

How to use RandomForestsGLS

The package `RandomForestsGLS` fits non-linear regression models on dependent data with Generalised Least Square (GLS) based Random Forest (RF-GLS) detailed in Saha, Basu and Datta (2021) <https://www.tandfonline.com/doi/full/10.1080/01621459.2021.1950003>. We will start by loading the `RandomForestsGLS` R package.

```
library(RandomForestsGLS)
```

Next, we discuss how the `RandomForestsGLS` package can be used for estimation and prediction in a non-linear regression setup under correlated errors in different scenarios.

1. Spatial Data

We consider spatial point referenced data with the following model:

$$y_i = m(\mathbf{x}_i) + w(\mathbf{s}_i) + \epsilon_i;$$

where, y_i , \mathbf{x}_i respectively denotes the observed response and the covariate corresponding to the i^{th} observed location \mathbf{s}_i . $m(\mathbf{x}_i)$ denotes the covariate effect, $w(\mathbf{s})$ accounts for spatial dependence beyond covariates, and ϵ accounts for the independent and identically distributed random Gaussian noise.

In the spatial mixed model setting, the package `RandomForestsGLS` allows for fitting $m(\cdot)$ using RF-GLS. Spatial random effects are modeled using Gaussian Process as is the practice. For model fitting, we use the computationally convenient Nearest Neighbor Gaussian Process (NNGP) (Datta, Banerjee, Finley, and Gelfand (2016)) for the spatial random effects $w(\cdot)$. Along with prediction of the covariate effect (mean function) $m(\cdot)$ we also offer kriging based prediction of spatial responses at new location.

Illustration

We simulate a data from the following model:

$$y_i = 10 \sin(\pi x_i) + w(\mathbf{s}_i) + \epsilon_i; \quad \epsilon \sim N(\mathbf{0}, \tau^2 \mathbf{I}), \tau^2 = 0.1; \quad w \sim \text{exponential GP}; \quad \sigma^2 = 10; \phi = 1.$$

Here, the mean function is $E(Y) = 10 \sin(\pi X)$; w accounts for the spatial correlation, which is generated as a exponential Gaussian process with spatial variance $\sigma^2 = 10$ and spatial correlation decay $\phi = 1$; and ϵ is the i.i.d random noise with variance $\tau^2 = 0.1$, which is also called the nugget in spatial literature.

For illustration purposes, we simulate with $n = 200$:

```
rmvn <- function(n, mu = 0, V = matrix(1)){
  p <- length(mu)
  if(any(is.na(match(dim(V),p))))
    stop("Dimension not right!")
  D <- chol(V)
  t(matrix(rnorm(n*p), ncol=p)%*%D + rep(mu,rep(n,p)))
}

set.seed(5)
n <- 200
```

```

coords <- cbind(runif(n,0,1), runif(n,0,1))
set.seed(2)
x <- as.matrix(runif(n),n,1)
sigma.sq = 10
phi = 1
tau.sq = 0.1
D <- as.matrix(dist(coords))
R <- exp(-phi*D)
w <- rmvn(1, rep(0,n), sigma.sq*R)
y <- rnorm(n, 10*sin(pi * x) + w, sqrt(tau.sq))

```

Model fitting

In the package `RandomForestsGLS`, the working precision matrix used in the GLS-loss are NNGP approximations of precision matrices corresponding to Matérn covariance function.

In order to fit the model, the code requires:

- Coordinates (`coords`): an $n \times 2$ matrix of 2-dimensional locations.
- Response (`y`): an n length vector of response at the observed coordinates.
- Covariates (`X`): an $n \times p$ matrix of the covariates in the observation coordinates.
- Covariates for estimation (`Xtest`): an $n_{test} \times p$ matrix of the covariates where we want to estimate the function. Must have identical variables as that of `X`. Default is `X`.
- Minimum size of leaf nodes (`nthsize`): We recommend not setting this value too small, as that will lead to very deep trees that takes a lot of time to be built and can produce unstable estimates. Default value is 20.
- The parameters corresponding to the covariance function (detailed afterwards).

For the details on choice of other parameters, please refer to the help file of the code `RFGLS_estimate_spatial`, which can be accessed with `?RFGLS_estimate_spatial`.

