Package 'RNAmf'

May 7, 2026

Type Package

Title Recursive Non-Additive Emulator for Multi-Fidelity Data

Version 1.1.4

Maintainer Junoh Heo <heojunoh@msu.edu>

Description Performs RNA emulation and active learning proposed by Heo and Sung (2025) <doi:10.1080/00401706.2024.2376173> for multi-fidelity computer experiments. The RNA emulator is particularly useful when the simulations with different fidelity level are nonlinearly correlated. The hyperparameters in the model are estimated by maximum likelihood estimation.

License GPL-3

Encoding UTF-8

Imports plgp, stats, methods, lhs, doParallel, foreach, doRNG, fields, mvtnorm, ggplot2, ggpubr, scales

Suggests knitr, rmarkdown

RoxygenNote 7.3.3

NeedsCompilation no

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

Repository CRAN

Date/Publication 2026-05-05 05:20:02 UTC

Contents

AL_RNAmf *Active Learning for Recursive Non-Additive Emulator*

Description

The function acquires the new point and fidelity level by maximizing one of the four active learning criteria: ALM, ALC, ALD, or ALMC.

- **ALM** (Active Learning MacKay): It calculates the ALM criterion $\frac{\sigma_l^{*2}(\boldsymbol{x})}{\sum_{j=1}^{l} C_j}$, where $\sigma_l^{*2}(\boldsymbol{x})$ is the posterior predictive variance at each fidelity level l and C_j is the simulation cost at level j.
- **ALD** (Active Learning Decomposition): It calculates the ALD criterion $\frac{V_l(\boldsymbol{x})}{\sum_{j=1}^{l} C_j}$, where $V_l(\boldsymbol{x})$ is the variance contribution of GP emulator at each fidelity level l and C_j is the simulation cost at level j.
- **ALC** (Active Learning Cohn): It calculates the ALC criterion $\frac{\Delta\sigma_L^2(l,\boldsymbol{x})}{\sum_{j=1}^{l} C_j} = \frac{\int_\Omega \sigma_L^{*2}(\boldsymbol{\xi}) - \tilde{\sigma}_L^{*2}(\boldsymbol{\xi};l,\boldsymbol{x})\mathrm{d}\boldsymbol{\xi}}{\sum_{j=1}^{l} C_j}$, where f_L is the highest-fidelity simulation code, $\sigma_L^{*2}(\boldsymbol{\xi})$ is the posterior variance of $f_L(\boldsymbol{\xi})$, C_j is the simulation cost at fidelity level j, and $\tilde{\sigma}_L^{*2}(\boldsymbol{\xi};l,\boldsymbol{x})$ is the posterior variance based on the augmented design combining the current design and a new input location $\boldsymbol{x}$ at each fidelity level lower than or equal to l. The integration is approximated by MC integration using uniform reference samples.
- **ALMC** (Active Learning MacKay-Cohn): A hybrid approach. It finds the optimal input location $\boldsymbol{x}^*$ by maximizing $\sigma_L^{*2}(\boldsymbol{x})$, the posterior predictive variance at the highest-fidelity level L. After selecting $\boldsymbol{x}^*$, it finds the optimal fidelity level by maximizing ALC criterion at $\boldsymbol{x}^*$, $\mathrm{argmax}_{l\in\{1,\ldots,L\}} \frac{\Delta\sigma_L^2(l,\boldsymbol{x}^*)}{\sum_{j=1}^{l} C_j}$, where C_j is the simulation cost at level j.

A new point is acquired on `Xcand`. If `Xcand=NULL` and `Xref=NULL`, a new point is acquired on unit hypercube $[0,1]^d$.

For details, see Heo and Sung (2025, <doi:10.1080/00401706.2024.2376173>).

