Package: remotePARTS (via r-universe)

October 21, 2026

Title Spatiotemporal Autoregression Analyses for Large Data Sets

Version 1.1

Description These tools were created to test map-scale hypotheses about trends in large remotely sensed data sets but any data with spatial and temporal variation can be analyzed. Tests are conducted using the PARTS method for analyzing spatially autocorrelated time series (Ives et al., 2021: <doi:10.1016/j.rse.2021.112678>). The method's unique approach can handle extremely large data sets that other spatiotemporal models cannot, while still appropriately accounting for spatial and temporal autocorrelation. This is done by partitioning the data into smaller chunks, analyzing chunks separately and then combining the separate analyses into a single, correlated test of the map-scale hypotheses.

URL https://github.com/morrowcj/remotePARTS

BugReports https://github.com/morrowcj/remotePARTS/issues

License GPL (>= 3)

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.3.2

Depends R (>= 4.0)

Imports stats, geosphere (>= 1.5.10), Rcpp (>= 1.0.5), CompQuadForm, foreach, parallel, iterators, doParallel

Suggests dplyr (>= 1.0.0), data.table, knitr, rmarkdown, markdown, sqldf, devtools, ggplot2, reshape2, sf, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppEigen

Config/testthat/edition 3

Repository https://morrowcj.r-universe.dev

Date/Publication 2025-07-25 06:42:13 UTC

RemoteUrl https://github.com/morrowcj/remoteparts

RemoteRef HEAD

RemoteSha c2391d50e33edc51b0aa515d459d5c1ad03b8dfc

Contents

calc_dfpart . . . 2
check_posdef . . . 3
chisqr . . . 4
chisqr.partGLS . . . 4
covar_taper . . . 5
crosspart_GLS . . . 6
distm_km . . . 8
fitAR . . . 9
fitAR_map . . . 11
fitCLS . . . 13
fitCLS_map . . . 15
fitCor . . . 18
fitGLS . . . 21
fitGLS_opt . . . 24
fitGLS_opt_FUN . . . 27
invert_chol . . . 28
logLik.mapTS . . . 29
logLik.remoteTS . . . 29
max_dist . . . 30
MC_GLSpart . . . 30
ndvi_AK10000 . . . 37
optimize_nugget . . . 38
part_chisqr . . . 39
part_ttest . . . 39
partGLS_ndviAK . . . 40
print.partGLS . . . 40
print.remoteCor . . . 41
print.remoteGLS . . . 41
print.remoteTS . . . 42
remoteGLS . . . 44
sample_partitions . . . 45
t.test.partGLS . . . 46

Index **47**

`calc_dfpart` *calculate degrees of freedom for partitioned GLS*

Description

calculate degrees of freedom for partitioned GLS

Usage

```
calc_dfpart(partsize, p, p0)
```

Arguments

partsize	number of pixels in each partition
p	number of predictors in alternate model
p0	number of parameters in null model

Value

a named vector containing the first and second degrees of freedom ("df1" and "df2", respectively)

---

check_posdef *Check if a matrix is positive definite*

Description

Check if a matrix is positive definite

Usage

```
check_posdef(M)
```

Arguments

M	numeric matrix

Details

check if a matrix is 1) square, 2) symmetric, and 3) positive definite

Value

returns a named logical vector with the following elements:

sqr logical: indicating whether M is square

sym logical: indicating whether M is symmetric

posdef logical: indicating whether M is positive-definitive

Examples

```
# distance matrix
M = distm_scaled(expand.grid(x = 1:3, y = 1:3))

# check if it is positive definitive
check_posdef(M)

# check if the covariance matrix is positive definitive
check_posdef(covar_exp(M, .1))
```

```
# non-symmetric matrix
check_posdef(matrix(1:9, 3, 3))

# non-square matrix
check_posdef(matrix(1:6, 3, 2))
```

`chisqr` *Conduct a chi-squared test*

Description

generic S3 method for a chi-squared test

Usage

```
chisqr(x, ...)
```

Arguments

`x`	object on which to conduct the test
`...`	additional arguments

Value

results of the chi-squared test (generic)

`chisqr.partGLS` *Conduct a chisqr test of "partGLS" object*

Description

Conduct a correlated chi-square test on a partitioned GLS

Usage

```
## S3 method for class 'partGLS'
chisqr(x, ...)
```

Arguments

`x`	"remoteGLS" object
`...`	additional arguments passed to print

Value

a p-value for the correlated chisqr test

`covar_taper` *Tapered-spherical distance-based covariance function*

Description

Tapered-spherical distance-based covariance function

Exponential distance-based covariance function

Exponential-power distance-based covariance function

Usage

```
covar_taper(d, theta, cor = NULL)

covar_exp(d, range)

covar_exppow(d, range, shape)
```

