mean(2.5)

evaluate_mean(constant_mean(2.5), 1:3)

# Estimate the constant instead.
x <- seq(0, 3, length.out = 15)
model <- fit_gp(x, 4 + sin(2 * x), rbf_kernel(), noise_variance = 0.01,
  mean = constant_mean(estimate_coefficients()))
coef(model)

```

Description

Methods for `stats::predict()`, `stats::logLik()`, `stats::nobs()`, `stats::fitted()`, `stats::residuals()`, and `summary()` on `exact` (`gaussianprocesses_model`), `sparse` (`gaussianprocesses_sparse_model`), `heteroscedastic` (`gaussianprocesses_heteroscedastic_model`), `latent` (`gaussianprocesses_latent_model`), and `state-space` (`gaussianprocesses_state_space_model`) models.

Usage

```
## S3 method for class 'gaussianprocesses_model'
predict(object, newdata, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
predict(object, newdata, ...)

## S3 method for class 'gaussianprocesses_state_space_model'
predict(object, newdata, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
predict(object, newdata, ...)

## S3 method for class 'gaussianprocesses_latent_model'
predict(object, newdata, ...)

## S3 method for class 'gaussianprocesses_model'
logLik(object, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
logLik(object, ...)

## S3 method for class 'gaussianprocesses_state_space_model'
logLik(object, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
logLik(object, ...)

## S3 method for class 'gaussianprocesses_latent_model'
logLik(object, ...)

## S3 method for class 'gaussianprocesses_model'
nobs(object, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
nobs(object, ...)
```

```
## S3 method for class 'gaussianprocesses_state_space_model'
nobs(object, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
nobs(object, ...)

## S3 method for class 'gaussianprocesses_latent_model'
nobs(object, ...)

## S3 method for class 'gaussianprocesses_model'
fitted(object, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
fitted(object, ...)

## S3 method for class 'gaussianprocesses_state_space_model'
fitted(object, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
fitted(object, ...)

## S3 method for class 'gaussianprocesses_latent_model'
fitted(object, ...)

## S3 method for class 'gaussianprocesses_model'
residuals(object, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
residuals(object, ...)

## S3 method for class 'gaussianprocesses_state_space_model'
residuals(object, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
residuals(object, ...)

## S3 method for class 'gaussianprocesses_model'
summary(object, ...)

## S3 method for class 'gaussianprocesses_sparse_model'
summary(object, ...)

## S3 method for class 'gaussianprocesses_state_space_model'
summary(object, ...)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
summary(object, ...)
```

```
## S3 method for class 'gaussianprocesses_latent_model'
summary(object, ...)

## S3 method for class 'summary.gaussianprocesses_model'
print(x, ...)
```

Arguments

<code>object</code>	A fitted model.
<code>newdata</code>	Prediction inputs. If missing, the training inputs (or, for time-series models, the training times) are used.
<code>...</code>	Further arguments passed to the package's prediction function: <code>predict_gp()</code> , <code>forecast_gp()</code> , <code>predict_sparse_gp()</code> , <code>predict_heteroscedastic_gp()</code> , or <code>predict_latent_gp()</code> .
<code>x</code>	A summary object.

Details

`predict()` returns exactly what the package's prediction function for the model returns. For models from `fit_time_series_gp()`, `newdata` are time values and `predict()` calls `forecast_gp()`; this includes every state-space model with time metadata.

`logLik()` returns the log marginal likelihood $\log p(y \mid X, \hat{\theta})$: `log_marginal_likelihood()` for exact and state-space models and, for sparse models, the FITC approximate marginal likelihood or the VFE lower bound. This is a type-II likelihood, with the latent function integrated out and the hyperparameters $\hat{\theta}$ plugged in. Its `df` attribute counts the hyperparameters estimated from the data: the parameters optimized by `optimize_gp()` or `optimize_sparse_gp()`, and 0 for models whose hyperparameters were supplied, plus the mean coefficients when they are estimated. For restricted likelihoods (REML estimates and vague coefficient priors) the `nobs` attribute is $n - p$, as for restricted likelihoods elsewhere in R, and such likelihoods are comparable only between models with the same mean. `stats::AIC()` and `stats::BIC()` therefore work, but comparing kernels by them is a heuristic: it is not an ordinary maximum-likelihood comparison, and the penalty does not account for the flexibility of the kernel. Heteroscedastic models have no `logLik()` method, because their noise function is a plug-in estimate rather than a likelihood maximizer. For latent models it is the Laplace approximation to the marginal likelihood, and `df` counts the parameters optimized by `optimize_latent_gp()` or the mean coefficients estimated by `fit_latent_gp()`. For state-space models `df` counts the parameters optimized by `optimize_time_series_gp()`.

`fitted()` returns the posterior mean of the latent function at the training inputs, and `residuals()` returns the observed responses minus those fitted values. For exact models the fitted values are $y - (\Sigma_\varepsilon + \delta I)\alpha$, which equals $m(X) + K\alpha$ without forming K ; δ is the numerical jitter. For state-space models they are the Kalman-smoother means. For latent models `fitted()` returns the posterior mode $\hat{f}$ of the latent function, and there is no `residuals()` method, because the responses are not on the scale of f .

`summary()` collects the training size and dimension, kernel, mean, noise, log marginal likelihood, optimization status, and numerical status, and prints them compactly.

Value

`predict()`: a prediction object. `logLik()`: an object of class `logLik`. `nobs()`: the number of training observations. `fitted()` and `residuals()`: numeric vectors with one value per training observation. `summary()`: an object of class `summary.gaussianprocesses_model`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 20)
y <- sin(2 * x) + 0.1 * cos(7 * x)
model <- optimize_gp(
  x,
  y,
  kernel = rbf_kernel(),
  noise_variance = 0.05,
  n_starts = 1
)

summary(model)
predict(model, c(-1, 0.5))$mean
logLik(model)
AIC(model)
head(residuals(model))
```

`gp_numerical_diagnostics`*Summarize numerical conditioning of a fitted GP*

Description

Reports numerical quantities separately from statistical fit diagnostics. In particular, observation-noise variance and numerical jitter remain distinct.

Usage

```
gp_numerical_diagnostics(
  model,
  condition_tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

`model` A fitted `gaussianprocesses_model`.

`condition_tolerance` Positive reciprocal-condition threshold used only to flag a potentially ill-conditioned stabilized covariance matrix.

Value

A named list of numerical diagnostics, including:

`observation_noise_variance` The statistical noise variance.

`numerical_jitter` The diagonal jitter δ that the factorization of the training covariance C needed.

`covariance_scale` The mean of the diagonal of C .

`relative_jitter` `numerical_jitter / covariance_scale`.

`cholesky_attempts` Factorization attempts, 1 if no jitter was needed.

`reciprocal_condition_number, condition_number` The 1-norm condition estimate of $C + \delta I$ and its reciprocal.

`potentially_ill_conditioned` Whether the reciprocal condition number is below `condition_tolerance`.

`observation_types` A data frame with one row per observation type ("value", or "derivative x1", ... for derivative observations): the number of observations, the mean diagonal of their block of C , and the reciprocal condition number of that block with the jitter. Derivative blocks scale with $1/\ell^2$, so their conditioning can differ from that of the values.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
# Duplicated inputs without observation noise need numerical jitter, which is
# reported separately from the (zero) noise variance.
model <- fit_gp(c(0, 0, 1), c(1, 1, 0), kernel = rbf_kernel(), noise_variance = 0)

diagnostics <- gp_numerical_diagnostics(model)
diagnostics[c(
  "observation_noise_variance",
  "numerical_jitter",
  "relative_jitter",
  "cholesky_attempts"
)]
```

gp_profile_likelihood *Profile likelihood of one hyperparameter*

Description

Fixes one estimated hyperparameter at each of a grid of values and maximizes the log marginal likelihood over the others.

Usage

```
gp_profile_likelihood(
  model,
  parameter,
  values = NULL,
  n_values = 11L,
  step = 1e-04
)
```

Arguments

model	A fitted gaussianprocesses_model, normally from <code>optimize_gp()</code> .
parameter	Path of the parameter, as in <code>kernel_parameters(model\$kernel, flatten = TRUE)</code> , or "noise_variance".
values	Values of the parameter on its natural scale. If NULL, a grid of <code>n_values</code> points spanning three standard errors either side of the estimate on the coordinate scale, or one unit when the standard error is not available.
n_values	Number of grid points when values is NULL.
step	Step of the central differences of the gradient, on the optimizer's coordinate scale.

Details

At each value the other estimated hyperparameters start from their estimates and are re-optimized with L-BFGS-B and the analytical gradient, within the bounds that `optimize_gp()` used. deviance is twice the drop in log marginal likelihood from the estimate, and converged whether the other parameters reached a stationary point: the optimizer converged, or the gradient there is below 10^{-4} . Under the usual asymptotics values with deviance below `qchisq(0.95, 1)` form an approximate 95% interval. A flat profile shows a poorly identified parameter directly.

Value

A data frame of class `gaussianprocesses_profile_likelihood` with one row per value: `value`, `log_marginal_likelihood`, `deviance`, and `converged`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
set.seed(1)
x <- seq(0, 10, length.out = 40)
model <- optimize_gp(x, sin(x) + rnorm(40, sd = 0.2), rbf_kernel(),
  noise_variance = 0.1, n_starts = 1)

gp_profile_likelihood(model, "length_scale", n_values = 5)
```

gp_scores

Proper scoring rules for Gaussian predictions

Description

Scores Gaussian predictive distributions against observations with proper scoring rules, interval scores and coverage across a grid of levels, and the probability integral transform. `gp_holdout_scores()` scores a fitted model on held-out data and `gp_loo_scores()` on its exact leave-one-out predictions.

Usage

```
gp_scores(
  observed,
  mean,
  sd,
  interval_levels = c(0.5, 0.8, 0.9, 0.95),
  reference_mean = NULL,
  reference_variance = NULL
)

gp_holdout_scores(
  model,
  x,
  y,
  interval_levels = c(0.5, 0.8, 0.9, 0.95),
  n_bins = 10L,
  seed = NULL,
  ...
)

gp_loo_scores(model, interval_levels = c(0.5, 0.8, 0.9, 0.95))
```

Arguments

observed	Numeric vector of observations.
mean, sd	Predictive means and standard deviations, one per observation; the standard deviations must be positive.
interval_levels	Levels of the central prediction intervals whose coverage and interval score are reported: the calibration curve.
reference_mean, reference_variance	Mean and variance of the reference Gaussian for the MSLL, usually those of the training responses. Without them the MSLL is NA.
model	A fitted model of any class.
x	Held-out inputs, as accepted by the model's <code>stats::predict()</code> method: time values for time-series models.
y	Held-out observations, one per row of x.
n_bins	Number of probability bins for the reliability of classification scores (<code>gp_classification_scores()</code>).
seed	Optional seed for the randomized PIT values of count scores (<code>gp_count_scores()</code>).
...	Further arguments to <code>stats::predict()</code> , such as <code>observation_noise_variance</code> for models fitted with observation-specific noise.

Details

With $z_i = (y_i - \mu_i)/\sigma_i$, the scores are

log score $\log \varphi(z_i) - \log \sigma_i$, the log predictive density. Higher is better.

CRPS $\sigma_i[z_i(2\Phi(z_i) - 1) + 2\varphi(z_i) - 1/\sqrt{\pi}]$, the continuous ranked probability score of Gneiting and Raftery (2007), in closed form. Lower is better; it is in the units of y .

interval score $(u - l) + \frac{2}{\alpha}(l - y)\mathbf{1}\{y < l\} + \frac{2}{\alpha}(y - u)\mathbf{1}\{y > u\}$ for the central interval $[l, u]$ of level $1 - \alpha$. Lower is better.

PIT $\Phi(z_i)$, uniform on $[0, 1]$ when the forecasts are calibrated.

SMSE The mean squared error divided by the variance of the observations (Rasmussen and Williams, 2006, section 2.5).

MSLL The mean of $-\log p(y_i)$ minus the same under the reference Gaussian; negative values beat that reference.

The log score, CRPS, and interval score are proper: in expectation they are optimized by the true predictive distribution.

`gp_holdout_scores()` predicts with `stats::predict()`, so it accepts every model class, and scores the observation predictive distribution, including noise. Its MSLL reference is the mean and variance of the training responses. `gp_loo_scores()` scores the exact leave-one-out predictions of `loo_gp()` against the same reference.

For latent models (`fit_latent_gp()`), `gp_holdout_scores()` scores the predictive distribution of new responses: with a Gaussian likelihood as above, with a Bernoulli likelihood by `gp_classification_scores()`, whose log loss uses the predictive log probabilities of `likelihood_predictive()`, and with a Poisson likelihood by `gp_count_scores()`, with the exposure of the predictions.

Value

An object of class `gaussianprocesses_scores` with

`summary` Named numeric vector: `n`, `log_score` (mean log predictive density), `crps` (mean), `smse`, `msll`, `pit_mean`, and `pit_variance` (1/12 for uniform PIT values).

`calibration` Data frame with one row per interval level: the `level`, empirical coverage, mean width, and mean `interval_score`.

`pointwise` Data frame with one row per observation: `observed`, `mean`, `sd`, `standardized_error`, `pit`, `log_score`, and `crps`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

References

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477), 359–378.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. Section 2.5.

See Also

[gp_calibration_diagnostics\(\)](#), [forecast_metrics\(\)](#).

Examples

```
x <- seq(0, 6, length.out = 40)
y <- sin(x) + rnorm(40, sd = 0.1)
training <- seq(1, 40, by = 2)

model <- fit_gp(x[training], y[training], rbf_kernel(),
  noise_variance = 0.01)
scores <- gp_holdout_scores(model, x[-training], y[-training])
scores
scores$calibration

gp_loo_scores(model)$summary
```

`gp_simulation_scenario`*Generate standard GP simulation scenarios*

Description

Creates deterministic input designs and seeded observations for common validation cases used throughout the benchmark suite.

Usage

```
gp_simulation_scenario(  
  scenario = c("smooth_1d", "near_singular", "ard_2d", "heteroscedastic_1d"),  
  n = 60L,  
  seed = 1729L  
)
```

Arguments

scenario	One of "smooth_1d", "near_singular", "ard_2d", or "heteroscedastic_1d".
n	Number of observations.
seed	Non-negative integer seed.

Value

A `gaussianprocesses_simulation` with a `scenario` element.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
simulation <- gp_simulation_scenario("ard_2d", n = 40)  
simulation  
simulation$kernel
```

gp_stack_outputs	<i>Stack several outputs into one input matrix</i>
------------------	--

Description

Converts inputs and one response column per output into the long format of multi-output models: one row per observed (input, output) pair, with the output index as the last input column.

Usage

```
gp_stack_outputs(x, y = NULL, outputs = NULL)
```

Arguments

x	Numeric vector or matrix of inputs, one row per observation.
y	Numeric matrix or data frame with one column per output and one row per row of x; NA marks an output that was not observed there, so outputs may be observed at different inputs (heterotopic data). If NULL, every row of x is stacked for each output in outputs, for prediction.
outputs	Output indices to stack when y is NULL.

Value

A list with x, the stacked input matrix whose last column, output, holds the output index; y, the stacked responses (only when y is given); output, the output index of each row; and row, the row of the original x. Rows are ordered by output, then by row.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[coregionalization_kernel\(\)](#).

