Package ‘DNAmf’

June 23, 2025

Type Package

Title Diffusion Non-Additive Model with Tunable Precision

Version 0.1.0

Maintainer Junoh Heo <heojunoh@msu.edu>

Description Performs Diffusion Non-Additive (DNA) model proposed by Heo, Boutelet, and Sung (2025+) <doi:10.48550/arXiv.2506.08328> for multi-fidelity computer experiments with tuning parameters. The DNA model captures nonlinear dependencies across fidelity levels using Gaussian process priors and is particularly effective when simulations at different fidelity levels are nonlinearly correlated. The DNA model targets not only interpolation across given fidelity levels but also extrapolation to smaller tuning parameters including the exact solution corresponding to a zero-valued tuning parameter, leveraging a nonseparable covariance kernel structure that models interactions between the tuning parameter and input variables. Closed-form expressions for the predictive mean and variance enable efficient inference and uncertainty quantification. Hyperparameters in the model are estimated via maximum likelihood estimation.

License GPL-3

Encoding UTF-8

Imports plgp, stats, methods, lhs, fields, mvtnorm

Suggests RNAmf

RoxygenNote 7.3.2

NeedsCompilation no

Author Junoh Heo [aut, cre],
Romain Boutelet [aut],
Chih-Li Sung [aut]

Repository CRAN

Date/Publication 2025-06-23 11:20:06 UTC

Contents

`closed_form` *Closed-form prediction for DNAmf model*

Description

The function computes the closed-form posterior mean and variance for the DNAmf model both at the fidelity levels used in model fitting and at any user-specified target fidelity level, using the chosen nonseparable kernel.

Usage

```
closed_form(
  fit1,
  fit2,
  targett,
  kernel,
  nn,
  tt,
  nlevel,
  x,
  XX = NULL,
  pseudo_yy = NULL
)
```

Arguments

`fit1`	A fitted GP object for f_1.
`fit2`	A fitted GP object for f.
`targett`	A numeric value of target tuning parameter to predict.
`kernel`	A character specifying the kernel type to be used. Choices are `"sqex"`(nonseparable squared exponential kernel), `"matern1.5"`(nonseparable Matern kernel with $\nu = 1.5$), or `"matern2.5"`(nonseparable Matern kernel with $\nu = 2.5$). Default is `"sqex"`.
`nn`	A vector specifying the number of design points at each fidelity level.
`tt`	A vector of tuning parameters for each fidelity level.
`nlevel`	The number of fidelity levels L.
`x`	A vector or matrix of new input locations to predict.
`XX`	A list containing a pseudo-complete inputs `X_star`($\{\mathcal{X}^*_l\}_{l=1}^L$), an original inputs `X_list`($\{\mathcal{X}_l\}_{l=1}^L$), and a pseudo inputs `X_tilde`($\left\{\widetilde{\mathcal{X}}_l\right\}_{l=1}^L$) for non-nested design.
`pseudo_yy`	A list containing a pseudo-complete outputs `y_star`($\{\mathbf{y}^*_l\}_{l=1}^L$), an original outputs `y_list`($\{\mathbf{y}_l\}_{l=1}^L$), and a pseudo outputs `y_tilde`($\{\widetilde{\mathbf{y}}_l\}_{l=1}^L$) imputed by `imputer`.

Value

A list of predictive posterior mean and variance for each level containing:

- `mu_1`, `sig2_1`, ..., `mu_L`, `sig2_L`: A vector of predictive posterior mean and variance at each level.
- `mu`: A vector of predictive posterior mean at target tuning parameter.
- `sig2`: A vector of predictive posterior variance at target tuning parameter.

`DNAmf` *Fitting a Diffusion Non-Additive model for multi-fidelity computer experiments with tuning parameters*

Description

The function fits DNA models for multi-fidelity computer experiments with tuning parameters. Available kernel choices include nonseparable squared exponential kernel, and nonseparable Matern kernel with smoothness parameter 1.5 and 2.5. The function returns a fitted model object of class `DNAmf`, produced by `DNAmf_internal`.