Usage

```
AL_RNAmf(criterion = c("ALM", "ALC", "ALD", "ALMC"), fit,
Xref = NULL, Xcand = NULL, MC = FALSE, mc.sample = 100, cost = NULL,
use_optim = TRUE, parallel = FALSE, ncore = 1, trace = TRUE)
```

Arguments

`criterion`	character string specifying the active learning criterion to use. Must be one of `"ALM"`, `"ALD"`, `"ALC"`, or `"ALMC"`. Default is `"ALM"`.
`fit`	object of class `RNAmf`.
`Xref`	vector or matrix of reference locations to approximate the integral of ALC. If `Xref=NULL`, $100 \times d$ points from 0 to 1 are generated by Latin hypercube design. Only used when `criterion="ALC"` or `"ALMC"`. Default is `NULL`.

Xcand	vector or matrix of the candidate set for grid-based search. If use_optim=FALSE, the criterion is evaluated and optimized only on this set. If Xcand=NULL, $100 \times d$ points from 0 to 1 generated by Latin hypercube design (or Xref for ALC and ALMC) are used. Default is NULL.
MC	logical indicating whether to use Monte Carlo approximation to impute the posterior variance (for ALC/ALMC). If FALSE, posterior means are used. Default is FALSE.
mc.sample	a number of MC samples generated for the imputation through MC approximation. Default is 100.
cost	vector of the costs for each level of fidelity. If cost=NULL, total costs at all levels would be 1. cost is encouraged to have an ascending order of positive values. Default is NULL.
use_optim	logical indicating whether to optimize the criterion using optim's gradient-based L-BFGS-B method. If TRUE, $5 \times d$ starting points are generated by Latin hypercube design for optimization. If FALSE, the point is selected from Xcand. Default is TRUE.
parallel	logical indicating whether to use parallel computation. Default is FALSE.
ncore	integer specifying the number of cores for parallel computation. Used only if parallel=TRUE. Default is 1.
trace	logical indicating whether to print the computational progress and time. Default is TRUE.

Details

For "ALC", or "ALMC", Xref plays a role of $\boldsymbol{\xi}$ to approximate the integration. To impute the posterior variance based on the augmented design $\tilde{\sigma}_L^{*2}(\boldsymbol{\xi}; l, \boldsymbol{x})$, MC approximation is used. Due to the nested assumption, imputing $y_{n_s+1}^{[s]}$ for each $1 \leq s \leq l$ by drawing samples from the posterior distribution of $f_s(\boldsymbol{x}_{n_s+1}^{[s]})$ based on the current design allows to compute $\tilde{\sigma}_L^{*2}(\boldsymbol{\xi}; l, \boldsymbol{x})$. Inverse of covariance matrix is computed by the Sherman-Morrison formula.

To search for the next acquisition $\boldsymbol{x}^*$ by maximizing AL criterion, the gradient-based optimization can be used by optim=TRUE. Firstly, $\tilde{\sigma}_L^{*2}(\boldsymbol{\xi}; l, \boldsymbol{x})$ is computed at $5 \times d$ number of points. After that, the point minimizing $\tilde{\sigma}_L^{*2}(\boldsymbol{\xi}; l, \boldsymbol{x})$ serves as a starting point of optimization by L-BFGS-B method. Otherwise, when optim=FALSE, AL criterion is optimized only on Xcand.

The point is selected by maximizing the ALC criterion: $\text{argmax}_{l \in \{1,\ldots,L\}; \boldsymbol{x} \in \Omega} \frac{\Delta \sigma_L^2(l, \boldsymbol{x})}{\sum_{j=1}^{l} C_j}$.

Value

A list containing:

- AL: The values of the selected criterion at the candidate points (if use_optim=FALSE) or the optimized value (if use_optim=TRUE). For ALMC, it returns the ALC scores for each level at the chosen point.
- cost: A copy of the cost argument.
- Xcand: A copy of the Xcand argument used.
- chosen: A list containing the chosen fidelity level and the new input location Xnext.
- time: The computation time in seconds.