Arguments

`d`	a numeric vector or matrix of distances
`theta`	distance beyond which covariances are forced to 0.
`cor`	optional correlation parameter. If included, the covariance is subtracted from `cor` and `theta` becomes `exp(-theta)` (i.e., a tapered spherical difference covariance function).
`range`	range parameter
`shape`	shape parameter

Details

`covar_taper` calculates covariance v as follows:

if `d <= theta`, then `v = ((1 - (d/theta))^2) * (1 + (d/(2 * theta)))`

if `d > theta`, then `v = 0`

`covar_exp` calculates covariance v as follows:

`v = exp(-d/range)`

`covar_exppow` calculates covariance v as follows:

`v = exp(-(d/range)^2)`

Note that `covar_exppow(..., shape = 1)` is equivalent to `covar_exp()` but is needed as a separate function for use with `fitCor`.

Value

a tapered-spherical transformation of d is returned.

the exponential covariance (v)

exponential-power covariance (v)

Examples

```
# simulate dummy data
map.width = 5 # square map width
coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix

# calculate distance
D = geosphere::distm(coords) # distance matrix

# visualize covariance matrix
image(covar_taper(D, theta = .5*max(D)))

# plot tapered covariance function
curve(covar_taper(x, theta = .5), from = 0, to = 1);abline(v = 0.5, lty = 2, col = "grey80")


# visualize covariance matrix
image(covar_exp(D, range = .2*max(D)))

# plot exponential function with different ranges
curve(covar_exp(x, range = .2), from = 0, to = 1)
curve(covar_exp(x, range = .1), from = 0, to = 1, col = "blue", add = TRUE)
legend("topright", legend = c("range = 0.2", "range = 0.1"), col = c("black", "blue"), lty = 1)


# visualize Exponential covariance matrix
image(covar_exppow(D, range = .2*max(D), shape = 1))

# visualize Exponential-power covariance matrix
image(covar_exppow(D, range = .2*max(D), shape = .5))

# plot exponential power function with different shapes
curve(covar_exppow(x, range = .2, shape = 1), from = 0, to = 1)
curve(covar_exppow(x, range = .2, shape = .5), from = 0, to = 1, col = "blue", add = TRUE)
legend("topright", legend = c("shape = 1.0", "shape = 0.5"), col = c("black", "blue"), lty = 1)
```

`crosspart_GLS` *Calculate cross-partition statistics in a partitioned GLS*

Description

Calculate cross-partition statistics between two GLS partitions

Usage

```
crosspart_GLS(
  xxi,
  xxj,
  xxi0,
```

```
  xxj0,
  invChol_i,
  invChol_j,
  Vsub,
  nug_i,
  nug_j,
  df1,
  df2,
  small = TRUE,
  ncores = NA
)
```

Arguments

xxi	numeric matrix xx from partition i
xxj	numeric matrix xx from partition j
xxi0	numeric matrix xx0 from partition i
xxj0	numeric matrix xx0 from partition j
invChol_i	numeric matrix invcholV from partition i
invChol_j	numeric matrix invcholV from partition j
Vsub	numeric variance matrix for Xij (upper block)
nug_i	nugget from partition i
nug_j	nugget from partition j
df1	first degree of freedom
df2	second degree of freedom
small	logical: if TRUE, only return rcoefij, rSSRij, and rSSEij
ncores	an optional integer indicating how many CPU threads to use for matrix calculations.

Value

crosspart_GLS returns a list of cross-partition statistics.

If small = FALSE, the list contains the following elements

Rij

Hi

Hj

Hi0

Hj0

SiR

SjR

rcoefij

rSSRij

rSSEij

Vcoefij

If `small = FALSE`, the list only contains the necessary elements `rcoefij`, `rSSRij`, and `rSSEij`.

See Also

Other partitionedGLS: `MC_GLSpart()`, `sample_partitions()`

`distm_km` *Calculate a distance matrix from coordinates*

Description

Calculate the distances among points from a single coordinate matrix or

Usage

```
distm_km(coords, coords2 = NULL)

distm_scaled(coords, coords2 = NULL, distm_FUN = "distm_km")
```

Arguments

`coords`	a coordinate matrix with 2 columns and rows corresponding to each location.
`coords2`	an optional coordinate matrix
`distm_FUN`	function used to calculate the distance matrix. This function dictates the units of "max.dist"

Details

`distm_km` is simply a wrapper for `geosphere::distm()`

Value

`distm_km` returns a distance matrix in km

A distance matrix is returned.

If `coords2 = NULL`, then distances among points in `coords` are calculated. Otherwise, distances are calculated between points in `coords` and `coords2`

`distm_km` returns a distance matrix in km and `distm_scaled` returns relative distances (between 0 and 1). The resulting matrix has the attribute "max.dist" which stores the maximum distance of the map. "max.dist" is in km for `distm_km` and in the units of `distm_FUN` for `distm_scaled`.