Usage

```
DNAmf(X, y, kernel = "sqex", t, constant = TRUE, init=NULL,
n.iter=50, multi.start=10, g = sqrt(.Machine$double.eps), burn.ratio = 0.75, ...)
```

Arguments

`X`	A list of input locations for all fidelity levels $1, \ldots, L$ combined.
`y`	A list of response values for all fidelity levels $1, \ldots, L$ combined.
`kernel`	A character specifying the kernel type to be used. Choices are `"sqex"`(squared exponential), `"matern1.5"`, or `"matern2.5"`. Default is `"sqex"`.
`t`	A vector of tuning parameters for each fidelity level.
`constant`	A logical indicating for constant mean of GP (`constant=TRUE`) or zero mean (`constant=FALSE`). Default is `TRUE`.
`init`	Optional vector of initial parameter values for optimization. Default is `NULL`.
`n.iter`	Number of iterations for the stochastic EM algorithm for non-nested designs. Default is 50.
`multi.start`	Number of random starting points for optimization. Default is 10.
`g`	Nugget term for numerical stability. Default is `sqrt(.Machine$double.eps)`.
`burn.ratio`	Fraction of iterations to discard as burn-in. Default is 0.75.
`...`	Additional arguments for compatibility with `DNAmf_internal`.

Details

The `DNAmf` function internally calls `DNAmf_internal` to fit the DNA model with nonseparable kernel.

The model structure is: $\begin{cases} f_1(\boldsymbol{x}) = W_1(\boldsymbol{x}), \\ f_l(\boldsymbol{x}) = W(t_l, \boldsymbol{x}, f_{l-1}(\boldsymbol{x})), \end{cases}$ where $W(t, \boldsymbol{x}, y) \sim GP(\alpha, \tau^2 K((t, \boldsymbol{x}, y), (t', \boldsymbol{x}', y')))$ is a GP model. Hyperparameters $(\alpha, \tau^2, \boldsymbol{\theta})$ are estimated by maximizing the log-likelihood via an optimization algorithm "L-BFGS-B". For `constant=FALSE`, $\alpha = 0$.

The nonseparable covariance kernel is defined as:

$$K((t, \boldsymbol{x}, y), (t', \boldsymbol{x}', y')) = \left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{-\left(\frac{\beta(d+1)}{2} + \delta\right)} \prod_{j=1}^{d} \phi(x_j, x'_j; \theta_j) \phi(y, y'; \theta_y),$$

where $\phi(\cdot, \cdot)$ depens on the chosen kernel:

- For nonseparable squared exponential kernel(`kernel = "sqex"`):

$$\phi(x, x'; \theta) = \exp\left(-\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{-\beta} \frac{(x-x')^2}{\theta} \right)$$

- For nonseparable Matern kernel with smoothness parameter of $\nu = 1.5$ (`kernel = "matern1.5"`):

$$\phi(x, x'; \theta) = \left(1 + \frac{1}{\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{\frac{\beta}{2}}} \frac{\sqrt{3}|x-x'|}{\theta} \right) \exp\left(-\frac{1}{\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{\frac{\beta}{2}}} \frac{\sqrt{3}|x-x'|}{\theta} \right)$$

- For nonseparable Matern kernel with smoothness parameter of $\nu = 2.5$ (`kernel = "matern2.5"`):

$$\phi(x, x'; \theta) = \left(1 + \frac{1}{\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{\frac{\beta}{2}}} \frac{\sqrt{5}|x-x'|}{\theta} + \frac{1}{3} \left(\frac{1}{\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{\frac{\beta}{2}}} \frac{\sqrt{5}|x-x'|}{\theta} \right)^2 \right)$$
$$\times \exp\left(-\frac{1}{\left(\frac{(t-t')^2}{\theta_t} + 1 \right)^{\frac{\beta}{2}}} \frac{\sqrt{5}|x-x'|}{\theta} \right)$$

When the input locations are not nested, the internal `makenested` function constructs nested designs as $\mathcal{X}^*_L = \mathcal{X}_L$ and $\mathcal{X}^*_l = \mathcal{X}_l \cup \mathcal{X}^*_{l+1}$ for $l = 1, \ldots, L-1$. The function `imputer` then imputes pseudo outputs $\widetilde{\mathbf{y}}_l := f_l(\widetilde{\mathcal{X}}_l)$ at pseudo inputs $\widetilde{\mathcal{X}}_l := \mathcal{X}^*_l \setminus \mathcal{X}_l$, using a stochastic EM algorithm.

For further details, see Heo, Boutelet, and Sung (2025+, <arXiv:2506.08328>).

Value

A fitted model object of class `DNAmf`.

See Also

predict.DNAmf for prediction.