Examples

```
### simulation costs ###
cost <- c(1, 3)

### 1-d Perdikaris function in Perdikaris, et al. (2017) ###
# low-fidelity function
f1 <- function(x) {
  sin(8 * pi * x)
}

# high-fidelity function
f2 <- function(x) {
  (x - sqrt(2)) * (sin(8 * pi * x))^2
}

### training data ###
n1 <- 13
n2 <- 8

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

### generate initial nested design ###
X <- NestedX(c(n1, n2), 1)
X1 <- X[[1]]
X2 <- X[[2]]

y1 <- f1(X1)
y2 <- f2(X2)

### n=100 uniform test data ###
x <- seq(0, 1, length.out = 100)

### fit an RNAmf ###
fit.RNAmf <- RNAmf(list(X1, X2), list(y1, y2), kernel = "sqex", constant=TRUE)

### 1. ALM Criterion ###
alm.RNAmf <- AL_RNAmf(criterion="ALM",
                      Xcand = x, fit=fit.RNAmf, cost = cost,
                      use_optim = FALSE, parallel = TRUE, ncore = 2)
print(alm.RNAmf$chosen)

### visualize ALM ###
oldpar <- par(mfrow = c(1, 2))
plot(x, alm.RNAmf$AL$ALM1,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALM criterion at the low-fidelity level",
     ylim = c(min(c(alm.RNAmf$AL$ALM1, alm.RNAmf$AL$ALM2)),
              max(c(alm.RNAmf$AL$ALM1, alm.RNAmf$AL$ALM2))))
points(alm.RNAmf$chosen$Xnext,
       alm.RNAmf$AL$ALM1[which(x == drop(alm.RNAmf$chosen$Xnext))],
       pch = 16, cex = 1, col = "red")
```

```
plot(x, alm.RNAmf$AL$ALM2,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALM criterion at the high-fidelity level",
     ylim = c(min(c(alm.RNAmf$AL$ALM1, alm.RNAmf$AL$ALM2)),
              max(c(alm.RNAmf$AL$ALM1, alm.RNAmf$AL$ALM2))))
par(oldpar)


### 2. ALD Criterion ###
ald.RNAmf <- AL_RNAmf(criterion="ALD",
                      Xcand = x, fit=fit.RNAmf, cost = cost,
                      use_optim = FALSE, parallel = TRUE, ncore = 2)
print(ald.RNAmf$chosen)

### visualize ALD ###
oldpar <- par(mfrow = c(1, 2))
plot(x, ald.RNAmf$AL$ALD1,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALD criterion at the low-fidelity level",
     ylim = c(min(c(ald.RNAmf$AL$ALD1, ald.RNAmf$AL$ALD2)),
              max(c(ald.RNAmf$AL$ALD1, ald.RNAmf$AL$ALD2))))
points(ald.RNAmf$chosen$Xnext,
       ald.RNAmf$AL$ALD1[which(x == drop(ald.RNAmf$chosen$Xnext))],
       pch = 16, cex = 1, col = "red")
plot(x, ald.RNAmf$AL$ALD2,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALD criterion at the high-fidelity level",
     ylim = c(min(c(ald.RNAmf$AL$ALD1, ald.RNAmf$AL$ALD2)),
              max(c(ald.RNAmf$AL$ALD1, ald.RNAmf$AL$ALD2))))
par(oldpar)


### 3. ALC Criterion ###
alc.RNAmf <- AL_RNAmf(criterion="ALC",
                      Xref = x, Xcand = x, fit=fit.RNAmf, cost = cost,
                      use_optim = FALSE, parallel = TRUE, ncore = 2)
print(alc.RNAmf$chosen)

### visualize ALC ###
oldpar <- par(mfrow = c(1, 2))
plot(x, alc.RNAmf$AL$ALC1,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALC criterion augmented at the low-fidelity level",
     ylim = c(min(c(alc.RNAmf$AL$ALC1, alc.RNAmf$AL$ALC2)),
              max(c(alc.RNAmf$AL$ALC1, alc.RNAmf$AL$ALC2))))
points(alc.RNAmf$chosen$Xnext,
       alc.RNAmf$AL$ALC1[which(x == drop(alc.RNAmf$chosen$Xnext))],
       pch = 16, cex = 1, col = "red")
plot(x, alc.RNAmf$AL$ALC2,
     type = "l", lty = 2,
     xlab = "x", ylab = "ALC criterion augmented at the high-fidelity level",
     ylim = c(min(c(alc.RNAmf$AL$ALC1, alc.RNAmf$AL$ALC2)),
              max(c(alc.RNAmf$AL$ALC1, alc.RNAmf$AL$ALC2))))
```

```
par(oldpar)


### 4. ALMC Criterion ###
almc.RNAmf <- AL_RNAmf(criterion="ALMC",
                       Xref = x, Xcand = x, fit=fit.RNAmf, cost = cost,
                       use_optim = FALSE, parallel = TRUE, ncore = 2)
print(almc.RNAmf$chosen)
```