Known Covariance Parameters

If the covariance parameters are known, we set `param_estimate = FALSE` (default value); the code additionally requires the following:

- Covariance Model (`cov.model`): Supported keywords are: “exponential”, “matern”, “spherical”, and “gaussian” for exponential, Matérn, spherical and Gaussian covariance function respectively. Default value is “exponential”.
- σ^2 (`sigma.sq`): The spatial variance. Default value is 1.
- τ^2 (`tau.sq`): The nugget. Default value is 0.01.
- ϕ (`phi`): The spatial correlation decay parameter. Default value is 5.
- ν (`nu`): The smoothing parameter corresponding to the Matérn covariance function. Default value is 0.5.

We can fit the model as follows:

```

set.seed(1)
est_known <- RFGLS_estimate_spatial(coords, y, x, ntree = 50, cov.model = "exponential",
                                     nthsize = 20, sigma.sq = sigma.sq, tau.sq = tau.sq,
                                     phi = phi)

```

The estimate of the function at the covariates `Xtest` is given in `estimation_reult$predicted`. For interpretation of the rest of the outputs, please see the help file of the code `RFGLS_estimate_spatial`. Using covariance models other than exponential model are in beta testing stage.

Unknown Covariance Parameters

If the covariance parameters are not known we set `param_estimate = TRUE`; the code additionally requires the covariance model (`cov.model`) to be used for parameter estimation prior to RF-GLS fitting. We fit the model with unknown covariance parameters as follows.

```
set.seed(1)
est_unknown <- RFGLS_estimate_spatial(coords, y, x, ntree = 50, cov.model = "exponential",
                                       nthsize = 20, param_estimate = TRUE)
```

Prediction of mean function

Given a fitted model using `RFGLS_estimate_spatial`, we can estimate the mean function at new covariate values as follows:

```
Xtest <- matrix(seq(0,1, by = 1/10000), 10001, 1)
RFGLS_predict_known <- RFGLS_predict(est_known, Xtest)
```

Performance comparison

We obtain the Mean Integrated Squared Error (MISE) of the estimate \hat{m} from RF-GLS on $[0,1]$ and compare it with that corresponding to the classical Random Forest (RF) obtained using package `randomForest` (with similar minimum `nodesize`, `nodesize = 20`, as default `nodesize` performs worse). We see that our method has a significantly smaller MISE. Additionally, we show that the MISE obtained with unknown parameters in RF-GLS is comparable to that of the MISE obtained with known covariance parameters.

```
library(randomForest)
set.seed(1)
RF_est <- randomForest(x, y, nodesize = 20)

RF_predict <- predict(RF_est, Xtest)
#RF MISE
mean((RF_predict - 10*sin(pi * Xtest))^2)
#> [1] 8.36778

#RF-GLS MISE
mean((RFGLS_predict_known$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.150152

RFGLS_predict_unknown <- RFGLS_predict(est_unknown, Xtest)
#RF-GLS unknown MISE
mean((RFGLS_predict_unknown$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.1851928
```