Examples

```
### Non-Additive example ###
library(RNAmf)

### Non-Additive Function ###
fl <- function(x, t){
  term1 <- sin(10 * pi * x / (5+t))
  term2 <- 0.2 * sin(8 * pi * x)
  term1 + term2
}

### training data ###
n1 <- 13; n2 <- 10; n3 <- 7; n4 <- 4; n5 <- 1;
m1 <- 2.5; m2 <- 2.0; m3 <- 1.5; m4 <- 1.0; m5 <- 0.5;
d <- 1

### fix seed to reproduce the result ###
set.seed(1)

### generate initial nested design ###
NestDesign <- NestedX(c(n1,n2,n3,n4,n5),d)

X1 <- NestDesign[[1]]
X2 <- NestDesign[[2]]
X3 <- NestDesign[[3]]
X4 <- NestDesign[[4]]
X5 <- NestDesign[[5]]

y1 <- fl(X1, t=m1)
y2 <- fl(X2, t=m2)
y3 <- fl(X3, t=m3)
y4 <- fl(X4, t=m4)
y5 <- fl(X5, t=m5)

### fit a DNAmf ###
fit.DNAmf <- DNAmf(X=list(X1, X2, X3, X4, X5), y=list(y1, y2, y3, y4, y5), kernel="sqex",
                   t=c(m1,m2,m3,m4,m5), multi.start=10, constant=TRUE)
```

---

imputer *Imputation step in stochastic EM for the non-nested DNA Model*

Description

The function performs the imputation step of the stochastic EM algorithm for the DNA model when the design is not nested. The function generates pseudo outputs $\widetilde{\mathbf{y}}_l$ at pseudo inputs $\widetilde{\mathcal{X}}_l$.

Usage

```
imputer(XX, yy, kernel=kernel, t, pred1, fit2)
```

Arguments

XX	A list of design sets for all fidelity levels, containing `X_star`, `X_list`, and `X_tilde`.
yy	A list of current observed and pseudo-responses, containing `y_star`, `y_list`, and `y_tilde`.
kernel	A character specifying the kernel type to be used. Choices are `"sqex"`(squared exponential), `"matern1.5"`, or `"matern2.5"`.
t	A vector of tuning parameters for each fidelity level.
pred1	Predictive results for the lowest fidelity level f_1. It should include `cov` obtained by setting `cov.out=TRUE`.
fit2	A fitted model object for higher fidelity levels f from (t_{-1}, X_{-1}, y_{-1}).

Details

For non-nested designs, pseudo-input locations $\widetilde{\mathcal{X}}_l$ are constructed using the internal `makenested` function. The `imputer` function then imputes the corresponding pseudo outputs $\widetilde{\mathbf{y}}_l = f_l(\widetilde{\mathcal{X}}_l)$ by drawing samples from the conditional normal distribution, given fixed parameter estimates and previous-level outputs $Y_{-L}^{*(m-1)}$, at the m-th iteration of the EM algorithm.

For further details, see Heo, Boutelet, and Sung (2025+, <arXiv:2506.08328>).

Value

An updated `yy` list containing:

- `y_star`: An updated pseudo-complete outputs $\mathbf{y}_l^*$.
- `y_list`: An original outputs $\mathbf{y}_l$.
- `y_tilde`: A newly imputed pseudo outputs $\widetilde{\mathbf{y}}_l$.

`predict.DNAmf` *Predictive posterior mean and variance for DNAmf object with non-separable kernel.*

Description

The function computes the predictive posterior mean and variance for the DNAmf model using closed-form expressions based on the chosen nonseparable kernel at given new input locations.

Usage

```
## S3 method for class 'DNAmf'
predict(object, x, targett = 0, nimpute = 50, ...)
```

Arguments

`object`	A fitted DNAmf object.
`x`	A vector or matrix of new input locations to predict.
`targett`	A numeric value of target tuning parameter to predict.
`nimpute`	Number of imputations for non-nested designs. Default is 50.
`...`	Additional arguments for compatibility with generic method `predict`.

Details

The `predict.DNAmf` function internally calls `closed_form`, which further calls `h1_sqex`, `h2_sqex`, `h2_sqex_single` for `kernel="sqex"`, or `h1_matern`, `h2_matern`, `h2_matern_single` for `kernel="matern1.5"` or`kernel="matern2.5"`, to recursively compute the closed-form posterior mean and variance at each level.

From the fitted model from `DNAmf`, the posterior mean and variance are calculated based on the closed-form expression derived by a recursive fashion. The formulas depend on its kernel choices.

If the fitted model was constructed with non-nested designs (`nested=FALSE`), the function generates `nimpute` sets of imputations for pseudo outputs via `imputer`.