`closed_form_RNA` *Closed-form prediction for RNAmf model*

Description

The function computes the closed-form posterior mean and variance for the RNAmf model both at the fidelity levels used in model fitting using the chosen kernel.

Usage

```
closed_form_RNA(fits, x, kernel, XX = NULL, pseudo_yy = NULL)
```

Arguments

`fits`	A fitted GP object from `RNAmf`.
`x`	A vector or matrix of new input locations to predict.
`kernel`	A character specifying the kernel type to be used. Choices are `"sqex"`(squared exponential kernel), `"matern1.5"`(Matern kernel with $\nu = 1.5$), or `"matern2.5"`(Matern kernel with $\nu = 2.5$). Default is `"sqex"`.
`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_RNA`.

Value

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

- `mu`: A list of predictive posterior mean at each fidelity level.
- `sig2`: A list of predictive posterior variance at each fidelity level.

imputer_RNA *Imputation step in stochastic EM for the non-nested RNA Model*

Description

The function performs the imputation step of the stochastic EM algorithm for the RNA 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_RNA(XX, yy, kernel=kernel, pred1, fits)
```

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".
pred1	Predictive results for the lowest fidelity level f_1. It should include cov obtained by setting cov.out=TRUE.
fits	A fitted GP object from RNAmf.

Details

The imputer_RNA 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 $Y_l^{*(m-1)}$ for each l, at the m-th iteration of the EM algorithm.

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 $\widetilde{\mathbf{y}}_l$.

NestedX *Constructing nested design sets for the RNA model.*

Description

The function constructs nested design sets with multiple fidelity levels $\mathcal{X}_l \subseteq \cdots \subseteq \mathcal{X}_1$ for use in RNAmf.

Usage

```
NestedX(n, d)
```

Arguments

n	A vector specifying the number of design points at each fidelity level l. Thus, the vector must have a positive value $n_1, \ldots, n_l$ where $n_1 > \cdots > n_l$.
d	A positive integer specifying the dimension of the design.

Details

The procedure replace the points of lower level design $\mathcal{X}_{l-1}$ with the closest points from higher level design $\mathcal{X}_l$. For details, see "NestedDesign".

Value

A list containing the nested design sets at each level, i.e., $\mathcal{X}_1, \ldots, \mathcal{X}_l$.

References

L. Le Gratiet and J. Garnier (2014). Recursive co-kriging model for design of computer experiments with multiple levels of fidelity. *International Journal for Uncertainty Quantification*, 4(5), 365-386; doi:10.1615/Int.J.UncertaintyQuantification.2014006914

Examples

```
### number of design points ###
n1 <- 30
n2 <- 15

### dimension of the design ###
d <- 2

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

### generate the nested design ###
NX <- NestedX(c(n1, n2), d)

### visualize nested design ###
```

```
plot(NX[[1]], col="red", pch=1, xlab="x1", ylab="x2")
points(NX[[2]], col="blue", pch=4)
```

`predict.RNAmf` *prediction of the RNAmf emulator with multiple fidelity levels.*

Description

The function computes the posterior mean and variance of RNA models with multiple fidelity levels by fitted model from `RNAmf`.

Usage