We plot the true $m(x) = 10\sin(\pi x)$ along with the loess-smoothed version of estimated $\hat{m}(\cdot)$ obtained from RF-GLS and RF where we show that RF-GLS estimate approximates $m(x)$ better than that corresponding to RF.

```
rfgls_loess_10 <- loess(RFGLS_predict_known$predicted ~ c(1:length(Xtest)), span=0.1)
rfgls_smoothed10 <- predict(rfgls_loess_10)

rf_loess_10 <- loess(RF_predict ~ c(1:length(RF_predict)), span=0.1)
rf_smoothed10 <- predict(rf_loess_10)

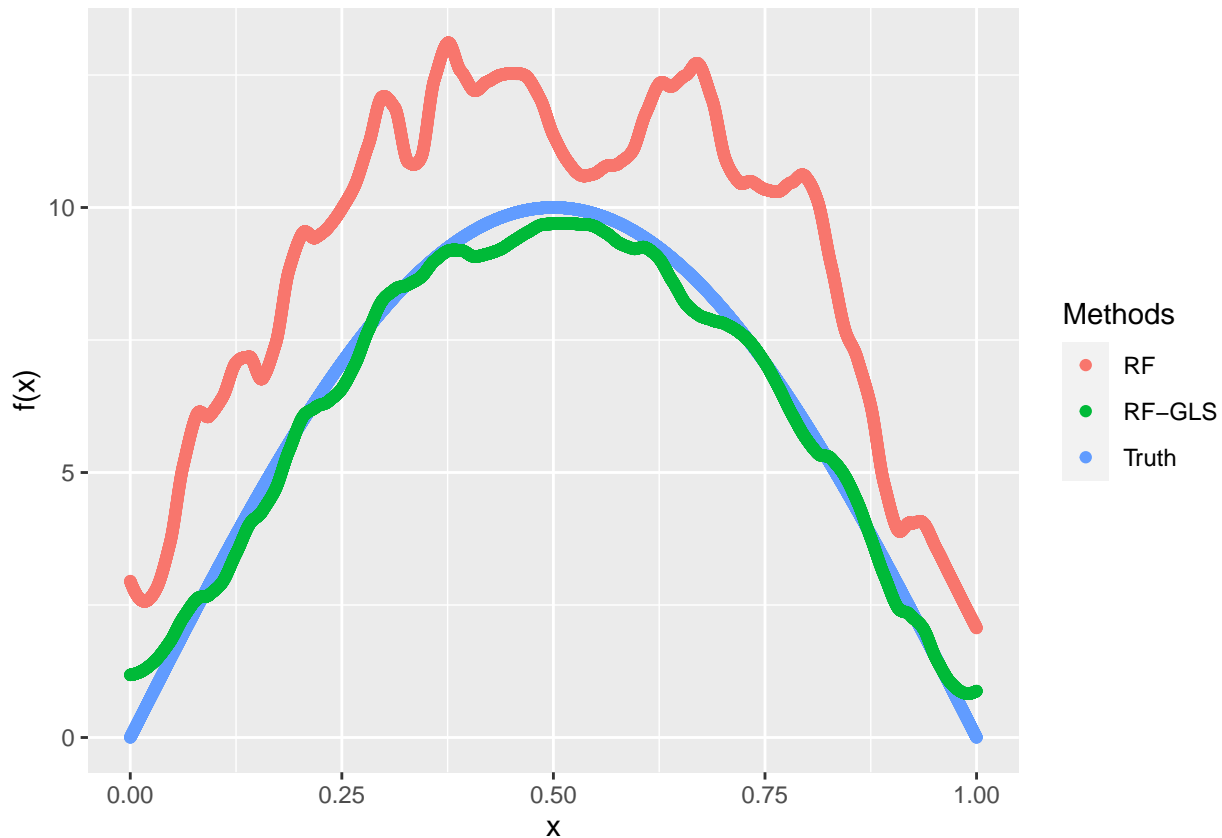
xval <- c(10*sin(pi * Xtest), rf_smoothed10, rfgls_smoothed10)
xval_tag <- c(rep("Truth", length(10*sin(pi * Xtest))), rep("RF", length(rf_smoothed10)),
              rep("RF-GLS", length(rfgls_smoothed10)))
```

```

plot_data <- as.data.frame(xval)
plot_data$Methods <- xval_tag
coval <- c(rep(seq(0,1, by = 1/10000), 3))
plot_data$Covariate <- coval

library(ggplot2)
ggplot(plot_data, aes(x=Covariate, y=xval, color=Methods)) +
  geom_point() + labs( x = "x") + labs( y = "f(x)")