Examples

```
x <- c(0, 1, 2)
y <- cbind(c(1.0, NA, 0.4), c(0.2, 0.3, NA))
stacked <- gp_stack_outputs(x, y)
stacked$x
stacked$y

gp_stack_outputs(c(0.5, 1.5), outputs = 1:2)$x
```

gp_time_index

*Build an explicit Gaussian-process time index***Description**

Converts numeric, Date, or POSIXt values to elapsed numeric time while retaining origin, unit, and input-class metadata. Input order is never changed: non-increasing time values are rejected.

Usage

```
gp_time_index(time, unit = "auto")
```

Arguments

time	Numeric, Date, or POSIXt vector in strictly increasing order.
unit	Time unit for Date or POSIXt inputs. "auto" uses days for Date values, seconds for POSIXt values, and the native scale for numeric inputs.

Details

Time values are never sorted silently. Non-increasing input is rejected so the temporal order supplied by the user remains explicit.

Value

A numeric vector of elapsed time with class `gaussianprocesses_time_index` and time metadata in attributes.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
gp_time_index(as.Date("2026-01-01") + c(0, 1, 7))

gp_time_index(
  as.POSIXct("2026-01-01 00:00", tz = "UTC") + c(0, 3600, 7200),
  unit = "hours"
)
```

```
initialize_coregionalization
```

Initialize a coregionalization kernel from the data

Description

Builds a [coregionalization_kernel\(\)](#) whose B is the empirical covariance between the outputs, a starting point from which optimization can move W .

Usage

```
initialize_coregionalization(output, y, x = NULL, rank = 1L, n_outputs = NULL)
```

Arguments

output	Output index of each observation, from 1 to n_outputs.
y	Numeric response of each observation.
x	Optional inputs of each observation, a vector or matrix without the output-index column. Covariances between two outputs are estimated from the inputs at which both are observed; without x, or with fewer than three shared inputs, they start at zero.
rank	Rank R of W .
n_outputs	Number of outputs; by default the largest index in output.

Details

The empirical covariance $\hat{B}$ of the centred responses is made positive semidefinite by setting negative eigenvalues to zero. W is taken from its leading R eigenvectors, scaled by the square roots of their eigenvalues, and κ from what remains on the diagonal, with at least 5% of each output's variance. The empirical variances include the observation noise, so B starts too large on the diagonal; optimization corrects that.

Value

A `gaussianprocesses_kernel`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[coregionalization_kernel\(\)](#), [gp_stack_outputs\(\)](#).

Examples

```

set.seed(1)
x <- seq(0, 5, length.out = 20)
y <- cbind(sin(x), 2 * sin(x) + 0.1) + rnorm(40, sd = 0.1)
stacked <- gp_stack_outputs(x, y)
kernel <- initialize_coregionalization(stacked$output, stacked$y,
                                     x = stacked$x[, 1])
coregionalization_matrix(kernel)

```

```
initialize_spectral_mixture
```

Initialise a spectral-mixture kernel from the data's spectrum

Description

Builds a `spectral_mixture_kernel()` whose components sit at the main peaks of the empirical spectrum of one-dimensional data, a starting point for `optimize_gp()`.

Usage

```
initialize_spectral_mixture(x, y, n_components = 2L)
```

Arguments

<code>x</code>	Numeric vector of inputs, regularly or irregularly spaced.
<code>y</code>	Numeric vector of responses.
<code>n_components</code>	Number of components Q .

Details

The spectrum of the centred responses is the periodogram (`stats::spec.pgram()`, without taper) when the inputs are regularly spaced, and the Lomb-Scargle periodogram otherwise, computed on a grid four times finer than the frequency resolution $1/T$, for an input range T , up to half the reciprocal median spacing. A mixture of Q Gaussians and a uniform background is fitted to the spectrum, as a density over frequency weighted by the power, by EM, starting from the Q largest local maxima. The background absorbs the flat spectrum of observation noise, which would otherwise widen the components.

Component q takes the mean and variance of Gaussian q as its frequency and spectral variance, and the share of the variance of y that the Gaussian explains as its weight; the background share is left to the noise. Spectral variances are at least $(1/(4T))^2$, the order of a spectral peak's width.

The likelihood of a spectral mixture is strongly multimodal, and this initialisation is only a starting point: optimize with several starts, for example `n_starts = 5`, which displace each frequency by one resolution bin in turn.

Value

A spectral-mixture kernel specification with `n_components` components.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[spectral_mixture_kernel\(\)](#).

Examples

```
set.seed(1)
x <- 0:199
y <- 2 * sin(2 * pi * 0.1 * x) + sin(2 * pi * 0.27 * x) +
  rnorm(200, sd = 0.5)

kernel <- initialize_spectral_mixture(x, y, n_components = 2)
kernel
```

kernel_diagonal	<i>Evaluate only the diagonal of a kernel matrix</i>
-----------------	--

Description

Computes $\text{diag}(K(x, x))$ without constructing the full covariance matrix. This is particularly useful in sparse Gaussian-process approximations.

Usage

```
kernel_diagonal(kernel, x)
```

Arguments

kernel	A Gaussian-process kernel specification.
x	Numeric vector or matrix of observations.

Value

A numeric vector with one prior variance per observation.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- sum_kernel(rbf_kernel(variance = 2), linear_kernel())
x <- cbind(1:3, 0:2)

kernel_diagonal(kernel, x)
all.equal(kernel_diagonal(kernel, x), diag(evaluate_kernel(kernel, x)))
```

kernel_gradient	<i>Derivatives of a covariance matrix with respect to log hyperparameters</i>
-----------------	---

Description

Computes analytical derivatives of the covariance matrix $K(x, y; \theta)$ with respect to the logarithm of every kernel parameter, $\partial K / \partial \log \theta_j$. These are the building blocks for gradient-based marginal-likelihood optimization, which works in log-parameter space.

Usage

```
kernel_gradient(kernel, x, y = NULL)
```

Arguments

kernel	A Gaussian-process kernel specification.
x	Numeric vector or matrix of observations.
y	Optional numeric vector or matrix. If NULL, derivatives of the covariance of x with itself are returned.

Details

Every kernel parameter is strictly positive, so derivatives are taken with respect to $\eta_j = \log \theta_j$. The derivative with respect to the parameter on its natural scale is $\partial K / \partial \theta_j = \theta_j^{-1} \partial K / \partial \eta_j$.

Elements are named and ordered exactly like `kernel_parameters(kernel, flatten = TRUE)`, including indexed ARD paths such as `kernel1.length_scale[2]`, so they can be matched to optimizer parameters by name. A scalar length scale shared by several input dimensions has a single derivative.

Leaf derivatives are closed-form expressions in the covariance matrix and the scaled input differences. Sums differentiate term by term, products use the product rule (without dividing by any factor, so exact zeros are safe), and scaled kernels use the chain rule. The derivations are in vignette("v07-kernel-derivatives", package = "gaussianprocesses").

Value

A named list with one numeric matrix per flattened kernel parameter. Each matrix has `nrow(x)` rows and `nrow(y)` columns (or `nrow(x)` columns when y is NULL).

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_input_gradient\(\)](#) for derivatives with respect to the inputs.

Examples

```
kernel <- sum_kernel(
  rbf_kernel(variance = 1.5, length_scale = c(0.5, 2)),
  white_noise_kernel(variance = 0.1)
)
x <- cbind(c(0, 0.5, 1.5), c(1, 0, 2))

gradient <- kernel_gradient(kernel, x)
names(gradient)

# Natural-scale derivative with respect to the first ARD length scale.
gradient[["kernel1.length_scale[1]"]] / 0.5
```

kernel_input_gradient *Derivatives of a covariance matrix with respect to the inputs*

Description

Computes $\partial k(x_i, y_j) / \partial x_{i,d}$ for every input dimension d : the covariance between the derivative of the process at x_i and its value at y_j .

Usage

```
kernel_input_gradient(kernel, x, y = NULL)
```

Arguments

kernel	A Gaussian-process kernel specification.
x	Numeric vector or matrix of inputs; the derivatives are taken with respect to these.
y	Optional numeric vector or matrix. If NULL, x is used.

Details

Input derivatives exist only for mean-square differentiable kernels. The Matérn-1/2 and white-noise kernels, and any sum or product that contains one, are not, and raise an error of class `gaussianprocesses_smoothness_error`. The Matérn-3/2 process is differentiable once, the Matérn-5/2 process twice, and the RBF, rational-quadratic, periodic, and linear processes infinitely often.

Derivatives are closed-form expressions for every kernel, written so that coincident inputs need no limit; sums, products, and scaled kernels follow the sum and product rules. `predict_gradient_gp()` uses them to predict the gradient of the latent function. The derivations are in vignette("v07-kernel-derivatives", package = "gaussianprocesses").

Value

A list with one numeric matrix per input dimension. Element `d` has `nrow(x)` rows and `nrow(y)` columns and holds $\partial k(x_i, y_j) / \partial x_{i,d}$.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_gradient\(\)](#) for derivatives with respect to the hyperparameters.

Examples

```
kernel <- rbf_kernel(variance = 1, length_scale = 0.5)
x <- c(0, 0.25, 1)

gradient <- kernel_input_gradient(kernel, x)
gradient[[1]]

# Compare with a central difference in the first input.
h <- 1e-6
(evaluate_kernel(kernel, x + h, x) -
 evaluate_kernel(kernel, x - h, x)) / (2 * h)
```

kernel_linear

Linear covariance kernel

Description

Computes a linear kernel from inner products between observations.

Usage

```
kernel_linear(x, y = NULL, variance = 1)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x. If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[linear_kernel\(\)](#) creates the same kernel as a reusable specification, which can be combined, optimized, and used in models.

Examples

```
x <- cbind(c(1, 2, 3), c(0, 1, 0))
kernel_linear(x, variance = 0.5)
```

kernel_matern	<i>Matérn covariance kernels</i>
---------------	----------------------------------

Description

Computes Matérn kernels for smoothness parameters $\nu = 1/2, 3/2$, and $5/2$. A scalar length scale gives an isotropic distance; one length scale per input dimension gives ARD.

Usage

```
kernel_matern12(x, y = NULL, variance = 1, length_scale = 1)
kernel_matern32(x, y = NULL, variance = 1, length_scale = 1)
kernel_matern52(x, y = NULL, variance = 1, length_scale = 1)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x. If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector with one value per input dimension. A vector enables ARD.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[matern_kernel_specs](#) for the same kernels as reusable specifications, which can be combined, optimized, and used in models.

Examples

```
x <- seq(0, 2, by = 0.5)

# Smoothness increases from nu = 1/2 to nu = 5/2.
kernel_matern12(x, length_scale = 0.7)[1, ]
kernel_matern32(x, length_scale = 0.7)[1, ]
kernel_matern52(x, length_scale = 0.7)[1, ]
```

kernel_parameters	<i>Extract kernel parameters</i>
-------------------	----------------------------------

Description

Extract kernel parameters

Usage

```
kernel_parameters(kernel, flatten = FALSE)
```

Arguments

kernel	A Gaussian-process kernel specification.
flatten	If FALSE, return a nested list matching the kernel structure. If TRUE, return a named numeric vector with stable parameter paths suitable for optimization. Vector parameters use indexed names such as length_scale[1].

Details

Flat parameter names encode the position of each nested kernel. Scalar parameters use paths such as `kernel1.variance`; vector-valued ARD parameters use paths such as `kernel1.length_scale[1]` and `kernel1.length_scale[2]`. These names remain stable when parameter values are updated and can be passed directly to optimization routines.

A large ARD length scale means covariance changes slowly along that input dimension, while a small value means covariance changes more rapidly. This is a model sensitivity parameterization and should not be interpreted as a causal importance measure.

Value

A nested list or named numeric vector.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- sum_kernel(
  rbf_kernel(length_scale = c(1, 2)),
  white_noise_kernel(variance = 0.1)
)

str(kernel_parameters(kernel))
kernel_parameters(kernel, flatten = TRUE)
```

kernel_periodic	<i>Periodic covariance kernel</i>
-----------------	-----------------------------------

Description

Computes a periodic kernel with a shared period and either an isotropic length scale or one ARD length scale per input dimension.

Usage

```
kernel_periodic(x, y = NULL, variance = 1, length_scale = 1, period = 1)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x. If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector with one value per input dimension. A vector enables ARD.
period	Positive period shared across input dimensions.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[periodic_kernel\(\)](#) creates the same kernel as a reusable specification, which can be combined, optimized, and used in models.

Examples

```
x <- c(0, 0.25, 1, 2)
covariance <- kernel_periodic(x, length_scale = 0.8, period = 1)

# Inputs a whole number of periods apart are perfectly correlated.
covariance[1, ]
```

kernel_rational_quadratic

Rational-quadratic covariance kernel

Description

The rational-quadratic kernel can be interpreted as a scale mixture of squared-exponential kernels with different length scales. Scalar and ARD length-scale parameterizations are both supported.

Usage

```
kernel_rational_quadratic(
  x,
  y = NULL,
  variance = 1,
  length_scale = 1,
  alpha = 1
)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x. If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector with one value per input dimension. A vector enables ARD.
alpha	Positive scale-mixture parameter.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[rational_quadratic_kernel\(\)](#) creates the same kernel as a reusable specification, which can be combined, optimized, and used in models.

Examples

```
x <- seq(0, 2, by = 0.5)
kernel_rational_quadratic(x, length_scale = 0.7, alpha = 0.5)

# As alpha grows, the kernel approaches the RBF kernel.
all.equal(
  kernel_rational_quadratic(x, alpha = 1e6),
  kernel_rbf(x),
  tolerance = 1e-5
)
```

kernel_rbf	<i>Squared-exponential covariance kernel</i>
------------	--

Description

Computes the squared-exponential (radial basis function) kernel between rows of x and y . A scalar length scale gives an isotropic kernel; one length scale per input dimension gives automatic relevance determination (ARD).

Usage

```
kernel_rbf(x, y = NULL, variance = 1, length_scale = 1)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x . If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector with one value per input dimension. A vector enables ARD.

Value

A numeric covariance matrix with $nrow(x)$ rows and $nrow(y)$ columns. Numeric vectors are treated as one-dimensional observations.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[rbf_kernel\(\)](#) creates the same kernel as a reusable specification, which can be combined, optimized, and used in models.

Examples

```
x <- seq(0, 2, by = 0.5)
kernel_rbf(x, variance = 2, length_scale = 0.7)

# ARD: one length scale per input dimension.
x2 <- cbind(c(0, 1, 2), c(0, 0, 1))
kernel_rbf(x2, length_scale = c(0.5, 3))
```

```
# Cross-covariance between two sets of inputs.
kernel_rbf(x, c(0.25, 1.75))
```

kernel_white_noise	<i>White-noise covariance kernel</i>
--------------------	--------------------------------------

Description

Returns variance when two observations are exactly identical and zero otherwise. For `kernel_white_noise(x)`, the diagonal therefore equals the requested white-noise variance.

Usage

```
kernel_white_noise(x, y = NULL, variance = 1)
```

Arguments

x	Numeric vector or matrix. Matrix rows are observations and columns are input dimensions.
y	Optional numeric vector or matrix with the same number of input dimensions as x. If NULL, computes the covariance of x with itself.
variance	Positive marginal variance.