```
## S3 method for class 'RNAmf'
predict(object, x = NULL, nimpute = 50, ...)
```

Arguments

`object`	An object of class `RNAmf` fitted by `RNAmf`.
`x`	A vector or matrix of new input locations for prediction.
`nimpute`	Number of imputations for non-nested designs. Default is 50.
`...`	Additional arguments for compatibility with generic method `predict`.

Details

The `predict.RNAmf` function internally calls `closed_form_RNA` to recursively compute the closed-form posterior mean and variance at each level.

From the fitted model from `RNAmf`, 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. For further details, see Heo and Sung (2025, <doi:10.1080/00401706.2024.2376173>).

Value

- `mu`: A list of vectors containing the predictive posterior mean at each fidelity level.
- `sig2`: A list of vectors containing the predictive posterior variance at each fidelity level.
- `time`: A scalar indicating the computation time.

See Also

`RNAmf` for model fitting.

Examples

```
### two levels example ###

### Perdikaris function ###
f1 <- function(x) {
  sin(8 * pi * x)
}

f2 <- function(x) {
  (x - sqrt(2)) * (sin(8 * pi * x))^2
}

### training data ###
n1 <- 13
n2 <- 8

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

### generate initial nested design ###
X <- NestedX(c(n1, n2), 1)
X1 <- X[[1]]
X2 <- X[[2]]

y1 <- f1(X1)
y2 <- f2(X2)

### n=100 uniform test data ###
x <- seq(0, 1, length.out = 100)

### fit an RNAmf ###
fit.RNAmf <- RNAmf(list(X1, X2), list(y1, y2), kernel = "sqex", constant=TRUE)

### predict ###
predy <- predict(fit.RNAmf, x)$mu[[2]]
predsig2 <- predict(fit.RNAmf, x)$sig2[[2]]

### RMSE ###
print(sqrt(mean((predy - f2(x))^2)))

### visualize the emulation performance ###
plot(x, predy,
  type = "l", lwd = 2, col = 3, # emulator and confidence interval
  ylim = c(-2, 1)
)
lines(x, predy + 1.96 * sqrt(predsig2 * length(y2) / (length(y2) - 2)), col = 3, lty = 2)
lines(x, predy - 1.96 * sqrt(predsig2 * length(y2) / (length(y2) - 2)), col = 3, lty = 2)

curve(f2(x), add = TRUE, col = 1, lwd = 2, lty = 2) # high fidelity function

points(X1, y1, pch = 1, col = "red") # low-fidelity design
points(X2, y2, pch = 4, col = "blue") # high-fidelity design
```

```
### three levels example ###
### Branin function ###
branin <- function(xx, l){
  x1 <- xx[1]
  x2 <- xx[2]
  if(l == 1){
    10*sqrt((-1.275*(1.2*x1+0.4)^2/pi^2+5*(1.2*x1+0.4)/pi+(1.2*x2+0.4)-6)^2 +
   (10-5/(4*pi))*cos((1.2*x1+0.4))+ 10) + 2*(1.2*x1+1.9) - 3*(3*(1.2*x2+2.4)-1) - 1 - 3*x2 + 1
  }else if(l == 2){
    10*sqrt((-1.275*(x1+2)^2/pi^2+5*(x1+2)/pi+(x2+2)-6)^2 +
    (10-5/(4*pi))*cos((x1+2))+ 10) + 2*(x1-0.5) - 3*(3*x2-1) - 1
  }else if(l == 3){
    (-1.275*x1^2/pi^2+5*x1/pi+x2-6)^2 + (10-5/(4*pi))*cos(x1)+ 10
  }
}

output.branin <- function(x, l){
  factor_range <- list("x1" = c(-5, 10), "x2" = c(0, 15))

 for(i in 1:length(factor_range)) x[i] <- factor_range[[i]][1] + x[i] * diff(factor_range[[i]])
  branin(x[1:2], l)
}