```



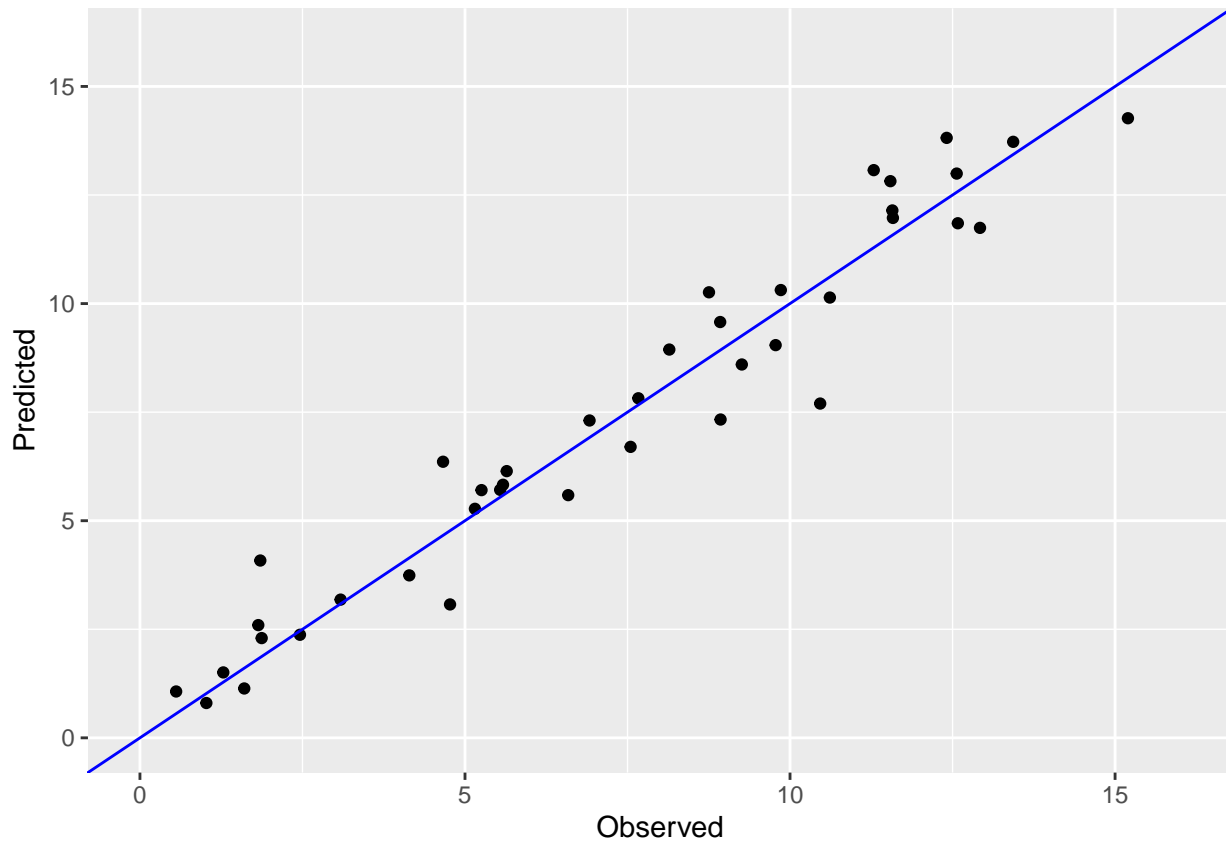
Prediction of spatial response

Given a fitted model using `RFGLS_estimate_spatial`, we can predict the spatial response/outcome at new locations provided the covariates at that location. This approach performs kriging at a new location using the mean function estimates at the corresponding covariate values. Here we partition the simulated data into training and test sets in 4:1 ratio. Next we perform prediction on the test set using a model fitted on the training set.

```

est_known_short <- RFGLS_estimate_spatial(coords[1:160,], y[1:160],
  matrix(x[1:160,],160,1), ntree = 50, cov.model = "exponential",
  nthsize = 20, param_estimate = TRUE)
RFGLS_predict_spatial <- RFGLS_predict_spatial(est_known_short, coords[161:200,],
  matrix(x[161:200,],40,1))
pred_mat <- as.data.frame(cbind(RFGLS_predict_spatial$prediction, y[161:200]))
colnames(pred_mat) <- c("Predicted", "Observed")
ggplot(pred_mat, aes(x=Observed, y=Predicted)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "blue") +
  ylim(0, 16) + xlim(0, 16)

```



Misspecification in covariance model

The following example considers a setting when the parameters are estimated from a misspecified covariance model. We simulate the spatial correlation from a Matérn covariance function with smoothing parameter $\nu = 1.5$. While fitting the RF-GLS, we estimate the covariance parameters using an exponential covariance model ($\nu = 0.5$) and show that the obtained MISE can compare favorably to that of classical RF.

```
#Data simulation from matern with nu = 1.5
nu = 3/2
R1 <- (D*phi)^nu/(2^(nu-1)*gamma(nu))*besselK(x=D*phi, nu=nu)
diag(R1) <- 1
set.seed(2)
w <- rmvn(1, rep(0,n), sigma.sq*R1)
y <- rnorm(n, 10*sin(pi * x) + w, sqrt(tau.sq))

#RF-GLS with exponential covariance
set.seed(3)
est_misspec <- RFGLS_estimate_spatial(coords, y, x, ntree = 50, cov.model = "exponential",
                                     nthsize = 20, param_estimate = TRUE)
RFGLS_predict_misspec <- RFGLS_predict(est_misspec, Xtest)

#RF
set.seed(4)
RF_est <- randomForest(x, y, nodesize = 20)
RF_predict <- predict(RF_est, Xtest)

#RF-GLS MISE
```

```
mean((RFGLS_predict_misspec$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.1380569
#RF MISE
mean((RF_predict - 10*sin(pi * Xtest))^2)
#> [1] 2.295639
```