Value

A numeric covariance matrix.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[white_noise_kernel\(\)](#) creates the same kernel as a reusable specification, which can be combined, optimized, and used in models.

Examples

```
# Only coincident inputs covary.
kernel_white_noise(c(0, 0, 1), variance = 0.1)
```

likelihood_predictive *Predictive distribution of new responses*

Description

Integrates a likelihood against a Gaussian distribution of the latent value, $\int p(y_* | f_*) N(f_* | \mu, \sigma^2) df_*$: the predictive distribution of a new response given the posterior of the latent function at its input.

Usage

```
likelihood_predictive(
  likelihood,
  mean,
  variance,
  observed = NULL,
  exposure = NULL,
  quadrature_order = 40L
)
```

Arguments

likelihood	A likelihood specification (gp_likelihoos).
mean, variance	Mean and variance of the latent value, one per prediction.
observed	Optional responses whose log predictive density is returned.
exposure	Optional positive exposure for Poisson likelihoods: one value or one per response.
quadrature_order	Number of Gauss-Hermite nodes for integrals without a closed form.

Details

Closed forms are used where they exist: the Gaussian likelihood adds its variance, the probit link gives $\Pr(y_* = 1) = \Phi(\mu/\sqrt{1 + \sigma^2})$, and the Poisson mean and variance are those of a Poisson with a lognormal rate, $Ee^{\mu + \sigma^2/2}$ and $E[y_*] + E^2e^{2\mu + \sigma^2}(e^{\sigma^2} - 1)$. The logistic probability and the Poisson and logistic predictive densities use adaptive Gauss-Hermite quadrature (Liu and Pierce, 1994). With $\hat{f}$ the mode of $\psi(f) = \log p(y | f) + \log N(f | \mu, \sigma^2)$, found by Newton's method, and $s = (-\psi''(\hat{f}))^{-1/2}$,

$$\int p(y | f) N(f | \mu, \sigma^2) df \approx \sqrt{2s} \sum_j w_j e^{t_j^2} e^{\psi(\hat{f} + \sqrt{2s}t_j)},$$

computed in logs. Centring the nodes on the integrand rather than on μ matters when the observation lies in the tail of the latent distribution, as for large counts.

Quadrature converges more slowly as σ^2 grows, because the logistic function has poles at $\pm i\pi$. With the default 40 nodes, the logistic probability is accurate to 10^{-10} for σ^2 up to 3; increase quadrature_order for wider latent distributions.

Value

A list with the predictive mean and variance of y_* , probability, $\Pr(y_* = 1)$, for Bernoulli likelihoods, and log_density of observed when it is given.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Liu, Q. and Pierce, D. A. (1994). A note on Gauss-Hermite quadrature. *Biometrika*, 81(3), 624–629.

Examples

```
likelihood_predictive(bernoulli_likelihood("probit"), mean = c(-1, 0, 2),
  variance = c(0.5, 1, 4))$probability

likelihood_predictive(poisson_likelihood(), mean = log(3), variance = 0.2,
  observed = 4, exposure = 2)
```

linear_kernel	Create a linear kernel specification
---------------	--------------------------------------

Description

Create a linear kernel specification

Usage

```
linear_kernel(variance = 1)
```

Arguments

variance Positive variance multiplier.

Value

An object of class gaussianprocesses_kernel.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_linear\(\)](#) computes the same covariance directly from parameter values, without a kernel specification.

Examples

```
# A linear trend component.
evaluate_kernel(linear_kernel(variance = 0.1), 1:4)
```

```
log_marginal_likelihood
```

Gaussian-process log marginal likelihood

Description

Computes the exact Gaussian log marginal likelihood from a fitted model using its stored Cholesky factor and solve vector.

Usage

```
log_marginal_likelihood(model)
```

Arguments

model	A fitted <code>gaussianprocesses_model</code> , <code>gaussianprocesses_latent_model</code> , or <code>gaussianprocesses_state_space_model</code> .
-------	---

Details

If numerical jitter was required during fitting, the stored Cholesky factor corresponds to the jitter-stabilized covariance used by the numerical calculation. Jitter remains distinct from statistical observation noise.

For a mean with estimated coefficients, the value is the profile likelihood, at the generalized least-squares estimate $\hat{\beta}$ (`estimate_coefficients("ml")`), or the restricted likelihood (`estimate_coefficients("reml")`). For coefficients with a Gaussian prior it is the marginal likelihood of the Gaussian process with covariance $k(x, x') + h(x)^\top B h(x')$, and for a vague prior it is the restricted likelihood. [mean_coefficients](#) gives the formulas.

For a latent model from [fit_latent_gp\(\)](#) or [optimize_latent_gp\(\)](#) the value is the Laplace approximation $\log q(y \mid X, \theta)$. For a state-space model from [fit_time_series_gp\(\)](#) it is the exact log marginal likelihood from the Kalman filter's prediction errors.

Value

A single finite numeric log marginal likelihood.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 15)
y <- sin(2 * x)

# Compare candidate length scales by their marginal likelihood.
sapply(c(0.2, 0.6, 2), function(length_scale) {
  log_marginal_likelihood(
    fit_gp(x, y, rbf_kernel(length_scale = length_scale), noise_variance = 0.01)
  )
})
```

log_marginal_likelihood_gradient

Gradient of the Gaussian-process log marginal likelihood

Description

Computes the exact gradient of [log_marginal_likelihood\(\)](#) with respect to the logarithm of every kernel parameter and, for models with a single homoscedastic noise variance, the logarithm of that noise variance.

Usage

```
log_marginal_likelihood_gradient(model)
```

Arguments

model	A fitted <code>gaussianprocesses_model</code> , <code>gaussianprocesses_latent_model</code> , or <code>gaussianprocesses_state_space_model</code> .
-------	---

Details

Let C be the observation covariance and $\alpha = C^{-1}(y - m(X))$. For $\eta_j = \log \theta_j$,

$$\frac{\partial \log p(y \mid X, \theta)}{\partial \eta_j} = \frac{1}{2} \text{tr} \left((\alpha \alpha^\top - C^{-1}) \frac{\partial C}{\partial \eta_j} \right).$$

For kernel parameters, $\partial C / \partial \eta_j$ is given by [kernel_gradient\(\)](#); for the noise variance σ_n^2 it is $\sigma_n^2 I$.

The trace requires every entry of the precision matrix C^{-1} . They are computed once from the stored Cholesky factor with [chol2inv\(\)](#), the Cholesky solve of $CX = I$; C is never passed to

`solve()` or a general-purpose inverse, and the precision matrix is not used to solve linear systems. The alternative, two triangular solves per parameter, never forms C^{-1} ; against an exact reference it was equally accurate for condition numbers up to 10^{12} and 4 to 18 times slower (inst/benchmarks/gradient-numeric.R). As in `log_marginal_likelihood()`, any numerical jitter used during fitting is part of the factorized covariance and is treated as a constant.

Models fitted with observation-specific noise variances have no noise parameter, so only kernel derivatives are returned for them.

For a latent model the gradient is that of the Laplace approximation, including the implicit dependence of the mode on the parameters (`optimize_latent_gp()`), in the optimizer coordinates of the kernel parameters and of the likelihood parameters.

Value

A named numeric vector. Names follow `kernel_parameters(model$kernel, flatten = TRUE)`, followed by `noise_variance` for homoscedastic models, or by `likelihood.<name>` for the parameters of the likelihood of a latent model.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 15)
model <- fit_gp(
  x,
  sin(x),
  kernel = rbf_kernel(length_scale = 0.8),
  noise_variance = 0.01
)
log_marginal_likelihood_gradient(model)
```

 loo_gp

Exact leave-one-out Gaussian-process diagnostics

Description

Computes exact leave-one-out (LOO) predictive summaries from a fitted exact Gaussian-process model without refitting the model once per observation. The calculation uses the fitted precision diagonal implied by the stored Cholesky factor.

Usage

```
loo_gp(model, interval_level = 0.95)
```

Arguments

`model` A fitted `gaussianprocesses_model`.

`interval_level` Probability level for LOO prediction intervals.

Details

When numerical jitter was required during fitting, the LOO calculation necessarily reflects the stabilized covariance used in that factorization; the amount of jitter is reported separately by `gp_numerical_diagnostics()`.

Estimated mean coefficients stay at their full-data estimate, as in `predict_gp()`. Marginalized coefficients are integrated out as in the posterior: the precision diagonal is that of $P_A = C^{-1} - C^{-1}H(B^{-1} + H^TC^{-1}H)^{-1}H^TC^{-1}$, so with a vague prior each prediction re-estimates the coefficients without the left-out observation.

Value

A data frame with observed values, LOO predictive means and variances, residuals, standardized residuals, probability integral transform (PIT) values, interval bounds, coverage indicators, and pointwise log predictive densities.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-2, 2, length.out = 20)
model <- fit_gp(
  x,
  sin(2 * x),
  kernel = rbf_kernel(length_scale = 0.6),
  noise_variance = 0.01
)

loo <- loo_gp(model)
head(loo[, c("observed", "mean", "sd", "pit", "covered")])
```

matern_kernel_specs	Create Matérn kernel specifications
---------------------	-------------------------------------

Description

Create Matérn kernel specifications

Usage

```
matern12_kernel(variance = 1, length_scale = 1)
```

```
matern32_kernel(variance = 1, length_scale = 1)
```

```
matern52_kernel(variance = 1, length_scale = 1)
```

Arguments

variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector. A vector supplies one ARD length scale per input dimension and is validated against the data when the kernel is evaluated.

Value

An object of class `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_matern12\(\)](#), [kernel_matern32\(\)](#), and [kernel_matern52\(\)](#) compute the same covariances directly from parameter values.

Examples

```
matern52_kernel(variance = 1, length_scale = 0.8)

# ARD length scales become indexed parameter paths.
kernel_parameters(matern32_kernel(length_scale = c(1, 2)), flatten = TRUE)
```

mean_coefficients	<i>Treatments of mean-function coefficients</i>
-------------------	---

Description

A parametric mean $m(x) = h(x)^T \beta$ has coefficients β that are fixed, estimated, or marginalized. Pass the result of these functions as the coefficients argument of [linear_mean\(\)](#), [polynomial_mean\(\)](#), [basis_mean\(\)](#), or as the value of [constant_mean\(\)](#); a numeric vector fixes the coefficients instead.

Usage

```
estimate_coefficients(method = c("ml", "reml"))

coefficient_prior(mean = 0, covariance = NULL)
```

Arguments

method	"ml" or "reml": the likelihood that <code>log_marginal_likelihood()</code> reports and <code>optimize_gp()</code> maximizes.
mean	Prior mean b of the coefficients: one value, recycled, or one per coefficient.
covariance	Prior covariance B : NULL for the vague limit $B^{-1} \rightarrow 0$; a positive number or vector, for a diagonal covariance; or a symmetric positive-definite matrix.

Details

Write H for the matrix of basis rows $h(x_i)^\top$, $C = K + \Sigma$ for the observation covariance, and p for the number of coefficients.

Estimated. The coefficients are the generalized least-squares estimate $\hat{\beta} = (H^\top C^{-1} H)^{-1} H^\top C^{-1} y$, which maximizes the likelihood for given hyperparameters. Predictions plug in $\hat{\beta}$ and do not include its uncertainty. With method = "ml" the likelihood is the profile likelihood, the Gaussian likelihood at $\hat{\beta}$. With method = "reml" it is the restricted likelihood

$$\ell_R = -\frac{1}{2} y^\top P y - \frac{1}{2} \log |C| - \frac{1}{2} \log |H^\top C^{-1} H| - \frac{n-p}{2} \log 2\pi,$$

with $P = C^{-1} - C^{-1} H (H^\top C^{-1} H)^{-1} H^\top C^{-1}$, which accounts for the p degrees of freedom used by $\hat{\beta}$ and so estimates variances with less bias.

Marginalized. With the prior $\beta \sim N(b, B)$, the model is exactly a Gaussian process with mean $h(x)^\top b$ and covariance $k(x, x') + h(x)^\top B h(x')$ (Rasmussen and Williams, 2006, section 2.7). Predictions use the posterior mean $\tilde{\beta} = (B^{-1} + H^\top C^{-1} H)^{-1} (H^\top C^{-1} y + B^{-1} b)$ and add the coefficient uncertainty $R_*(B^{-1} + H^\top C^{-1} H)^{-1} R_*^\top$, $R_* = H_* - K_{*X} C^{-1} H$, to the predictive covariance; posterior draws include it too. The likelihood is the marginal likelihood of that Gaussian process. In the vague limit $\tilde{\beta} = \hat{\beta}$ and the likelihood, up to a constant, is the restricted likelihood above.

Rank-deficient bases make estimated coefficients, and those with a vague prior, unidentifiable, and raise an error of class `gaussianprocesses_rank_deficiency_error`. A proper prior identifies them.

Value

An object of class `gaussianprocesses_coefficients`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. Section 2.7.

See Also

[linear_mean\(\)](#), and the universal-kriging section of vignette("v03-marginal-likelihood-and-prediction").

Examples

```
x <- seq(0, 4, length.out = 25)
y <- 1 + 0.5 * x + sin(2 * x)

estimated <- fit_gp(x, y, rbf_kernel(length_scale = 0.5),
  noise_variance = 0.01, mean = linear_mean())
coef(estimated)

# A vague prior gives the same coefficients and adds their uncertainty to
# predictions away from the data.
marginalized <- fit_gp(x, y, rbf_kernel(length_scale = 0.5),
  noise_variance = 0.01, mean = linear_mean(coefficient_prior()))
predict_gp(estimated, 8)$latent_sd
predict_gp(marginalized, 8)$latent_sd
```

optimize_gp

Optimize Gaussian-process hyperparameters

Description

Estimates kernel parameters and, optionally, observation-noise variance by maximizing the exact log marginal likelihood. Optimization is performed in log-parameter space with deterministic multiple starts.

Usage

```
optimize_gp(
  x,
  y,
  kernel,
  noise_variance = 1e-06,
  mean = zero_mean(),
  optimize_noise = TRUE,
  n_starts = 3L,
  start_spread = 1,
  lower = 1e-08,
  upper = 1e+08,
  control = list(maxit = 500),
  initial_jitter = 0,
```