### training data ###
n1 <- 20; n2 <- 15; n3 <- 10

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

### generate initial nested design ###
X <- NestedX(c(n1, n2, n3), 2)
X1 <- X[[1]]
X2 <- X[[2]]
X3 <- X[[3]]

y1 <- apply(X1,1,output.branin, l=1)
y2 <- apply(X2,1,output.branin, l=2)
y3 <- apply(X3,1,output.branin, l=3)

### n=10000 grid test data ###
x <- as.matrix(expand.grid(seq(0, 1, length.out = 100),seq(0, 1, length.out = 100)))

### fit an RNAmf ###
fit.RNAmf <- RNAmf(list(X1, X2, X3), list(y1, y2, y3), kernel = "sqex", constant=TRUE)

### predict ###
pred.RNAmf <- predict(fit.RNAmf, x)
predy <- pred.RNAmf$mu[[3]]
predsig2 <- pred.RNAmf$sig2[[3]]

### RMSE ###
print(sqrt(mean((predy - apply(x,1,output.branin, l=3))^2)))
```

```
### visualize the emulation performance ###
x1 <- x2 <- seq(0, 1, length.out = 100)
oldpar <- par(mfrow=c(1,2))
image(x1, x2, matrix(apply(x,1,output.branin, l=3), ncol=100),
zlim=c(0,310), main="Branin function")
image(x1, x2, matrix(predy, ncol=100),
zlim=c(0,310), main="RNAmf prediction")
par(oldpar)

### predictive variance ###
print(predsig2)
```

RNAmf *Fitting the Recursive Non-Additive model with multiple fidelity levels*

Description

The function fits RNA models with designs of multiple fidelity levels. The estimation method is based on MLE. Available kernel choices include the squared exponential kernel, and the Matern kernel with smoothness parameter 1.5 and 2.5. The function returns the fitted model at each level $1, \ldots, l$, the number of fidelity levels l, the kernel choice, whether constant mean or not, and the computation time.

Usage

```
RNAmf(X_list, y_list, kernel = "sqex", constant = TRUE,
init = NULL, n.iter = 50, burn.ratio = 0.75, trace = TRUE, ...)
```

Arguments

X_list	A list of the matrices of input locations for all fidelity levels.
y_list	A list of the vectors or matrices of response values for all fidelity levels.
kernel	A character specifying the kernel type to be used. Choices are "sqex"(squared exponential), "matern1.5", or "matern2.5". Default is "sqex".
constant	A logical indicating for constant mean of GP (constant=TRUE) or zero mean (constant=FALSE). Default is TRUE.
init	A list of the vectors 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.
burn.ratio	Fraction of iterations to discard as burn-in. Default is 0.75.
trace	A logical indicating to print progress of iterations if TRUE, or not if FALSE. Default is TRUE.
...	Additional arguments for compatibility with optim.

Details

Consider the model $\begin{cases} f_1(\boldsymbol{x}) = W_1(\boldsymbol{x}), \\ f_l(\boldsymbol{x}) = W_l(\boldsymbol{x}, f_{l-1}(\boldsymbol{x})) \quad \text{for} \quad l \geq 2, \end{cases}$ where f_l is the simulation code at fidelity level l, and $W_l(\boldsymbol{x}) \sim GP(\alpha_l, \tau_l^2 K_l(\boldsymbol{x}, \boldsymbol{x}'))$ is GP model. Hyperparameters $(\alpha_l, \tau_l^2, \boldsymbol{\theta_l})$ are estimated by maximizing the log-likelihood via an optimization algorithm "L-BFGS-B". For `constant=FALSE`, $\alpha_l = 0$.

Covariance kernel is defined as: $K_l(\boldsymbol{x}, \boldsymbol{x}') = \prod_{j=1}^{d} \phi(x_j, x_j'; \theta_{lj})$ with $\phi(x, x'; \theta) = \exp\left(-\frac{(x-x')^2}{\theta}\right)$ for squared exponential kernel; `kernel="sqex"`, $\phi(x, x'; \theta) = \left(1 + \frac{\sqrt{3}|x-x'|}{\theta}\right) \exp\left(-\frac{\sqrt{3}|x-x'|}{\theta}\right)$ for Matern kernel with the smoothness parameter of 1.5; `kernel="matern1.5"` and $\phi(x, x'; \theta) = \left(1 + \frac{\sqrt{5}|x-x'|}{\theta} + \frac{5(x-x')^2}{3\theta^2}\right) \exp\left(-\frac{\sqrt{5}|x-x'|}{\theta}\right)$ for Matern kernel with the smoothness parameter of 2.5; `kernel="matern2.5"`.

For further details, see Heo and Sung (2025, <doi:10.1080/00401706.2024.2376173>).

Value

A list of class `RNAmf` with:

- `fits`: A list of fitted Gaussian process models $f_l(x)$ at each level $1, \ldots, l$. Each element contains: $\begin{cases} f_1 \text{ for } (X_1, y_1), \\ f_l \text{ for } ((X_l, f_{l-1}(X_l)), y_l), \end{cases}$.
- `level`: The number of fidelity levels l.
- `kernel`: A copy of `kernel`.
- `constant`: A copy of `constant`.
- `nested`: A logical indicating whether the design is nested.
- `time`: A scalar indicating the computation time.

See Also

`predict.RNAmf` for prediction.

Examples