2. Autoregressive Time Series Data

RF-GLS can also be used for function estimation in a time series setting under autoregressive errors. We consider time series data with errors from an AR(q) process as follows:

$$y_t = m(\mathbf{x}_t) + e_t; e_t = \sum_{i=1}^q \rho_i e_{t-i} + \eta_t$$

where, y_i, \mathbf{x}_i denotes the response and the covariate corresponding to the t^{th} time point, e_t is an AR(q) pprocess, η_t denotes the i.i.d. white noise and (ρ_1, \dots, ρ_q) are the model parameters that captures the dependence of e_t on $(e_{t-1}, \dots, e_{t-q})$.

In the AR time series scenario, the package **RandomForestsGLS** allows for fitting $m(\cdot)$ using RF-GLS. RF-GLS exploits the sparsity of the closed form precision matrix of the AR process for model fitting and prediction of mean function $m(\cdot)$.

Illustration

Here, we simulate from the AR(1) process as follows:

$$y = 10 \sin(\pi x) + \mathbf{e}; e_t = \rho e_{t-1} + \eta_t; \eta_t \sim N(0, \sigma^2); e_1 = \eta_1; \rho = 0.9; \sigma^2 = 10.$$

Here, $E(Y) = 10 \sin(\pi X)$; \mathbf{e} which is an AR(1) process, accounts for the temporal correlation, σ^2 denotes the variance of white noise part of the AR(1) process and ρ captures the degree of dependence of e_t on e_{t-1} .

For illustration purposes, we simulate with $n = 200$:

```
rho <- 0.9
set.seed(1)
b <- rho
s <- sqrt(sigma.sq)
eps = arima.sim(list(order = c(1,0,0), ar = b), n = n, rand.gen = rnorm, sd = s)
y <- c(eps + 10*sin(pi * x))
```

Model fitting

In case of time series data, the code requires:

- Response (**y**): an n length vector of response at the observed time points.
- Covariates (**X**): an $n \times p$ matrix of the covariates in the observation time points.
- Covariates for estimation (**Xtest**): an $n_{test} \times p$ matrix of the covariates where we want to estimate the function. Must have identical variables as that of **X**. Default is **X**.
- Minimum size of leaf nodes (**nthsize**): We recommend not setting this value too small, as that will lead to very deep trees that takes a lot of time to be built and can produce unstable estimates. Default value is 20.
- The parameters corresponding to the AR process (detailed afterwards).

For the details on choice of other parameters, please refer to the help file of the code **RFGLS_estimate_timeseries**, which can be accessed with **?RFGLS_estimate_timeseries**.

Known AR process Parameters

If the AR process parameters are known we set `param_estimate = FALSE` (default value); the code additionally requires `lag_params = c(ρ_1, \dots, ρ_q)`.

We can fit the model as follows:

```
set.seed(1)
est_temp_known <- RFGLS_estimate_timeseries(y, x, ntree = 50, lag_params = rho, nthsize = 20)
```

Unknown AR process Parameters

If the AR process parameters are not known, we set `param_estimate = TRUE`; the code requires the order of the AR process, which is obtained from the length of the `lag_params` input vector. Hence if we want to estimate the parameters from a AR(q) process, `lag_params` should be any vector of length q. Here we fit the model with `q = 1`

```
set.seed(1)
est_temp_unknown <- RFGLS_estimate_timeseries(y, x, ntree = 50, lag_params = rho,
                                              nthsize = 20, param_estimate = TRUE)
```

Prediction of mean function

This part of time series data analysis is identical to that corresponding to the spatial data.

```
Xtest <- matrix(seq(0,1, by = 1/10000), 10001, 1)
RFGLS_predict_temp_known <- RFGLS_predict(est_temp_known, Xtest)
```