```

    fallback_jitter = 1e-10,
    jitter_multiplier = 10,
    max_attempts = 8L,
    symmetry_tolerance = sqrt(.Machine$double.eps),
    gradient = c("analytical", "numerical"),
    derivative = NULL,
    noise_groups = NULL
)

```

Arguments

<code>x</code>	Numeric vector or matrix of training inputs.
<code>y</code>	Numeric response vector.
<code>kernel</code>	Initial Gaussian-process kernel specification.
<code>noise_variance</code>	Non-negative observation-noise variance: a scalar, or one value per observation. With <code>optimize_noise = TRUE</code> , a scalar is the starting value of a homoscedastic noise variance, and a vector is a known relative noise pattern Σ_0 whose overall scale s is estimated, $\Sigma = s\Sigma_0$, starting from $s = 1$. With <code>optimize_noise = FALSE</code> the noise is fixed at the given value or values.
<code>mean</code>	Gaussian-process mean specification. Estimated or marginalized coefficients are not optimizer parameters: for each value of the hyperparameters they are computed in closed form, and the objective is the profile, restricted, or marginal likelihood (mean_coefficients).
<code>optimize_noise</code>	If <code>TRUE</code> , optimize the noise variance, or the scale of a noise pattern, jointly with the kernel parameters.
<code>n_starts</code>	Number of deterministic optimization starts.
<code>start_spread</code>	Log-space displacement used to construct additional deterministic starts around the supplied parameters.
<code>lower</code>	Strictly positive lower bound applied to all optimized parameters on the natural scale.
<code>upper</code>	Strictly positive upper bound applied to all optimized parameters on the natural scale.
<code>control</code>	List passed to <code>stats::optim()</code> . The default allows 500 L-BFGS-B iterations per start: simple kernels converge in a few dozen, but composites with periodic components can need a few hundred. Each iteration costs about one GP fit with analytical gradients.
<code>initial_jitter</code>	Non-negative jitter tried first, relative to the covariance scale (the mean of its diagonal).
<code>fallback_jitter</code>	Positive relative jitter tried after the first failed factorization.
<code>jitter_multiplier</code>	Multiplicative jitter escalation factor.
<code>max_attempts</code>	Maximum Cholesky attempts per objective evaluation.
<code>symmetry_tolerance</code>	Relative tolerance for checking that the covariance matrix is symmetric.

gradient	How L-BFGS-B obtains the gradient of the objective. "analytical" (the default) uses <code>log_marginal_likelihood_gradient()</code> , which costs one GP fit per optimizer step. "numerical" lets <code>stats::optim()</code> use finite differences, which costs two fits per parameter per step.
derivative	Optional observation types for derivative observations, as in <code>fit_gp()</code> .
noise_groups	Optional group of each observation, such as the output of a multi-output model (<code>gp_stack_outputs()</code>): with <code>optimize_noise = TRUE</code> , one noise variance is estimated per group, as <code>noise_variance[g]</code> for the <i>g</i> th of the sorted unique groups. <code>noise_variance</code> then gives their starting values: one value for every group, or one per group in that order. Experimental.

Details

Kernel parameters are identified through the stable paths returned by `kernel_parameters(kernel, flatten = TRUE)`. Every parameter is optimized on an unconstrained coordinate determined by its constraint. Positive parameters, including the noise variance, are optimized on the log scale and stay positive. Real-valued parameters, such as the location of a `changepoint_kernel()`, are optimized on the identity scale. `$optimization$coordinates` records the coordinate of each parameter. The deterministic starts do not depend on gradient.

A changepoint location is bounded by the training range of its input column, and the starts after the first place it at evenly spaced quantiles of that column, because the likelihood is often multimodal in the location. A spectral-mixture frequency is bounded by 0 and the Nyquist frequency of its column, and later starts displace it by one frequency-resolution bin (`spectral_mixture_kernel()`).

The objective is the likelihood that `log_marginal_likelihood()` reports for the mean: `$optimization$likelihood` records which, and `$optimization$mean_coefficients` the coefficients at the optimum.

The returned diagnostics retain every start, convergence code, objective value, final parameters, and the gradient method. Evaluation counts are reported per start in `start_results` and summed over all starts:

`function_evaluations` Evaluations of the negative log marginal likelihood, including those `stats::optim()` makes to approximate gradients by finite differences.

`gradient_evaluations` Analytical gradient evaluations, or the number of finite-difference gradient approximations.

`model_fits` GP fits, the dominant cost. With analytical gradients the value and gradient at the same point share one fit.

`counts` is the unmodified `stats::optim()` result for the best start; with numerical gradients it excludes the finite-difference evaluations.

`$optimization$optima` groups the starts by the optimum they reached: starts whose log marginal likelihoods differ by at most 10^{-3} and whose coordinates differ by at most 10^{-2} share one. It reports, for each distinct optimum, its starts, log marginal likelihood, and gap to the best. Several optima with large gaps mean that the starts matter; several with nearly equal likelihoods but different parameters suggest a ridge of poorly identified parameters (`gp_hyperparameter_uncertainty()`). `summary()` reports them.

Value

A fitted `gaussianprocesses_model` with optimized kernel and noise parameters. Optimization diagnostics are stored in `$optimization`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy. The noise_groups argument (one noise variance per group) is experimental.

Examples

```
x <- seq(-2, 2, length.out = 20)
y <- sin(2 * x) + 0.1 * cos(9 * x)

model <- optimize_gp(x, y, kernel = rbf_kernel(), noise_variance = 0.05, n_starts = 1)
model

kernel_parameters(model$kernel, flatten = TRUE)
model$noise_variance
```

optimize_latent_gp	<i>Optimize the hyperparameters of a latent Gaussian-process model</i>
--------------------	--

Description

Estimates kernel parameters and, optionally, likelihood parameters by maximizing the Laplace approximation to the log marginal likelihood, with the optimizer of [optimize_gp\(\)](#): L-BFGS-B on unconstrained coordinates, deterministic multiple starts, and analytical gradients.

Usage

```
optimize_latent_gp(
  x,
  y,
  kernel,
  likelihood,
  mean = zero_mean(),
  exposure = NULL,
  optimize_likelihoood = TRUE,
  n_starts = 3L,
  start_spread = 1,
  lower = 1e-08,
  upper = 1e+08,
  control = list(maxit = 500),
  method = "laplace",
  max_iterations = 100L,
  convergence_tolerance = 1e-14,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
```

```

max_attempts = 8L,
symmetry_tolerance = sqrt(.Machine$double.eps),
gradient = c("analytical", "numerical")
)

```

Arguments

<code>x</code>	Numeric vector or matrix of training inputs. Matrix rows are observations and columns are input dimensions.
<code>y</code>	Responses, one per row of <code>x</code> , as the likelihood accepts them: numbers for Gaussian likelihoods; 0/1, logical values, or a two-level factor whose second level is 1 for Bernoulli likelihoods; and non-negative integer counts for Poisson likelihoods.
<code>kernel</code>	Initial kernel specification.
<code>likelihood</code>	Likelihood specification; its parameters, if any, are the starting values of the likelihood parameters.
<code>mean</code>	Gaussian-process mean specification, with fixed coefficients, coefficients estimated by maximum likelihood, or coefficients with a Gaussian <code>coefficient_prior()</code> .
<code>exposure</code>	Optional positive exposure for Poisson likelihoods: one value or one per observation (see <code>poisson_likelihood()</code>).
<code>optimize_likelihood</code>	If TRUE, optimize the likelihood parameters, such as the variance of <code>gaussian_likelihood()</code> , jointly with the kernel parameters. Likelihoods without parameters ignore it.
<code>n_starts</code> , <code>start_spread</code> , <code>lower</code> , <code>upper</code> , <code>control</code> , <code>gradient</code>	As in <code>optimize_gp()</code> .
<code>method</code>	The approximation: "laplace".
<code>max_iterations</code>	Maximum number of Newton iterations.
<code>convergence_tolerance</code>	The Newton iterations stop when the Newton decrement is at most this value (see Details).
<code>initial_jitter</code> , <code>fallback_jitter</code> , <code>jitter_multiplier</code> , <code>max_attempts</code>	The jitter policy for the factorization of B , relative to its mean diagonal, as in <code>fit_gp()</code> .
<code>symmetry_tolerance</code>	Relative tolerance for checking that B is symmetric.

Details

The objective is $\log q(y \mid X, \theta)$ of `fit_latent_gp()`. Its gradient (Rasmussen and Williams, 2006, Algorithm 5.1) has an explicit part, with the mode $\hat{f}$ held fixed, and an implicit part through the dependence of $\hat{f}$ on θ :

$$\frac{\partial \hat{f}}{\partial \theta_j} = (I + KW)^{-1} \frac{\partial K}{\partial \theta_j} \nabla \log p(y \mid \hat{f}),$$

which needs the third derivatives of the log likelihood (`evaluate_likelihood()`). Likelihood parameters are named `likelihood.<name>`, for example `likelihood.variance`.

Mean coefficients estimated by maximum likelihood (`estimate_coefficients("ml")`) are optimized jointly with the other parameters, on the real line, as `mean.<name>` (for example `mean.intercept`), starting from the generalized linear model fitted without the latent process. Their gradient is $H^\top \nabla \log p(y | \hat{f})$ plus the implicit part through $\partial \hat{f} / \partial \beta = (I + KW)^{-1} H$.

Each evaluation starts the Newton iterations from the previous mode when that is better than the prior mean, which saves most of the iterations once the optimizer takes small steps. The returned model is refitted at the optimum from the prior mean, so it equals `fit_latent_gp()` with the optimized parameters.

`$optimization` has the fields of `optimize_gp()`, with `optimize_likelihood` and `approximation` in place of the noise fields, and `at_bound`, the parameters that ended at lower or upper (on the natural scale) or at the limits that `optimize_gp()` describes for changepoint locations and spectral frequencies.

Value

A fitted `gaussianprocesses_latent_model` with optimized parameters and optimizer diagnostics in `$optimization`.

Bounds and separable data

Every positive parameter is searched within `[lower, upper]`. If any parameter ends at a bound, a warning of class `gaussianprocesses_bound_warning` names it, because the likelihood may still increase beyond the bound and the bound, not the data, then determines the estimate. With binary responses this happens when the classes are separable: larger signal variances let the latent function separate them more sharply, and the approximate marginal likelihood keeps increasing up to very large variances. On twenty separable points on a line it peaks only near a variance of 1800, with a nearly flat maximum, and a smaller upper ends the search at the bound with this warning. Setting upper to the largest plausible variance makes the limit explicit.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[fit_latent_gp\(\)](#).

Examples

```
x <- seq(0, 10, length.out = 40)
counts <- c(1, 0, 2, 1, 3, 2, 4, 5, 3, 6, 7, 5, 8, 6, 9, 7, 6, 8, 5, 6,
  4, 5, 3, 4, 2, 3, 2, 1, 2, 1, 1, 0, 1, 2, 1, 3, 2, 4, 3, 5)

model <- optimize_latent_gp(x, counts, rbf_kernel(), poisson_likelihood(),
  n_starts = 1)
kernel_parameters(model$kernel, flatten = TRUE)
```

optimize_sparse_gp	<i>Optimize the hyperparameters of a sparse Gaussian process</i>
--------------------	--

Description

Estimates kernel parameters, optionally the noise, and optionally the inducing inputs of a sparse model ([fit_sparse_gp\(\)](#)) by maximizing the VFE bound or the FITC approximate marginal likelihood, with the optimizer of [optimize_gp\(\)](#): L-BFGS-B on unconstrained coordinates, deterministic multiple starts, and analytical gradients.

Usage

```
optimize_sparse_gp(
  x,
  y,
  kernel,
  noise_variance = 1e-06,
  mean = zero_mean(),
  n_inducing = NULL,
  inducing_points = NULL,
  selection = c("farthest", "quantile", "variance"),
  method = c("vfe", "fitc"),
  optimize_inducing = FALSE,
  optimize_noise = TRUE,
  n_starts = 3L,
  start_spread = 1,
  lower = 1e-08,
  upper = 1e+08,
  control = list(maxit = 500),
  fitc_diagonal_floor = 1e-10,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps),
  gradient = c("analytical", "numerical")
)
```

Arguments

x	Numeric vector or matrix of training inputs.
y	Numeric response vector.
kernel	Initial kernel specification.
noise_variance	Noise variance: a scalar starting value, or one value per observation forming a known pattern whose scale is estimated, as in optimize_gp() . With <code>optimize_noise = FALSE</code> it is fixed.

mean	Gaussian-process mean specification with fixed coefficients.
n_inducing	Optional number of inducing points selected from x .
inducing_points	Optional explicit inducing locations. If supplied, <code>n_inducing</code> is ignored.
selection	Inducing-point selection method used when explicit points are not supplied: "farthest", "quantile", or "variance" (greedy conditional-variance selection under kernel; see select_inducing_points()).
method	The approximation: "vfe" (the default here) or "fitc".
optimize_inducing	If TRUE, optimize the inducing inputs jointly with the hyperparameters, starting from the selected or supplied points. The kernel must be differentiable in its inputs (kernel_input_gradient()).
optimize_noise, n_starts, start_spread, lower, upper, control, gradient	As in optimize_gp() .
fitc_diagonal_floor	Strictly positive numerical floor applied to the FITC diagonal term before inversion.
initial_jitter	Non-negative jitter tried first in the inducing-system Cholesky factorizations, relative to the scale of each factorized matrix (the mean of its diagonal).
fallback_jitter	Positive relative jitter tried after the first failed factorization.
jitter_multiplier	Multiplicative jitter escalation factor.
max_attempts	Maximum Cholesky attempts.
symmetry_tolerance	Relative tolerance for checking that covariance matrices are symmetric.

Details

VFE is the default because its objective is a lower bound on the exact log marginal likelihood: maximizing it fits the hyperparameters of the exact model as well as the inducing points allow, and pays for poor coverage through the trace term. FITC's objective is not a bound; its diagonal correction can absorb noise, and maximizing it tends to underestimate the noise variance (Bauer, van der Wilk, and Rasmussen, 2016). The research note on sparse approximations compares the two.

The gradient is analytical for both approximations, with the inducing points fixed: with $C = Q_{ff} + \Lambda$ and $M = C^{-1}rr^\top C^{-1} - C^{-1}$,

$$\frac{\partial}{\partial \theta_j} \log N(r \mid 0, C) = \frac{1}{2} \text{tr} \left(M \left(\frac{\partial Q_{ff}}{\partial \theta_j} + \frac{\partial \Lambda}{\partial \theta_j} \right) \right),$$

where ∂Q_{ff} follows from the derivatives of K_{uf} and K_{uu} , plus the derivative of the VFE trace term. It is computed in $O(nm^2)$ time without forming an $n \times n$ matrix, processing the observations in blocks.

Value

A fitted `gaussianprocesses_sparse_model` with optimizer diagnostics in `$optimization`, which has the fields of `optimize_gp()`.

Optimizing the inducing inputs

With `optimize_inducing = TRUE` the $m \times D$ inducing inputs Z are optimized as well, on the real line, as parameters named `inducing.<j>.<d>`. Their gradient follows from the same terms: Z_{jd} changes only row j of K_{uf} and row and column j of K_{uu} ,

$$\frac{\partial \mathcal{F}}{\partial Z_{jd}} = \sum_i G_{ji}^{uf} \frac{\partial k(z_j, x_i)}{\partial z_{jd}} + 2 \sum_l G_{jl}^{uu} \frac{\partial k(z_j, z_l)}{\partial z_{jd}},$$

with the kernel input derivatives of `kernel_input_gradient()` and the sensitivities G^{uf} and G^{uu} of the objective to K_{uf} and K_{uu} , at the same $O(nm^2)$ cost.

Optimizing Z is recommended with VFE: the bound can only move towards the exact marginal likelihood, so better inducing inputs make the approximation better. With FITC it is prone to over-fitting: the inducing inputs can move or collapse so that FITC's diagonal correction absorbs the noise, and the noise variance is then underestimated (Bauer, van der Wilk, and Rasmussen, 2016). Inducing points that become nearly identical under the kernel are reported in `$inducing_collapse` with a warning of class `gaussianprocesses_inducing_collapse_warning`; K_{uu} is factorized with the package's jitter policy, so the fit stays defined. Greedy variance selection (`selection = "variance"`) gives a good starting point.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

`fit_sparse_gp()`, `optimize_gp()`.

Examples

```
set.seed(1)
x <- seq(-3, 3, length.out = 300)
y <- sin(2 * x) + rnorm(300, sd = 0.2)

model <- optimize_sparse_gp(x, y, rbf_kernel(), noise_variance = 0.1,
  n_inducing = 20, n_starts = 1)
model
model$noise_variance
```

```
optimize_time_series_gp
```

Optimize the hyperparameters of a time-series GP

Description

Estimates the kernel parameters and the noise of a GP on ordered time-series observations by maximizing the log marginal likelihood, with the optimizer of [optimize_gp\(\)](#).

Usage

```
optimize_time_series_gp(
  time,
  y,
  kernel,
  unit = "auto",
  method = c("exact", "state_space"),
  ...
)
```

Arguments

time	Numeric, Date, or POSIXt vector in strictly increasing order.
y	Numeric response vector.
kernel	Initial kernel specification.
unit	Time unit passed to gp_time_index() .
method	"exact" conditions a Gaussian prior on the observations with fit_gp() . "state_space" computes the same posterior and log marginal likelihood in $O(n)$ time by Kalman filtering and smoothing; it needs a Matérn-1/2, 3/2, or 5/2 kernel, a sum of such kernels, or scaled versions of them (see State-space inference).
...	Additional arguments passed to optimize_gp() . With method = "state_space", the arguments noise_variance, mean, optimize_noise, n_starts, start_spread, lower, upper, and control apply, with the meanings and defaults of optimize_gp() .

Details

With method = "state_space" each objective value is one Kalman filter pass, in $O(n)$ time ([fit_time_series_gp\(\)](#) describes the method). Gradients are finite-difference approximations by L-BFGS-B (gradient = "numerical" in [optimize_gp\(\)](#)); analytical sensitivity recursions are not implemented yet.

Value

A fitted model as returned by [fit_time_series_gp\(\)](#), with optimizer diagnostics in \$optimization.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
set.seed(1)
time <- sort(runif(200, 0, 50))
y <- sin(time / 3) + rnorm(200, sd = 0.2)

model <- optimize_time_series_gp(time, y, matern52_kernel(),
  noise_variance = 0.1, n_starts = 1, method = "state_space")
model$optimization$optimized_parameters
```

parametric_means	<i>Parametric mean functions</i>
------------------	----------------------------------

Description

Regression trends $m(x) = h(x)^\top \beta$, the mean of universal kriging, with fixed, estimated, or marginalized coefficients.