See Also

`?geosphere::distm()`

Examples

```
map.width = 3 # square map width
coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix
distm_scaled(coords) # calculate relative distance matrix
```

`fitAR` *AR regressions by REML*

Description

`fitAR` is used to fit AR(1) time series regression analysis using restricted maximum likelihood

Usage

```
fitAR(formula, data = NULL)

AR_fun(par, y, X, logLik.only = TRUE)
```

Arguments

`formula`	a model formula, as used by `stats::lm()`
`data`	optional data environment to search for variables in `formula`. As used by `lm()`
`par`	AR parameter value
`y`	vector of time series (response)
`X`	model matrix (predictors)
`logLik.only`	logical: should only the partial log-likelihood be computed

Details

This function finds the restricted maximum likelihood (REML) to estimate parameters for the regression model with AR(1) random error terms

$$y(t) = X(t)\beta + \varepsilon(t)$$
$$\varepsilon(t) = \rho\varepsilon(t-1) + \delta(t)$$

where $y(t)$ is the response at time t;

$X(t)$ is a model matrix containing covariates;

β is a vector of effects of $X(t)$; $\varepsilon(t)$ is the autocorrelated random error;

$\delta \sim N(0, \sigma)$ is a temporally independent Gaussian random variable with mean zero and standard deviation σ;

and ρ is the AR(1) autoregression parameter

`fitAR` estimates the parameter via mathematical optimization of the restricted log-likelihood function.

`AR_fun` is the work horse behind `fitAR` that is called by `optim` to estimate the autoregression parameter ρ.

Value

`fitAR` returns a list object of class "remoteTS", which contains the following elements.

call the function call

coefficients a named vector of coefficients

SE the standard errors of parameter estimates

tstat the t-statistics for coefficients

pval the p-values corresponding to t-tests of coefficients

MSE the model mean squared error

logLik the log-likelihood of the model fit

residuals the residuals: response minus fitted values

fitted.values the fitted mean values

rho The AR parameter, determined via REML

rank the numeric rank of the fitted model

df.residual the residual degrees of freedom

terms the `stats::terms` object used

Output is structured similarly to an "lm" object.

When `logLik.only == F`, `AR_fun` returns the output described in `?fitAR`. When `logLik.only == T`, it returns a quantity that is linearly and negatively related to the restricted log likelihood (i.e., partial log-likelihood).

References

Ives, A. R., K. C. Abbott, and N. L. Ziebarth. 2010. Analysis of ecological

time series with ARMA(p,q) models. Ecology 91:858-871.

See Also

`fitAR_map` to easily apply `fit_AR` to many pixels; `fitCLS` and `fitCLS_map` for conditional least squares time series analyses.

Other remoteTS: `fitAR_map()`, `fitCLS()`, `fitCLS_map()`

Other remoteTS: `fitAR_map()`, `fitCLS()`, `fitCLS_map()`

Examples

```
# simulate dummy data
t = 1:30 # times series
Z = rnorm(30) # random independent variable
x = .2*Z + (.05*t) # generate dependent effects
x[2:30] = x[2:30] + .2*x[1:29] # add autocorrelation

# fit the AR model, using Z as a covariate
(AR = fitAR(x ~ Z))
```

```
# get specific components
AR$residuals
AR$coefficients
AR$pval

# now using time as a covariate
(AR.time <- fitAR(x ~ t))

# source variable from a dataframe
df = data.frame(y = x, t.scaled = t/30, Z = Z)
fitAR(y ~ t.scaled + Z, data = df)

## Methods
summary(AR)
residuals(AR)
coefficients(AR)
```

`fitAR_map` *Map-level AR REML*

Description

`fitAR_map` is used to fit AR REML regression to each spatial location (pixel) within spatiotemporal data.

Usage

```
fitAR_map(
  Y,
  coords,
  formula = "y ~ t",
  X.list = list(t = 1:ncol(Y)),
  resids.only = FALSE
)
```

Arguments

`Y`	a spatiotemporal response variable: a numeric matrix or data frame where columns correspond to time points and rows correspond to pixels.
`coords`	a numeric coordinate matrix or data frame, with two columns and rows corresponding to each pixel
`formula`	a model formula, passed to `fitAR()`: the left side of the formula should always be "y" and the right hand side should refer to variables in `X.list`
`X.list`	a named list of temporal or spatiotemporal predictor variables: elements must be either numeric vectors with one element for each time point or a matrix/data frame with rows corresponding to pixels and columns corresponding to time point. These elements must be named and referred to in `formula`

`resids.only` logical: should output beyond coordinates and residuals be withheld? Useful when passing output to `fitCor()`

Details

`fitAR_map` is a wrapper function that applies `fitAR` to many pixels.