For further details, see Heo, Boutelet, and Sung (2025+, <arXiv:2506.08328>).

Value

A list of predictive posterior mean and variance for each level and computation time containing:

- `mu_1`, `sig2_1`, ..., `mu_L`, `sig2_L`: A vector of predictive posterior mean and variance at each level.
- `mu`: A vector of predictive posterior mean at target tuning parameter.
- `sig2`: A vector of predictive posterior variance at target tuning parameter `targett`.
- `time`: Total computation time in seconds.

See Also

`DNAmf` for the user-level function.

Examples

```
### Non-Additive example ###
library(RNAmf)

### Non-Additive Function ###
fl <- function(x, t){
  term1 <- sin(10 * pi * x / (5+t))
  term2 <- 0.2 * sin(8 * pi * x)
  term1 + term2
}

### training data ###
n1 <- 13; n2 <- 10; n3 <- 7; n4 <- 4; n5 <- 1;
```

```
m1 <- 2.5; m2 <- 2.0; m3 <- 1.5; m4 <- 1.0; m5 <- 0.5;
d <- 1
eps <- sqrt(.Machine$double.eps)
x <- seq(0,1,0.01)

### fix seed to reproduce the result ###
set.seed(1)

### generate initial nested design ###
NestDesign <- NestedX(c(n1,n2,n3,n4,n5),d)

X1 <- NestDesign[[1]]
X2 <- NestDesign[[2]]
X3 <- NestDesign[[3]]
X4 <- NestDesign[[4]]
X5 <- NestDesign[[5]]

y1 <- fl(X1, t=m1)
y2 <- fl(X2, t=m2)
y3 <- fl(X3, t=m3)
y4 <- fl(X4, t=m4)
y5 <- fl(X5, t=m5)

### fit a DNAmf ###
fit.DNAmf <- DNAmf(X=list(X1, X2, X3, X4, X5), y=list(y1, y2, y3, y4, y5), kernel="sqex",
                   t=c(m1,m2,m3,m4,m5), multi.start=10, constant=TRUE)

### predict ###
pred.DNAmf <- predict(fit.DNAmf, x, targett=0)
predydiffu <- pred.DNAmf$mu
predsig2diffu <- pred.DNAmf$sig2

### RMSE ###
print(sqrt(mean((predydiffu-fl(x, t=0))^2))) # 0.1162579

### visualize the emulation performance ###
oldpar <- par(mfrow = c(2,3))
create_plot_base <- function(i, mesh_size, x, pred_mu, pred_sig2,
                             X_points = NULL, y_points = NULL, add_points = TRUE, yylim) {
  lower <- pred_mu - qnorm(0.995) * sqrt(pred_sig2)
  upper <- pred_mu + qnorm(0.995) * sqrt(pred_sig2)

  plot(x, pred_mu, type = "n", ylim = c(-yylim, yylim), xlab = "", ylab = "",
       main = paste0("Mesh size = ", mesh_size), axes = FALSE)
  box()

  polygon(c(x, rev(x)), c(upper, rev(lower)),
          col = adjustcolor("blue", alpha.f = 0.2), border = NA)
  lines(x, pred_mu, col = "blue", lwd = 2)
  lines(x, fl(x, mesh_size), lty = 2, col = "black", lwd = 2)

  if (add_points && !is.null(X_points) && !is.null(y_points)) {
    points(X_points, y_points, col = "red", pch = 16, cex = 1.3)
  }
}

mesh_sizes <- c(m1, m2, m3, m4, m5, 0)
mu_list <- list(pred.DNAmf$mu_1, pred.DNAmf$mu_2, pred.DNAmf$mu_3,
                pred.DNAmf$mu_4, pred.DNAmf$mu_5, pred.DNAmf$mu)
sig2_list <- list(pred.DNAmf$sig2_1, pred.DNAmf$sig2_2, pred.DNAmf$sig2_3,
                  pred.DNAmf$sig2_4, pred.DNAmf$sig2_5, pred.DNAmf$sig2)
X_list <- list(X1, X2, X3, X4, X5, NULL)
y_list <- list(y1, y2, y3, y4, y5, NULL)

plots <- mapply(function(i, m, mu, sig2, X, y) {
  create_plot_base(i, m, x, mu, sig2, X, y, add_points = !is.null(X), yylim=1.5)
}, i = 1:6, m = mesh_sizes, mu = mu_list, sig2 = sig2_list,
X = X_list, y = y_list, SIMPLIFY = FALSE)
par(oldpar)
```

Index