Usage

```
linear_mean(coefficients = estimate_coefficients())

polynomial_mean(
  degree = 2L,
  coefficients = estimate_coefficients(),
  center = NULL,
  scale = NULL
)

basis_mean(basis, coefficients = estimate_coefficients())
```

Arguments

coefficients	estimate_coefficients() (the default), coefficient_prior() , or a numeric vector that fixes the coefficients, in the order of the basis columns.
degree	Non-negative integer polynomial degree.
center, scale	Optional centring and scaling of the inputs for <code>polynomial_mean()</code> : one value, recycled, or one per input column. If NULL, they are set from the training inputs when a model is fitted.
basis	A function of the input matrix that returns a numeric matrix with one row per input and one column per coefficient.

Details

The bases are:

`linear_mean()` $h(x) = (1, x_1, \dots, x_D)$, on the raw inputs, so the coefficients are an intercept and slopes. Coefficients are named `intercept`, `x1`, ..., `xD`.

`polynomial_mean()` An intercept and the powers $z_d, z_d^2, \dots, z_d^k$ of each standardized input $z_d = (x_d - c_d)/s_d$, without interactions. Powers of raw inputs make badly conditioned bases, so by default c_d is the midpoint and s_d half the range of the training inputs, which maps them to $[-1, 1]$. A fitted model stores these values in its mean specification and uses them for prediction, and the coefficients, named `intercept`, `x1`, `x1^2`, ..., refer to the standardized inputs. `polynomial_mean(0)` is an unknown constant.

`basis_mean()` The columns returned by `basis`, named by its column names or `h1`, `h2`, ...

`mean_coefficients` describes the three treatments of β . `coef()` and `vcov()` return the coefficients of a fitted model and their covariance.

Parametric means are supported by exact models: `fit_gp()`, `optimize_gp()`, and the functions that use them. Sparse and heteroscedastic models accept them only with fixed coefficients.

Value

An object of class `gaussianprocesses_mean`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- cbind(seq(0, 4, length.out = 30), rep(c(-1, 1), 15))
y <- 2 + 0.5 * x[, 1] - x[, 2] + sin(2 * x[, 1])

model <- fit_gp(x, y, rbf_kernel(length_scale = c(0.5, 5)),
  noise_variance = 0.01, mean = linear_mean())
coef(model)
sqrt(diag(vcov(model)))

# A quadratic trend in one input; coefficients refer to inputs scaled to
# [-1, 1].
x1 <- seq(-3, 3, length.out = 30)
quadratic <- fit_gp(x1, x1^2 + sin(3 * x1), rbf_kernel(length_scale = 0.4),
  noise_variance = 0.01, mean = polynomial_mean(2))
coef(quadratic)
quadratic$mean$scale

# Any basis, here a periodic one with fixed coefficients.
seasonal <- basis_mean(
  function(x) cbind(sin = sin(x[, 1]), cos = cos(x[, 1])),
  coefficients = c(1, 0.5)
```

```
)
evaluate_mean(seasonal, c(0, pi / 2))
```

periodic_kernel	Create a periodic kernel specification
-----------------	--

Description

Create a periodic kernel specification

Usage

```
periodic_kernel(variance = 1, length_scale = 1, period = 1)
```

Arguments

variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector. A vector supplies one ARD length scale per input dimension and is validated against the data when the kernel is evaluated.
period	Positive period.

Value

An object of class `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_periodic\(\)](#) computes the same covariance directly from parameter values, without a kernel specification.

Examples

```
# Monthly data with a yearly cycle.
kernel <- periodic_kernel(length_scale = 0.8, period = 12)
evaluate_kernel(kernel, c(0, 6, 12))
```

predict_gp

*Predict from an exact Gaussian-process regression model***Description**

Computes the posterior mean and uncertainty at new input locations. Latent function uncertainty and future-observation uncertainty are reported separately. The stored Cholesky factor from model fitting is reused; no covariance-matrix inverse is formed.

Usage

```
predict_gp(
  model,
  x,
  interval_level = 0.95,
  include_covariance = FALSE,
  variance_tolerance = sqrt(.Machine$double.eps),
  observation_noise_variance = NULL,
  block_size = NULL
)
```

Arguments

model	A fitted gaussianprocesses_model, or a state-space model from fit_time_series_gp() , which is predicted by Kalman smoothing at inputs on its time-index scale (block_size does not apply; memory is linear in the number of inputs).
x	Numeric vector or matrix of prediction inputs. Matrix rows are observations and columns must match the fitted input dimension.
interval_level	Probability level for latent credible intervals and noisy-observation prediction intervals.
include_covariance	If TRUE, return the full latent and observation posterior covariance matrices. Otherwise only marginal variances are returned.
variance_tolerance	Relative tolerance used when deciding whether a small negative posterior variance is floating-point error or a material numerical failure.
observation_noise_variance	Optional non-negative scalar or vector giving observation-noise variance at the prediction inputs. It defaults to the fitted scalar noise variance for homoscedastic models. Models fitted with observation-specific training noise require this argument.
block_size	Number of prediction inputs processed together when include_covariance = FALSE. NULL (the default) chooses the largest block whose training-by-block cross-covariance holds at most 2^{20} values (8 MiB). Results do not depend on the block size. It is ignored when include_covariance = TRUE.

Details

With n training and m prediction inputs, $C = R^\top R$ the factorized training covariance, and $V = R^{-\top} K(X, X_*)$, the latent posterior variance at prediction input j is

$$\text{Var}[f(x_{*j}) \mid y] = k(x_{*j}, x_{*j}) - \sum_i V_{ij}^2.$$

This is exact. The default marginal prediction evaluates only the cross-covariance $K(X, X_*)$, in blocks of `block_size` prediction inputs, and the prior variances $k(x_{*j}, x_{*j})$ with `kernel_diagonal()`. It takes $O(n^2m)$ time and $O(nb)$ memory for blocks of b inputs, beyond the $O(n^2)$ fitted factor.

`include_covariance = TRUE` also forms the $m \times m$ prior covariance and the latent and observation posterior covariances, which takes $O(n^2m + nm^2)$ time and $O(m^2)$ memory. Each $m \times m$ matrix takes $8m^2$ bytes, 512 MB for 8000 inputs, and several are held at once. It is opt-in because pointwise means, variances, and intervals do not need it; joint quantities such as function draws (`sample_gp_posterior()`) do.

A parametric mean contributes $h(x_*)^\top \beta$ with its fixed, estimated, or posterior-mean coefficients. Estimated coefficients are plugged in. Marginalized coefficients add their uncertainty, $R_*(B^{-1} + H^\top C^{-1} H)^{-1} R_*^\top$ with $R_* = H_* - V^\top R^{-\top} H$, to the latent variances and covariances (`mean_coefficients`).

Value

An object of class `gaussianprocesses_prediction`. Its fields are shared by every prediction in the package: `predict_sparse_gp()`, `predict_heteroscedastic_gp()`, and `forecast_gp()` return the same fields with the same meaning, and may add their own.

`x` The prediction inputs as a matrix.

`mean` The posterior mean of the latent function.

`latent_variance`, `latent_sd` The posterior variance and standard deviation of the latent function.

`observation_noise_variance` The noise variance of a new observation.

`observation_variance`, `observation_sd` The variance and standard deviation of a new noisy observation: `latent_variance + observation_noise_variance`.

`interval_level`, `latent_interval`, `prediction_interval` The interval level and the intervals for the latent function and for a new observation.

`latent_covariance`, `observation_covariance` The full covariance matrices, only with `include_covariance = TRUE`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```

x <- seq(-2, 2, length.out = 15)
model <- fit_gp(
  x,
  sin(2 * x),
  kernel = rbf_kernel(length_scale = 0.6),
  noise_variance = 0.01
)

prediction <- predict_gp(model, c(0, 3))

# Latent-function and noisy-observation uncertainty are reported separately;
# both grow away from the data.
prediction$latent_sd
prediction$observation_sd
prediction$prediction_interval

```

predict_gradient_gp	<i>Predict the gradient of the latent function</i>
---------------------	--

Description

Computes the posterior mean and variance of each partial derivative $\partial f(x_*)/\partial x_{*,d}$ of the latent function of an exact Gaussian-process model. Differentiation is linear, so these derivatives are jointly Gaussian with the observations and the posterior is exact.

Usage

```

predict_gradient_gp(
  model,
  x,
  interval_level = 0.95,
  variance_tolerance = sqrt(.Machine$double.eps),
  block_size = NULL
)

```

Arguments

model	A fitted gaussianprocesses_model.
x	Numeric vector or matrix of prediction inputs.
interval_level	Probability level for the credible intervals.
variance_tolerance	Relative tolerance used when deciding whether a small negative posterior variance is floating-point error.
block_size	Number of prediction inputs processed together; NULL chooses it from a memory budget, as in predict_gp() .

Details

With $C = R^\top R$ the factorized training covariance and $\alpha = C^{-1}(y - m(X))$,

$$\mathbb{E}[\partial_d f(x_*) \mid y] = \partial_d m(x_*) + \partial_{x_*,d} k(x_*, X) \alpha,$$

$$\text{Var}[\partial_d f(x_*) \mid y] = \partial_d \partial_d' k(x_*, x_*) - v_d^\top v_d, \quad v_d = R^{-\top} \partial_{x_*,d} k(X, x_*).$$

The kernel derivatives come from [kernel_input_gradient\(\)](#) and its second-order counterpart. Only mean-square differentiable kernels have a gradient process; others raise an error of class `gaussianprocesses_smoothness_error`.

The variances are marginal: covariances between partial derivatives, or between points, are not returned.

Parametric means add $\partial h(x_*)^\top \beta / \partial x_{*,d}$ to the mean. Marginalized coefficients add their uncertainty to the variance, as in [predict_gp\(\)](#), with H_* and $K(x_*, X)$ replaced by their derivatives. [basis_mean\(\)](#) has no input derivatives and raises an error.

Value

An object of class `gaussianprocesses_gradient_prediction` with the prediction inputs `x` and, as matrices with one row per input and one column per input dimension, the posterior mean, variance, and sd of the partial derivatives and the credible-interval bounds lower and upper at `interval_level`.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(0, 2 * pi, length.out = 25)
model <- fit_gp(
  x,
  sin(x),
  rbf_kernel(length_scale = 1),
  noise_variance = 1e-4
)

gradient <- predict_gradient_gp(model, c(0, pi / 2, pi))

# The derivative of sin is cos.
cbind(estimate = gradient$mean[, 1], truth = cos(c(0, pi / 2, pi)))
gradient$sd
```

predict_heteroscedastic_gp

Predict from an approximate heteroscedastic GP

Description

The latent-function posterior and the input-dependent noise process are predicted separately. Observation variance is the sum of latent posterior variance and the plug-in estimated noise variance. Uncertainty about the log-noise process is returned separately and is not folded into the observation variance.

Usage

```
predict_heteroscedastic_gp(
  model,
  x,
  interval_level = 0.95,
  include_covariance = FALSE,
  variance_tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

model	A fitted heteroscedastic GP model.
x	Numeric vector or matrix of prediction inputs.
interval_level	Probability level for returned intervals.
include_covariance	If TRUE, include the full latent and observation covariance matrices.
variance_tolerance	Relative tolerance for latent posterior variances.