```
### two levels example ###

### Perdikaris function ###
f1 <- function(x) {
  sin(8 * pi * x)
}

f2 <- function(x) {
  (x - sqrt(2)) * (sin(8 * pi * x))^2
}

### training data ###
n1 <- 13
```

```
n2 <- 8

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

### generate initial nested design ###
X <- NestedX(c(n1, n2), 1)
X1 <- X[[1]]
X2 <- X[[2]]

y1 <- f1(X1)
y2 <- f2(X2)

### fit an RNAmf ###
fit.RNAmf <- RNAmf(list(X1, X2), list(y1, y2), kernel = "sqex", constant=TRUE)


### three levels example ###

### Branin function ###
branin <- function(xx, l){
  x1 <- xx[1]
  x2 <- xx[2]
  if(l == 1){
    10*sqrt((-1.275*(1.2*x1+0.4)^2/pi^2+5*(1.2*x1+0.4)/pi+(1.2*x2+0.4)-6)^2 +
              (10-5/(4*pi))*cos((1.2*x1+0.4))+ 10) +
              2*(1.2*x1+1.9) - 3*(3*(1.2*x2+2.4)-1) - 1 - 3*x2 + 1
  }else if(l == 2){
    10*sqrt((-1.275*(x1+2)^2/pi^2+5*(x1+2)/pi+(x2+2)-6)^2 +
              (10-5/(4*pi))*cos((x1+2))+ 10) + 2*(x1-0.5) - 3*(3*x2-1) - 1
  }else if(l == 3){
    (-1.275*x1^2/pi^2+5*x1/pi+x2-6)^2 + (10-5/(4*pi))*cos(x1)+ 10
  }
}

output.branin <- function(x, l){
  factor_range <- list("x1" = c(-5, 10), "x2" = c(0, 15))

 for(i in 1:length(factor_range)) x[i] <- factor_range[[i]][1] + x[i] * diff(factor_range[[i]])
  branin(x[1:2], l)
}

### training data ###
n1 <- 20; n2 <- 15; n3 <- 10

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

### generate initial nested design ###
X <- NestedX(c(n1, n2, n3), 2)
X1 <- X[[1]]
X2 <- X[[2]]
X3 <- X[[3]]
```

```
y1 <- apply(X1,1,output.branin, l=1)
y2 <- apply(X2,1,output.branin, l=2)
y3 <- apply(X3,1,output.branin, l=3)

### fit an RNAmf ###
fit.RNAmf <- RNAmf(list(X1, X2, X3), list(y1, y2, y3), kernel = "sqex", constant=TRUE)
```

Index