Here also, similar to the spatial data scenario, RF-GLS outperforms classical RF in terms of MISE both with true and estimated AR process parameters.

```
library(randomForest)
set.seed(1)
RF_est_temp <- randomForest(x, y, nodesize = 20)

RF_predict_temp <- predict(RF_est_temp, Xtest)
#RF MISE
mean((RF_predict_temp - 10*sin(pi * Xtest))^2)
#> [1] 7.912517

#RF-GLS MISE
mean((RFGLS_predict_temp_known$predicted - 10*sin(pi * Xtest))^2)
#> [1] 2.471876

RFGLS_predict_temp_unknown <- RFGLS_predict(est_temp_unknown, Xtest)
#RF-GLS unknown MISE
mean((RFGLS_predict_temp_unknown$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.8791857
```

Misspecification in AR process order

We consider a scenario where the order of autoregression used for RF-GLS model fitting is mis-specified. We simulate the AR errors from an AR(2) process and fit RF-GLS with an AR(1) process.

```
#Simulation from AR(2) process
rho1 <- 0.7
rho2 <- 0.2
```

```

set.seed(2)
b <- c(rho1, rho2)
s <- sqrt(sigma.sq)
eps = arima.sim(list(order = c(2,0,0), ar = b), n = n, rand.gen = rnorm, sd = s)
y <- c(eps + 10*sin(pi * x))

#RF-GLS with AR(1)
set.seed(3)
est_misspec_temp <- RFGLS_estimate_timeseries(y, x, ntree = 50, lag_params = 0,
                                              nthsize = 20, param_estimate = TRUE)
RFGLS_predict_misspec_temp <- RFGLS_predict(est_misspec_temp, Xtest)

#RF
set.seed(4)
RF_est_temp <- randomForest(x, y, nodesize = 20)
RF_predict_temp <- predict(RF_est_temp, Xtest)

#RF-GLS MISE
mean((RFGLS_predict_misspec_temp$predicted - 10*sin(pi * Xtest))^2)
#> [1] 1.723218
#RF MISE
mean((RF_predict_temp - 10*sin(pi * Xtest))^2)
#> [1] 3.735003

```

Parallelization

For `RFGLS_estimate_spatial`, `RFGLS_estimate_timeseries`, `RFGLS_predict` and `RFGLS_predict_spatial` one can also take the advantage of parallelization, contingent upon the availability of multiple cores. The component `h` in all the functions determines the number of cores to be used. Here we demonstrate an example with `h = 2`.

```

#simulation from exponential distribution
set.seed(5)
n <- 200
coords <- cbind(runif(n,0,1), runif(n,0,1))
set.seed(2)
x <- as.matrix(runif(n),n,1)
sigma.sq = 10
phi = 1
tau.sq = 0.1
nu = 0.5
D <- as.matrix(dist(coords))
R <- exp(-phi*D)
w <- rmvn(1, rep(0,n), sigma.sq*R)
y <- rnorm(n, 10*sin(pi * x) + w, sqrt(tau.sq))

#RF-GLS model fitting and prediction with parallel computation
set.seed(1)
est_known_pl <- RFGLS_estimate_spatial(coords, y, x, ntree = 50, cov.model = "exponential",
                                       nthsize = 20, sigma.sq = sigma.sq, tau.sq = tau.sq,
                                       phi = phi, h = 2)
RFGLS_predict_known_pl <- RFGLS_predict(est_known_pl, Xtest, h = 2)

```



```

#MISE from single core
mean((RFGLS_predict_known$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.150152
#MISE from parallel computation
mean((RFGLS_predict_known_pl$predicted - 10*sin(pi * Xtest))^2)
#> [1] 0.150152

```

For `RFGLS_estimate_spatial` with very small dataset (`n`) and small number of trees (`ntree`), communication overhead between the nodes for parallelization outweighs the benefits of the parallel computing hence it is recommended to parallelize only for moderately large `n` and/or `ntree`. It is strongly recommended that the max value of `h` is kept strictly less than the number of total cores available. Parallelization for `RFGLS_estimate_timeseries` can be addressed identically. For `RFGLS_predict` and `RFGLS_predict_spatial`, even for large dataset, single core performance is very fast, hence unless `ntest` and `ntree` are very high, we do not recommend using parallelization for `RFGLS_predict` and `RFGLS_predict_spatial`.