Value

An object of class `gaussianprocesses_heteroscedastic_prediction`. It has the fields that every prediction in the package shares (see [predict_gp\(\)](#)), with the plug-in noise estimate as `observation_noise_variance`, and in addition `noise_log_variance`, its posterior variance `noise_log_variance_uncertain` and `noise_variance_interval` from the log-noise GP.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```

simulation <- gp_simulation_scenario("heteroscedastic_1d", n = 60)
model <- fit_heteroscedastic_gp(
  simulation$x,
  simulation$observed,
  kernel = matern52_kernel(length_scale = 0.2),
  noise_kernel = rbf_kernel(length_scale = 0.4),
  max_iterations = 30
)

prediction <- predict_heteroscedastic_gp(model, c(0.1, 0.9))

# Observation variance = latent variance + estimated noise variance.
cbind(
  latent = prediction$latent_variance,
  noise = prediction$observation_noise_variance,
  observation = prediction$observation_variance
)

```

predict_latent_gp	<i>Predict from a latent Gaussian-process model</i>
-------------------	---

Description

Computes the approximate posterior of the latent function at new inputs and the predictive distribution of new responses through the likelihood: class probabilities for Bernoulli likelihoods and predictive means and variances for every likelihood.

Usage

```

predict_latent_gp(
  model,
  x,
  interval_level = 0.95,
  include_covariance = FALSE,
  variance_tolerance = sqrt(.Machine$double.eps),
  exposure = NULL,
  quadrature_order = 40L,
  block_size = NULL
)

```

Arguments

model	A fitted gaussianprocesses_latent_model from fit_latent_gp() or optimize_latent_gp() .
x	Numeric vector or matrix of prediction inputs.
interval_level	Probability level of the latent credible intervals.
include_covariance	If TRUE, also return the full latent posterior covariance matrix.

variance_tolerance	Relative tolerance used when deciding whether a small negative posterior variance is floating-point error or a material numerical failure.
exposure	Optional positive exposure of the new responses for Poisson likelihoods: one value or one per prediction input. It defaults to 1.
quadrature_order	Number of Gauss-Hermite nodes for predictive integrals without a closed form (<code>likelihood_predictive()</code>).
block_size	Number of prediction inputs processed together when <code>include_covariance = FALSE</code> , as in <code>predict_gp()</code> .

Details

Under the Laplace approximation the latent values at new inputs are Gaussian (Rasmussen and Williams, 2006, Algorithm 3.2), with

$$\mathbb{E}[f_*] = m(x_*) + k_*^\top \nabla \log p(y \mid \hat{f}), \quad \text{Var}[f_*] = k(x_*, x_*) - v^\top v,$$

where $k_* = k(X, x_*)$, $v = R^{-\top} W^{1/2} k_*$, and $R^\top R = B$. The predictive distribution of a new response integrates the likelihood against this Gaussian with `likelihood_predictive()`.

Value

An object of class `gaussianprocesses_latent_prediction` with the shared latent fields of `predict_gp()`, `x`, `mean`, `latent_variance`, `latent_sd`, `interval_level`, and `latent_interval` (and `latent_covariance` with `include_covariance = TRUE`), and the predictive distribution of a new response: `response_mean`, `response_variance`, probability $\Pr(y_* = 1)$ and `probability_interval` for Bernoulli likelihoods, `rate_interval`, `prediction_interval`, and `exposure` for Poisson likelihoods, and `prediction_interval` for Gaussian likelihoods. A non-Gaussian response has no observation-noise variance, so the observation fields of `predict_gp()` are replaced by these; `prediction_interval` has its meaning there, an interval for a new response.

Class probabilities

For a Bernoulli likelihood, probability is $\Pr(y_* = 1 \mid y)$ for the response coded 1: the second level of a factor response (`model$response_levels`), or `TRUE`. It averages the link over the latent posterior,

$$\Pr(y_* = 1 \mid y) = \int \pi(f_*) N(f_* \mid \mu_*, \sigma_*^2) df_*,$$

which for the probit link is $\Phi(\mu_*/\sqrt{1 + \sigma_*^2})$ exactly and for the logistic link is computed by Gauss-Hermite quadrature. It is not $\pi(\mu_*)$: the plug-in probability ignores the latent uncertainty and is too confident, by up to 0.04 at $\sigma_*^2 = 1$ and 0.11 at $\sigma_*^2 = 4$. MacKay's approximation $\sigma(\kappa\mu_*)$ with $\kappa = (1 + \pi\sigma_*^2/8)^{-1/2}$ is closer but still off by up to 0.005 at $\sigma_*^2 = 1$ and 0.01 at $\sigma_*^2 = 4$, so it is not used.

`probability_interval` describes the uncertainty of the class probability $\pi(f_*)$ itself: because the link is increasing, the `interval_level` quantiles of $\pi(f_*)$ are the link applied to the latent interval, exactly under the Gaussian approximation. Joint uncertainty, for example of a difference of probabilities, comes from the probability draws of `sample_gp_posterior()`.

Multi-class (softmax) classification is not supported.

Counts

For a Poisson likelihood the rate $\lambda_* = E_* e^{f_*}$ is lognormal: its mean is $E_* e^{\mu_* + \sigma_*^2/2}$ (response_mean, also the mean of the count) and its quantiles are $E_* e^{\mu_* \pm z \sigma_*}$ exactly, because the exponential is increasing (rate_interval). A new count has the Poisson-lognormal predictive distribution; prediction_interval gives its equal-tailed quantiles at interval_level, computed by quadrature (gp_count_scores()). Counts are discrete, so these intervals cover at least interval_level: their coverage is conservative.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

See Also

[fit_latent_gp\(\)](#), [sample_gp_posterior\(\)](#).

Examples

```
x <- seq(-3, 3, length.out = 30)
y <- as.numeric(sin(x) + 0.3 * cos(5 * x) > 0)
model <- fit_latent_gp(x, y, rbf_kernel(variance = 4), bernoulli_likelihood())

prediction <- predict_latent_gp(model, c(-2, 0, 2, 6))

# Far from the data the latent variance grows and the probability moves
# towards 1/2, unlike the logistic function of the latent mean.
cbind(prediction$probability, plogis(prediction$mean))
```

predict_sparse_gp	<i>Predict from a sparse Gaussian-process model</i>
-------------------	---

Description

Computes the posterior of the latent function and of new observations under a FITC or VFE model ([fit_sparse_gp\(\)](#)).

Usage

```
predict_sparse_gp(
  model,
  x,
  interval_level = 0.95,
  include_covariance = FALSE,
  variance_tolerance = sqrt(.Machine$double.eps),
  observation_noise_variance = NULL,
  block_size = NULL
)
```

Arguments

model	A fitted sparse GP model.
x	Numeric vector or matrix of prediction inputs.
interval_level	Probability level for intervals.
include_covariance	If TRUE, construct the full latent covariance matrix. Otherwise only marginal variances are computed.
variance_tolerance	Relative tolerance for posterior variances.
observation_noise_variance	Optional prediction-time observation-noise variance. Required when the sparse model was fitted with observation- specific training noise.
block_size	Number of prediction inputs processed together when include_covariance = FALSE. NULL chooses the largest block whose inducing-by-block cross-covariance holds at most 2^{20} values, as in predict_gp() ; memory then grows linearly with the number of prediction inputs. Results do not depend on it.

Details

With $A = K_{uu} + K_{uf}\Lambda^{-1}K_{fu}$, where Λ is the FITC diagonal or, for VFE, the noise, the latent posterior at new inputs is

$$\mu_* = m(x_*) + K_{*u}A^{-1}K_{uf}\Lambda^{-1}(y - m(X)), \quad \Sigma_{**} = K_{**} - Q_{**} + K_{*u}A^{-1}K_{u*},$$

computed through the Cholesky factors of K_{uu} and $B = I + V\Lambda^{-1}V^\top$. For VFE these are the predictive equations of the deterministic training conditional.

Value

An object of class `gaussianprocesses_sparse_prediction`, with the shared prediction fields of [predict_gp\(\)](#) and approximation.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(-3, 3, length.out = 200)
y <- sin(x) + 0.1 * cos(7 * x)
kernel <- rbf_kernel(length_scale = 0.8)

sparse <- fit_sparse_gp(x, y, kernel, noise_variance = 0.01, n_inducing = 15)
exact <- fit_gp(x, y, kernel, noise_variance = 0.01)

x_new <- c(-1, 0, 1)
cbind(
  exact = predict_gp(exact, x_new)$mean,
```

```
sparse = predict_sparse_gp(sparse, x_new)$mean
)
```

product_kernel	<i>Multiply covariance kernels</i>
----------------	------------------------------------

Description

The covariance matrices generated by all supplied kernels are multiplied element-wise.

Usage

```
product_kernel(...)
```

Arguments

... Two or more Gaussian-process kernel specifications.

Value

A composite kernel specification.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
# A locally periodic kernel: a periodic pattern whose shape drifts slowly.
kernel <- product_kernel(
  periodic_kernel(period = 12),
  rbf_kernel(length_scale = 24)
)
evaluate_kernel(kernel, c(0, 12, 48))[1, ]
```

`rational_quadratic_kernel`*Create a rational-quadratic kernel specification*

Description

Create a rational-quadratic kernel specification

Usage

```
rational_quadratic_kernel(variance = 1, length_scale = 1, alpha = 1)
```

Arguments

<code>variance</code>	Positive marginal variance.
<code>length_scale</code>	Positive scalar or numeric vector. A vector supplies one ARD length scale per input dimension and is validated against the data when the kernel is evaluated.
<code>alpha</code>	Positive scale-mixture parameter.

Value

An object of class `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_rational_quadratic\(\)](#) computes the same covariance directly from parameter values, without a kernel specification.

Examples

```
kernel <- rational_quadratic_kernel(length_scale = 0.8, alpha = 2)
evaluate_kernel(kernel, c(0, 0.5, 1))
```

rbf_kernel	Create an RBF kernel specification
------------	------------------------------------

Description

Kernel specifications are reusable objects that store covariance parameters separately from the observations on which they are evaluated. They are the building blocks for composite kernels and later Gaussian-process fitting.

Usage

```
rbf_kernel(variance = 1, length_scale = 1)
```

Arguments

variance	Positive marginal variance.
length_scale	Positive scalar or numeric vector. A vector supplies one ARD length scale per input dimension and is validated against the data when the kernel is evaluated.

Value

An object of class `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_rbf\(\)](#) computes the same covariance directly from parameter values, without a kernel specification.

Examples

```
kernel <- rbf_kernel(variance = 1.5, length_scale = c(0.5, 2))
kernel

evaluate_kernel(kernel, cbind(c(0, 1), c(0, 1)))
```

rolling_origin_gp	<i>Rolling-origin evaluation for time-aware Gaussian processes</i>
-------------------	--

Description

Performs expanding-window rolling-origin evaluation with fixed kernel and noise hyperparameters. Each split is fitted using only observations available at that origin, preventing look-ahead leakage.

Usage

```
rolling_origin_gp(
  time,
  y,
  kernel,
  initial_window,
  horizon = 1L,
  step = 1L,
  noise_variance = 1e-06,
  mean = zero_mean(),
  unit = "auto",
  interval_level = 0.95,
  ...
)
```

Arguments

time	Numeric, Date, or POSIXt vector in strictly increasing order.
y	Numeric response vector.
kernel	Fixed Gaussian-process kernel specification.
initial_window	Number of observations in the first training window.
horizon	Number of future observations predicted at each origin.
step	Number of observations between consecutive origins.
noise_variance	Non-negative observation-noise variance: a scalar, or one value per observation. With one value per observation, each training window uses the values of its rows and each forecast uses the values of its target rows.
mean	Gaussian-process mean specification.
unit	Time unit passed to <code>gp_time_index()</code> .
interval_level	Probability level for prediction intervals.
...	Additional arguments passed to <code>fit_time_series_gp()</code> , such as <code>method = "state_space"</code> .

Details

Fixed hyperparameters prevent future observations from influencing earlier origins through repeated whole-series hyperparameter tuning. Hyperparameters can be selected separately using an appropriate training period.

Value

A data frame with class `gaussianprocesses_backtest`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
time <- 0:23
y <- sin(2 * pi * time / 12)

backtest <- rolling_origin_gp(
  time,
  y,
  kernel = time_series_kernel(period = 12),
  initial_window = 12,
  horizon = 3,
  step = 3,
  noise_variance = 0.05
)
head(backtest[, c("origin_index", "horizon", "observed", "mean", "covered")])
```

```
run_gp_benchmark_suite
```

Run the standard GP benchmark suite

Description

Runs seeded synthetic scenarios through exact and FITC regression. Accuracy, performance, and numerical stability are returned as separate tables so faster execution is never conflated with better numerical agreement.

Usage

```
run_gp_benchmark_suite(
  scenarios = c("smooth_1d", "near_singular", "ard_2d", "heteroscedastic_1d"),
  n = 60L,
  n_inducing = 15L,
  repeats = 3L,
  seed = 1729L
)
```

Arguments

scenarios	Character vector of standard scenario names.
n	Number of observations per scenario.
n_inducing	Number of FITC inducing points.
repeats	Number of timing repetitions. Medians are reported.
seed	Base non-negative integer seed. Scenario-specific offsets are applied deterministically.

Details

Timing results use median elapsed wall-clock time and are intended for local comparison, not CI gating. Retained memory is measured with `utils::object.size()` and is not a peak-allocation measurement.

The exact Gaussian process is used as the numerical reference when assessing FITC approximation error. Synthetic scenarios also retain their latent draw so both exact and sparse posterior means can be compared against the same known simulation truth.

Value

An object of class `gaussianprocesses_benchmark_suite` containing separate accuracy, performance, and stability data frames.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

Examples

```
# A deliberately small run; see inst/benchmarks/run-suite.R for a full one.
suite <- run_gp_benchmark_suite(
  scenarios = "smooth_1d",
  n = 30,
  n_inducing = 8,
  repeats = 1
)
suite$accuracy
```

```
sample_gp_posterior.gaussianprocesses_latent_model
```

Draw functions from a Gaussian-process posterior

Description

Draws latent function values from the posterior of a fitted model and, optionally, noisy observations from the posterior predictive distribution.

Usage

```
## S3 method for class 'gaussianprocesses_latent_model'
sample_gp_posterior(
  model,
  x,
  n_draws = 1L,
  type = c("latent", "observation"),
  seed = NULL,
  exposure = NULL,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps),
  ...
)

sample_gp_posterior(model, x, n_draws = 1L, ...)

## Default S3 method:
sample_gp_posterior(model, x, n_draws = 1L, ...)

## S3 method for class 'gaussianprocesses_model'
sample_gp_posterior(
  model,
  x,
  n_draws = 1L,
  type = c("latent", "observation"),
  observation_noise_variance = NULL,
  seed = NULL,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps),
  ...
)

## S3 method for class 'gaussianprocesses_heteroscedastic_model'
sample_gp_posterior(
  model,
  x,
  n_draws = 1L,
  type = c("latent", "observation"),
  seed = NULL,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
```

```

    max_attempts = 8L,
    symmetry_tolerance = sqrt(.Machine$double.eps),
    ...
)

## S3 method for class 'gaussianprocesses_sparse_model'
sample_gp_posterior(
  model,
  x,
  n_draws = 1L,
  type = c("latent", "observation"),
  observation_noise_variance = NULL,
  seed = NULL,
  initial_jitter = 0,
  fallback_jitter = 1e-10,
  jitter_multiplier = 10,
  max_attempts = 8L,
  symmetry_tolerance = sqrt(.Machine$double.eps),
  ...
)

## S3 method for class 'gaussianprocesses_state_space_model'
sample_gp_posterior(
  model,
  x,
  n_draws = 1L,
  type = c("latent", "observation"),
  observation_noise_variance = NULL,
  seed = NULL,
  ...
)

```

Arguments

model	A fitted gaussianprocesses_model (from fit_gp() , optimize_gp() , or fit_time_series_gp()), a gaussianprocesses_heteroscedastic_model (from fit_heteroscedastic_gp()), a gaussianprocesses_latent_model (from fit_latent_gp() or optimize_latent_gp()), or a gaussianprocesses_sparse_model (from fit_sparse_gp() or optimize_sparse_gp()).
x	Numeric vector or matrix of prediction inputs. For models fitted with fit_time_series_gp() , use the numeric time index.
n_draws	Number of independent draws.
type	"latent" draws the latent function $f(x)$. "observation" also draws noisy observations $y = f(x) + \varepsilon$; both are returned, with the observations generated from the same latent draws.
seed	Optional non-negative integer seed. The caller's random state is restored afterwards.
exposure	For latent Poisson models, optional exposure of the observation draws, as in predict_latent_gp() .

<code>initial_jitter</code>	Non-negative jitter tried first, relative to the covariance scale (the mean of its diagonal).
<code>fallback_jitter</code>	Positive relative jitter tried after the first failed factorization.
<code>jitter_multiplier</code>	Multiplicative jitter escalation factor.
<code>max_attempts</code>	Maximum Cholesky attempts.
<code>symmetry_tolerance</code>	Relative tolerance for checking that the covariance matrix is symmetric.
<code>...</code>	Further arguments passed to methods.
<code>observation_noise_variance</code>	For exact and sparse models, optional non-negative scalar or vector of noise variances at x , used only when <code>type = "observation"</code> . It defaults to the fitted noise variance of homoscedastic models and is required for models fitted with observation-specific noise.

Details

The posterior mean μ and latent covariance Σ come from `predict_gp()` (or `predict_heteroscedastic_gp()`), which uses triangular solves with the fitted Cholesky factor; no covariance matrix is inverted. Latent draws are $\mu + R^\top z$ with $\Sigma = R^\top R$ and $z \sim \mathcal{N}(0, I)$. Posterior covariances are often nearly singular, for example at closely spaced or repeated inputs; the factor then uses the package's jitter policy, and the jitter is reported.

Observation draws add independent noise, $y = f + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \text{diag}(\sigma_*^2))$, so their covariance is $\Sigma + \text{diag}(\sigma_*^2)$.

For heteroscedastic models, the latent draws come from the mean GP and the noise variances σ_*^2 are the plug-in estimates of `predict_heteroscedastic_gp()`. The draws are therefore conditional on the estimated noise function: uncertainty in the noise process is not propagated, consistent with the approximation being a plug-in method rather than full Bayesian inference.

For latent models, the latent draws come from the Gaussian (Laplace) approximation of `predict_latent_gp()`, and observation draws are drawn from the likelihood given each latent draw: Bernoulli, Poisson (with exposure), or Gaussian responses. Their `observation_noise_variance` is `NULL`. For Bernoulli likelihoods the samples also contain probability, the class probability $\pi(f)$ of each latent draw, whose mean over draws is the predictive probability of `predict_latent_gp()`; for Poisson likelihoods they contain rate, the rate Ee^f of each latent draw.

For sparse models (FITC or VFE, `fit_sparse_gp()`), the draws use the sparse posterior covariance $K_{**} - Q_{**} + K_{*u}A^{-1}K_{u*}$ of `predict_sparse_gp()` at the k sampling inputs. It costs $O(km^2 + k^2m)$ time for m inducing points, and its factorization $O(k^3)$ time and $O(k^2)$ memory, as for exact models. The number of inducing points enters only through $m \times k$ cross-covariances, so the limit is the number of sampling inputs, a few thousand, rather than m . Observation draws need `observation_noise_variance` for models fitted with observation-specific noise.

Value

An object of class `gaussianprocesses_samples` with one row per input point and one column per draw in latent and, for `type = "observation"`, in observation. It also contains the posterior

mean, the latent_covariance that was sampled, the observation_noise_variance used for observation draws, the numerical jitter with its relative_jitter and covariance_scale, and the cholesky_attempts.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- c(-2, -1, 0, 1.5)
y <- sin(2 * x)
model <- fit_gp(x, y, rbf_kernel(length_scale = 0.7), noise_variance = 0.01)

x_new <- seq(-3, 3, length.out = 100)
draws <- sample_gp_posterior(model, x_new, n_draws = 5, seed = 1)
draws

matplot(x_new, draws$latent, type = "l", lty = 1, ylab = "f(x)")
points(x, y, pch = 19)

# Noisy observation draws are generated from the same latent draws.
noisy <- sample_gp_posterior(
  model,
  x_new,
  n_draws = 1,
  type = "observation",
  seed = 2
)
head(cbind(latent = noisy$latent[, 1], observation = noisy$observation[, 1]))
```

sample_gp_prior

Draw functions from a Gaussian-process prior

Description

Draws latent function values $f(x) \sim \mathcal{N}(m(x), K(x, x))$ at the supplied inputs. Use [simulate_gp_data\(\)](#) for a single prior draw with added observation noise.

Usage

```
sample_gp_prior(
  x,
  kernel,
  mean = zero_mean(),
  n_draws = 1L,
  seed = NULL,
```

```

    initial_jitter = 0,
    fallback_jitter = 1e-10,
    jitter_multiplier = 10,
    max_attempts = 8L,
    symmetry_tolerance = sqrt(.Machine$double.eps)
)

```

Arguments

<code>x</code>	Numeric vector or matrix of inputs. Matrix rows are input points.
<code>kernel</code>	A Gaussian-process kernel specification, including ARD and composite kernels.
<code>mean</code>	Gaussian-process mean specification. Its coefficients must be fixed or have a Gaussian <code>coefficient_prior()</code> , whose uncertainty the draws include.
<code>n_draws</code>	Number of independent function draws.
<code>seed</code>	Optional non-negative integer seed. The caller's random state is restored afterwards.
<code>initial_jitter</code>	Non-negative jitter tried first, relative to the covariance scale (the mean of its diagonal).
<code>fallback_jitter</code>	Positive relative jitter tried after the first failed factorization.
<code>jitter_multiplier</code>	Multiplicative jitter escalation factor.
<code>max_attempts</code>	Maximum Cholesky attempts.
<code>symmetry_tolerance</code>	Relative tolerance for checking that the covariance matrix is symmetric.

Details

Each draw is $m(x) + R^\top z$ with $K(x, x) = R^\top R$ and $z \sim \mathcal{N}(0, I)$. Prior covariances on closely spaced inputs are often numerically singular; the Cholesky factor then uses the package's jitter policy, and the jitter is reported in the result. A single draw with the same seed equals the latent values of `simulate_gp_data()`.

Value

An object of class `gaussianprocesses_samples`. Its latent element is a matrix with one row per input point and one column per draw. It also contains the prior mean, the `latent_covariance` that was sampled, the numerical `jitter`, the `relative_jitter` and `covariance_scale` it was computed from (jitter is relative jitter times the mean diagonal of the covariance), and the `cholesky_attempts`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
x <- seq(0, 5, length.out = 100)
draws <- sample_gp_prior(x, rbf_kernel(length_scale = 0.8), n_draws = 3, seed = 1)
draws

matplot(x, draws$latent, type = "l", lty = 1, ylab = "f(x)")
```

scale_kernel

Scale a covariance kernel

Description

Multiplies a covariance kernel by a positive scalar. The scalar is retained as a named parameter and can therefore be optimized together with parameters of nested kernels.

Usage

```
scale_kernel(kernel, scale = 1)
```

Arguments

kernel	A Gaussian-process kernel specification.
scale	Positive scalar multiplier.

Value

A composite kernel specification.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- scale_kernel(rbf_kernel(), scale = 3)
all.equal(
  evaluate_kernel(kernel, 1:3),
  3 * evaluate_kernel(rbf_kernel(), 1:3)
)
```

select_dimensions	<i>Restrict a kernel to selected input columns</i>
-------------------	--

Description

Makes a kernel act on some input columns only: $k_S(x, x') = k(x_S, x'_S)$ for the selected columns S . Sums of restricted kernels give additive models such as $f(x) = f_1(x_1) + f_2(x_2, x_3)$.

Usage

```
select_dimensions(kernel, columns)
```

Arguments

kernel	A Gaussian-process kernel specification.
columns	Positive integer indices of the input columns the kernel acts on, without duplicates.

Details

A restricted kernel is positive semidefinite whenever kernel is, because it is kernel composed with a linear projection. A sum $\sum_j k_j(x_{S_j}, x'_{S_j})$ is the covariance of $f(x) = \sum_j f_j(x_{S_j})$ with independent components f_j .

The selection is not a hyperparameter. It adds no parameter and no level to parameter paths: `select_dimensions(rbf_kernel(), 2)` has the paths `variance` and `length_scale`, like `rbf_kernel()`. ARD parameters of kernel have one value per selected column.

Columns index the inputs the kernel receives. The number of input columns is not known when a specification is built, so column indices and ARD lengths are checked when the kernel is evaluated and when a model is fitted. Derivatives with respect to unselected input columns are zero.

Value

A composite kernel specification.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
# An additive model f(x) = f1(x1) + f2(x2, x3).
kernel <- sum_kernel(
  select_dimensions(rbf_kernel(length_scale = 0.5), 1),
  select_dimensions(matern52_kernel(length_scale = c(1, 2)), 2:3)
)
```

```

kernel
names(kernel_parameters(kernel, flatten = TRUE))

x <- cbind(c(0, 0.5, 1), c(1, 0, -1), c(2, 2, 0))
all.equal(
  evaluate_kernel(kernel, x),
  evaluate_kernel(rbf_kernel(length_scale = 0.5), x[, 1]) +
  evaluate_kernel(matern52_kernel(length_scale = c(1, 2)), x[, 2:3])
)

```

select_inducing_points

Select inducing points deterministically

Description

Select inducing points deterministically

Usage

```

select_inducing_points(
  x,
  n_inducing,
  method = c("farthest", "quantile", "variance"),
  kernel = NULL
)

```

Arguments

x	Numeric vector or matrix of candidate inputs.
n_inducing	Number of inducing points to select.
method	Selection method. "farthest" performs deterministic farthest-point sampling after column standardization. "quantile" selects points near equally spaced empirical quantiles and is available only for one-dimensional inputs. "variance" performs greedy conditional-variance selection under kernel.
kernel	Kernel specification, needed by method = "variance".

Details

Repeated input rows are offered as candidates only once, so the selected inducing locations are always distinct. n_inducing therefore cannot exceed the number of distinct rows in x.

Greedy conditional-variance selection (Burt, Rasmussen, and van der Wilk, 2019, 2020) adds, one at a time, the input whose prior variance is least explained by the points chosen so far: the largest residual $k(x, x) - Q(x, x)$ with Q the Nyström approximation from the chosen points. This is the pivot order of a pivoted Cholesky decomposition of $K(X, X)$, computed incrementally without forming $K(X, X)$ in $O(nm^2)$ time and $O(nm)$ memory. It minimizes the trace term $\text{tr}(K_{ff} - Q_{ff})$ greedily, so it is the natural choice for VFE, whose bound loses half that trace over the noise

variance. If the prior variance is explained, to a relative 10^{-12} , before `n_inducing` points are chosen, the selection stops with a warning of class `gaussianprocesses_inducing_warning` and returns fewer points.

Value

A numeric matrix with one inducing point per row. The selected training-row indices are stored in the `indices` attribute.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2019). Rates of convergence for sparse variational Gaussian process regression. *Proceedings of the 36th International Conference on Machine Learning*, 862–871.

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2020). Convergence of sparse variational inference in Gaussian processes regression. *Journal of Machine Learning Research*, 21(131), 1–63.

Examples

```
x <- seq(0, 10, length.out = 101)

inducing <- select_inducing_points(x, n_inducing = 5, method = "quantile")
inducing[, 1]
attr(inducing, "indices")
```

simulate_gp_data	<i>Simulate observations from a Gaussian process</i>
------------------	--

Description

Draws a latent Gaussian-process realization at supplied inputs and then adds independent Gaussian observation noise. The kernel, mean, noise variances, and numerical jitter used for simulation are retained in the result.

Usage

```
simulate_gp_data(
  x,
  kernel,
  mean = zero_mean(),
  noise_variance = 0,
  seed = NULL,
  initial_jitter = 0,
```

```

    fallback_jitter = 1e-10,
    jitter_multiplier = 10,
    max_attempts = 8L,
    symmetry_tolerance = sqrt(.Machine$double.eps)
  )

```

Arguments

<code>x</code>	Numeric vector or matrix of simulation inputs.
<code>kernel</code>	Gaussian-process kernel specification.
<code>mean</code>	Gaussian-process mean specification. Its coefficients must be fixed or have a Gaussian coefficient_prior() , whose uncertainty the simulated latent function includes.
<code>noise_variance</code>	Non-negative scalar or one non-negative value per observation.
<code>seed</code>	Optional non-negative integer seed. The caller's global random state is restored after simulation.
<code>initial_jitter</code>	Non-negative jitter tried first, relative to the covariance scale (the mean of its diagonal).
<code>fallback_jitter</code>	Positive relative jitter tried after the first failed factorization.
<code>jitter_multiplier</code>	Multiplicative jitter escalation factor.
<code>max_attempts</code>	Maximum Cholesky attempts.
<code>symmetry_tolerance</code>	Relative tolerance for checking that the covariance matrix is symmetric.

Value

An object of class `gaussianprocesses_simulation`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```

simulation <- simulate_gp_data(
  seq(0, 1, length.out = 30),
  kernel = rbf_kernel(length_scale = 0.2),
  noise_variance = 0.01,
  seed = 1
)
simulation

head(cbind(latent = simulation$latent, observed = simulation$observed))

```

spectral_mixture_kernel

Spectral-mixture kernels

Description

A spectral-mixture kernel (Wilson and Adams, 2013) is a sum of Q components with Gaussian spectral densities. With enough components it approximates any stationary covariance.

Usage

```
spectral_mixture_kernel(
  n_components = 1L,
  weights = 1/n_components,
  frequencies = 0,
  spectral_variances = 1
)
```

Arguments

n_components	Number of components Q .
weights	Positive component weights w_q : one value, recycled, or one per component.
frequencies	Frequencies μ_q , in cycles per input unit: one value, one per component, or a matrix with one row per component and one column per input dimension.
spectral_variances	Positive spectral variances v_q , in squared cycles per input unit, shaped like frequencies.

Details

By Bochner's theorem a stationary kernel is the Fourier transform of a non-negative spectral density. The spectral mixture uses $S(s) = \sum_q \frac{w_q}{2} [N(s; \mu_q, V_q) + N(s; -\mu_q, V_q)]$, whose transform is

$$k(\tau) = \sum_{q=1}^Q w_q \prod_{d=1}^D \exp(-2\pi^2 \tau_d^2 v_{qd}) \cos(2\pi \tau_d \mu_{qd}).$$

So $k(0) = \sum_q w_q$. A component with $\mu_q = 0$ is an RBF kernel with variance w_q and length scale $\ell = 1/(2\pi\sqrt{v_q})$; a component with a small spectral variance is a long-lived oscillation with period $1/\mu_q$.

Each component is a kernel with parameters weight, frequency, and spectral_variance, and the mixture is their `sum_kernel()`, so parameter paths are `kernel1.weight`, `kernel1.frequency`, and so on; a single component has no prefix. Frequencies are real-valued and optimized on the identity scale, the others on the log scale. Because $\cos$ is even, $-\mu$ and μ give the same kernel. `optimize_gp()` bounds every frequency by 0 and the Nyquist frequency $1/(2\Delta)$ of its input column, with Δ the smallest spacing between distinct inputs, and displaces it in later starts by one frequency-resolution bin, the reciprocal of the input range.

The likelihood is strongly multimodal: components can swap, merge, or settle on harmonics. Start from the data's spectrum with `initialize_spectral_mixture()`, and use several starts; the examples use `n_starts = 5`.

Value

A kernel specification: one spectral component, or the sum of `n_components` of them.

Stability

Experimental: this interface may change in a minor release, and every change is listed in NEWS. See [gaussianprocesses-package](#) for the policy.

References

Wilson, A. G. and Adams, R. P. (2013). Gaussian process kernels for pattern discovery and extrapolation. *Proceedings of the 30th International Conference on Machine Learning*, 1067–1075.

See Also

`initialize_spectral_mixture()`.

Examples

```
kernel <- spectral_mixture_kernel(
  n_components = 2,
  weights = c(1, 0.5),
  frequencies = c(0, 0.25),
  spectral_variances = c(0.01, 0.001)
)
kernel

# k(0) is the sum of the weights.
evaluate_kernel(kernel, 0)

# A component with frequency 0 is an RBF kernel.
all.equal(
  evaluate_kernel(spectral_mixture_kernel(1, 2, 0, 1 / (4 * pi^2)), 0:3),
  evaluate_kernel(rbf_kernel(variance = 2, length_scale = 1), 0:3)
)
```

sum_kernel

Add covariance kernels

Description

The covariance matrices generated by all supplied kernels are added element-wise.

Usage

```
sum_kernel(...)
```

Arguments

... Two or more Gaussian-process kernel specifications.

Value

A composite kernel specification.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- sum_kernel(linear_kernel(variance = 0.1), periodic_kernel(period = 12))
kernel

names(kernel_parameters(kernel, flatten = TRUE))
```

time_series_kernel	<i>Create a trend-periodic-local time-series kernel</i>
--------------------	---

Description

Constructs a transparent additive example consisting of a nonstationary linear covariance component, a periodic covariance component, and a local Matérn-3/2 component. The components are explicit so stationarity is never implied for the full model.

Usage

```
time_series_kernel(
  period,
  trend_variance = 1,
  periodic_variance = 1,
  periodic_length_scale = 1,
  local_variance = 1,
  local_length_scale = 1,
  changepoint = NULL,
  changepoint_steepness = 1
)
```

Arguments

period	Positive period in the same units as the time index.
trend_variance	Positive variance of the linear trend component.
periodic_variance	Positive variance of the periodic component.
periodic_length_scale	Positive periodic length scale.
local_variance	Positive variance of the local Matérn component.
local_length_scale	Positive local Matérn length scale.
changepoint	Optional location of a structural break, on the time index scale of <code>gp_time_index()</code> : elapsed time since the first training time, in its unit.
changepoint_steepness	Positive steepness of the break.

Details

With a changepoint, the kernel is a `changepoint_kernel()` between two independent copies of the trend-periodic-local kernel, so every component can change at the break; the copies start from the same parameter values and are estimated separately. To place a break at a date, convert it with the training times: `gp_time_index(c(time[1], date), unit)[2]`.

Value

A composite `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy. The changepoint argument, which builds a `changepoint_kernel()`, is experimental.

Examples

```
kernel <- time_series_kernel(period = 12)
kernel

# A structural break 30 time units after the first observation.
names(kernel_parameters(
  time_series_kernel(period = 12, changepoint = 30),
  flatten = TRUE
))
```

`update_kernel_parameters`*Update parameters in a kernel specification*

Description

Parameters are identified by the names returned by `kernel_parameters(kernel, flatten = TRUE)`. Partial updates are supported, which allows an optimizer or tuning routine to modify selected parameters without manually unpacking nested kernel objects.

Usage

```
update_kernel_parameters(kernel, parameters)
```

Arguments

<code>kernel</code>	A Gaussian-process kernel specification.
<code>parameters</code>	Named numeric vector of parameter updates.

Value

A new kernel specification with updated parameters.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

Examples

```
kernel <- sum_kernel(
  rbf_kernel(length_scale = c(1, 2)),
  white_noise_kernel(variance = 0.1)
)

updated <- update_kernel_parameters(
  kernel,
  c("kernel1.length_scale[2]" = 5, "kernel2.variance" = 0.01)
)
kernel_parameters(updated, flatten = TRUE)
```

white_noise_kernel	Create a white-noise kernel specification
--------------------	---

Description

Create a white-noise kernel specification

Usage

```
white_noise_kernel(variance = 1)
```

Arguments

variance	Positive white-noise variance.
----------	--------------------------------

Value

An object of class `gaussianprocesses_kernel`.

Stability

Stable: from version 1.0.0 this interface changes incompatibly only in a major release, after a deprecation period. Results and options that concern an experimental model class, kernel, or argument follow that interface's tier. See [gaussianprocesses-package](#) for the policy.

See Also

[kernel_white_noise\(\)](#) computes the same covariance directly from parameter values, without a kernel specification.

Examples

```
# Add an independent nugget to a smooth kernel.
kernel <- sum_kernel(rbf_kernel(), white_noise_kernel(variance = 0.05))
evaluate_kernel(kernel, c(0, 0.5))
```

Index

basis_mean (parametric_means), 85
basis_mean(), 41, 73, 91
benchmark_exact_gp_scaling, 3
benchmark_sparse_gp, 4
bernoulli_likelihood (gp_likeliheids), 39
bernoulli_likelihood(), 33

changepoint_kernel, 6
changepoint_kernel(), 77, 116
chol2inv(), 70
coef(), 14, 21, 29, 86
coef.gaussianprocesses_latent_model, 7
coef.gaussianprocesses_model
 (coef.gaussianprocesses_latent_model), 7
coefficient_prior (mean_coefficients), 73
coefficient_prior(), 20, 41, 79, 85, 107, 112
compare_gp_predictions, 8
constant_mean (gp_mean_functions), 41
constant_mean(), 15, 73
coregionalization_kernel, 9
coregionalization_kernel(), 11, 52, 54
coregionalization_matrix, 11
coregionalization_matrix(), 10

estimate_coefficients
 (mean_coefficients), 73
estimate_coefficients(), 41, 85
evaluate_kernel, 12
evaluate_likelihood, 13
evaluate_likelihood(), 40, 79
evaluate_mean, 14

fit_gp, 15
fit_gp(), 19–21, 25, 29, 30, 40, 77, 79, 84, 86, 104
fit_heteroscedastic_gp, 17
fit_heteroscedastic_gp(), 29, 104
fit_latent_gp, 19
fit_latent_gp(), 8, 29, 30, 40, 44, 49, 69, 79, 80, 93, 95, 104
fit_sparse_gp, 22
fit_sparse_gp(), 29, 81, 83, 95, 104, 105
fit_time_series_gp, 25
fit_time_series_gp(), 27, 29, 30, 44, 69, 84, 88, 100, 104

fitted.gaussianprocesses_heteroscedastic_model
 (gp_model_methods), 42
fitted.gaussianprocesses_latent_model
 (gp_model_methods), 42
fitted.gaussianprocesses_model
 (gp_model_methods), 42
fitted.gaussianprocesses_sparse_model
 (gp_model_methods), 42
fitted.gaussianprocesses_state_space_model
 (gp_model_methods), 42
forecast_gp, 27
forecast_gp(), 26, 44, 89
forecast_metrics, 28
forecast_metrics(), 50

gaussian_likelihood (gp_likeliheids), 39
gaussian_likelihood(), 19, 79
gaussianprocesses-package, 4–6, 8–14, 16, 19, 22, 24, 26, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 45, 46, 48, 50–54, 56, 58–66, 68, 70–74, 78, 80, 83, 85–87, 89, 91, 92, 95–99, 101, 102, 106–109, 111, 112, 114–118
gaussianprocesses_heteroscedastic_model
 (gaussianprocesses_model), 29
gaussianprocesses_latent_model
 (gaussianprocesses_model), 29
gaussianprocesses_model, 16, 19, 25, 29
gaussianprocesses_sparse_model
 (gaussianprocesses_model), 29

- gaussianprocesses_state_space_model
 (gaussianprocesses_model), 29
- gp_calibration_diagnostics, 31
- gp_calibration_diagnostics(), 50
- gp_classification_scores, 32
- gp_classification_scores(), 36, 49
- gp_coefficients
 (coef.gaussianprocesses_latent_model), 7
- gp_count_scores, 34
- gp_count_scores(), 40, 49, 95
- gp_diagnostics, 36
- gp_holdout_scores (gp_scores), 48
- gp_holdout_scores(), 32–34
- gp_hyperparameter_uncertainty, 37
- gp_hyperparameter_uncertainty(), 77
- gp_likelihoods, 13, 20, 39, 67
- gp_loo_scores (gp_scores), 48
- gp_mean_functions, 41
- gp_model_methods, 42
- gp_numerical_diagnostics, 45
- gp_numerical_diagnostics(), 16, 37, 72
- gp_profile_likelihood, 47
- gp_profile_likelihood(), 38, 39
- gp_scores, 48
- gp_scores(), 34, 36
- gp_simulation_scenario, 51
- gp_stack_outputs, 52
- gp_stack_outputs(), 10, 54, 77
- gp_time_index, 53
- gp_time_index(), 25, 84, 100, 116
- initialize_coregionalization, 54
- initialize_coregionalization(), 10
- initialize_spectral_mixture, 55
- initialize_spectral_mixture(), 114
- kernel_diagonal, 56
- kernel_diagonal(), 89
- kernel_gradient, 57
- kernel_gradient(), 59, 70
- kernel_input_gradient, 58
- kernel_input_gradient(), 6, 16, 58, 82, 83, 91
- kernel_linear, 59
- kernel_linear(), 69
- kernel_matern, 60
- kernel_matern12 (kernel_matern), 60
- kernel_matern12(), 73
- kernel_matern32 (kernel_matern), 60
- kernel_matern32(), 73
- kernel_matern52 (kernel_matern), 60
- kernel_matern52(), 73
- kernel_parameters, 61
- kernel_periodic, 62
- kernel_periodic(), 87
- kernel_rational_quadratic, 63
- kernel_rational_quadratic(), 98
- kernel_rbf, 65
- kernel_rbf(), 12, 99
- kernel_white_noise, 66
- kernel_white_noise(), 118
- likelihood_predictive, 67
- likelihood_predictive(), 40, 49, 94
- linear_kernel, 68
- linear_kernel(), 60
- linear_mean (parametric_means), 85
- linear_mean(), 8, 15, 21, 41, 73, 75
- log_marginal_likelihood, 69
- log_marginal_likelihood(), 16, 21, 44, 70, 71, 74, 77
- log_marginal_likelihood_gradient, 70
- log_marginal_likelihood_gradient(), 21, 38, 77
- logLik(), 21
- logLik.gaussianprocesses_heteroscedastic_model
 (gp_model_methods), 42
- logLik.gaussianprocesses_latent_model
 (gp_model_methods), 42
- logLik.gaussianprocesses_model
 (gp_model_methods), 42
- logLik.gaussianprocesses_sparse_model
 (gp_model_methods), 42
- logLik.gaussianprocesses_state_space_model
 (gp_model_methods), 42
- loo_gp, 71
- loo_gp(), 49
- matern12_kernel (matern_kernel_specs), 72
- matern32_kernel (matern_kernel_specs), 72
- matern52_kernel (matern_kernel_specs), 72
- matern_kernel_specs, 61, 72
- mean_coefficients, 8, 15, 69, 73, 76, 86, 89

`nobs.gaussianprocesses_heteroscedastic_model` `(gp_model_methods)`, 42
`nobs.gaussianprocesses_latent_model` `(gp_model_methods)`, 42
`nobs.gaussianprocesses_model` `(gp_model_methods)`, 42
`nobs.gaussianprocesses_sparse_model` `(gp_model_methods)`, 42
`nobs.gaussianprocesses_state_space_model` `(gp_model_methods)`, 42

`optimize_gp`, 75
`optimize_gp()`, 6, 16, 29, 30, 37, 38, 44, 47, 55, 74, 78–84, 86, 104, 113
`optimize_latent_gp`, 78
`optimize_latent_gp()`, 22, 29, 40, 44, 69, 71, 93, 104
`optimize_sparse_gp`, 81
`optimize_sparse_gp()`, 29, 30, 44, 104
`optimize_time_series_gp`, 84
`optimize_time_series_gp()`, 26, 27, 44

`parametric_means`, 85
`periodic_kernel`, 87
`periodic_kernel()`, 63
`poisson_likelihood(gp_likelihooods)`, 39
`poisson_likelihood()`, 20, 34, 79
`polynomial_mean(parametric_means)`, 85
`polynomial_mean()`, 41, 73
`predict.gaussianprocesses_heteroscedastic_model` `(gp_model_methods)`, 42
`predict.gaussianprocesses_latent_model` `(gp_model_methods)`, 42
`predict.gaussianprocesses_model` `(gp_model_methods)`, 42
`predict.gaussianprocesses_sparse_model` `(gp_model_methods)`, 42
`predict.gaussianprocesses_state_space_model` `(gp_model_methods)`, 42
`predict_gp`, 88
`predict_gp()`, 5, 14, 16, 27, 44, 72, 90–92, 94, 96, 105
`predict_gradient_gp`, 90
`predict_gradient_gp()`, 6, 16, 59
`predict_heteroscedastic_gp`, 92
`predict_heteroscedastic_gp()`, 44, 89, 105
`predict_latent_gp`, 93
`predict_latent_gp()`, 22, 34–36, 44, 104, 105
`predict_sparse_gp`, 95
`predict_sparse_gp()`, 44, 89, 105
`print.summary.gaussianprocesses_model` `(gp_model_methods)`, 42
`product_kernel`, 97
`product_kernel()`, 12

`rational_quadratic_kernel`, 98
`rational_quadratic_kernel()`, 64
`rbf_kernel`, 99
`rbf_kernel()`, 12, 65
`readRDS()`, 31
`residuals.gaussianprocesses_heteroscedastic_model` `(gp_model_methods)`, 42
`residuals.gaussianprocesses_model` `(gp_model_methods)`, 42
`residuals.gaussianprocesses_sparse_model` `(gp_model_methods)`, 42
`residuals.gaussianprocesses_state_space_model` `(gp_model_methods)`, 42
`rolling_origin_gp`, 100
`rolling_origin_gp()`, 28
`run_gp_benchmark_suite`, 101

`sample_gp_posterior` `(sample_gp_posterior.gaussianprocesses_latent_model)`, 102
`sample_gp_posterior()`, 22, 26, 89, 94, 95
`sample_gp_posterior.gaussianprocesses_latent_model`, 102
`sample_gp_prior`, 106
`saveRDS()`, 31
`scale_kernel`, 108
`scale_kernel()`, 12
`select_dimensions`, 109
`select_dimensions()`, 10
`select_inducing_points`, 110
`select_inducing_points()`, 23, 82
`simulate_gp_data`, 111
`simulate_gp_data()`, 106, 107
`solve()`, 71
`spectral_mixture_kernel`, 113
`spectral_mixture_kernel()`, 55, 56, 77
`stats::AIC()`, 44
`stats::BIC()`, 44
`stats::fitted()`, 42
`stats::glm()`, 40

`stats::logLik()`, 42
`stats::nobs()`, 42
`stats::optim()`, 76, 77
`stats::predict()`, 42, 49
`stats::residuals()`, 42
`stats::spec.pgram()`, 55
`sum_kernel`, 114
`sum_kernel()`, 12, 113
`summary()`, 42, 77
`summary.gaussianprocesses_heteroscedastic_model`
 (`gp_model_methods`), 42
`summary.gaussianprocesses_latent_model`
 (`gp_model_methods`), 42
`summary.gaussianprocesses_model`
 (`gp_model_methods`), 42
`summary.gaussianprocesses_sparse_model`
 (`gp_model_methods`), 42
`summary.gaussianprocesses_state_space_model`
 (`gp_model_methods`), 42

`time_series_kernel`, 115
`time_series_kernel()`, 7

`update_kernel_parameters`, 117
`utils::object.size()`, 102

`vcov()`, 29, 86
`vcov.gaussianprocesses_model`
 (`coef.gaussianprocesses_latent_model`),
 7

`white_noise_kernel`, 118
`white_noise_kernel()`, 66

`zero_mean(gp_mean_functions)`, 41
`zero_mean()`, 15