The function loops through the rows of `Y`, matched with rows of spatiotemporal predictor matrices. Purely temporal predictors, given by vectors, are used for all pixels. These predictor variables, given by the right side of `formula` are sourced from named elements in `X.list`.

Value

`fitCLS_map` returns a list object of class "mapTS".

The output will always contain at least these elements:

call the function call

coords the coordinate matrix or dataframe

residuals time series residuals: rows correspond to pixels (`coords`)

When `resids.only = FALSE`, the output will also contain the following components. Matrices have rows that correspond to pixels and columns that correspond to time points and vector elements correspond to pixels.

coefficients a numeric matrix of coefficients

SEs a numeric matrix of coefficient standard errors

tstats a numeric matrix of t-statistics for coefficients

pvals a numeric matrix of p-values for coefficients t-tests

rhos a vector of rho values for each pixel

MSEs a numeric vector of MSEs

logLiks a numeric vector of log-likelihoods

fitted.values a numeric matrix of fitted values

An attribute called "resids.only" is also set to match the value of `resids.only`

See Also

`fitAR` for fitting AR REML to individual time series and `fitCLS` & `fitCLS_map` for time series analysis based on conditional least squares.

Other remoteTS: `fitAR()`, `fitCLS()`, `fitCLS_map()`

Examples

```
# simulate dummy data
 time.points = 9 # time series length
 map.width = 5 # square map width
 coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix
 ## create empty spatiotemporal variables:
 X <- matrix(NA, nrow = nrow(coords), ncol = time.points) # response
 Z <- matrix(NA, nrow = nrow(coords), ncol = time.points) # predictor
# setup first time point:
 Z[, 1] <- .05*coords[,"x"] + .2*coords[,"y"]
 X[, 1] <- .5*Z[, 1] + rnorm(nrow(coords), 0, .05) #x at time t
 ## project through time:
 for(t in 2:time.points){
   Z[, t] <- Z[, t-1] + rnorm(map.width^2)
   X[, t] <- .2*X[, t-1] + .1*Z[, t] + .05*t + rnorm(nrow(coords), 0 , .25)
 }

# visualize dummy data (NOT RUN)
library(ggplot2);library(dplyr)
data.frame(coords, X) %>%
  reshape2::melt(id.vars = c("x", "y")) %>%
  ggplot(aes(x = x, y = y, fill = value)) +
  geom_tile() +
  facet_wrap(~variable)

# fit AR, showing all output
fitAR_map(X, coords, formula = y ~ t, resids.only = TRUE)

# fit AR with temporal and spatiotemporal predictors
(AR.map <- fitAR_map(X, coords, formula = y ~ t + Z, X.list = list(t = 1:ncol(X),
                     Z = Z), resids.only = FALSE))
## extract some values
AR.map$coefficients # coefficients
AR.map$logLik # log-likelihoods

## Methods
summary(AR.map)
residuals(AR.map)
coefficients(AR.map)
```

`fitCLS` *CLS for time series*

Description

`fitCLS` is used to fit conditional least squares regression to time series data.

Usage

```
fitCLS(
  formula,
  data = NULL,
  lag.y = 1,
  lag.x = 1,
  debug = FALSE,
  model = FALSE,
  y = FALSE
)
```

Arguments

formula	a model formula, as used by stats::lm()
data	optional data environment to search for variables in formula. As used by lm()
lag.y	an integer indicating the lag (in time steps) between y and y.0
lag.x	an integer indicating the lag (in time steps) between y and the independent variables (except y.0).
debug	logical debug mode
model	logical, should the used model matrix be returned? As used by lm()
y	logical, should the used response variable be returned? As used by lm()

Details

This function regresses the response variable (y) at time t, conditional on the response at time t-lag.y and the specified dependent variables (X) at time t-lag.x:

$$y(t) = y(t - lag.y) + X(t - lag.x) + \varepsilon$$

where $y(t)$ is the response at time t;

$X(t)$ is a model matrix containing covariates;

β is a vector of effects of $X(t)$;

and $\varepsilon(t)$ is a temporally independent Gaussian random variable with mean zero and standard deviation σ

stats::lm() is then called, using the above equation.

Value

fitCLS returns a list object of class "remoteTS", which inherits from "lm". In addition to the default "lm" components, the output contains these additional list elements:

tstat the t-statistics for coefficients

pval the p-values corresponding to t-tests of coefficients

MSE the model mean squared error

logLik the log-likelihood of the model fit

See Also

fitCLS_map to easily apply fitCLS to many pixels; fitAR and fitAR_map for AR time series analyses.

Other remoteTS: fitAR(), fitAR_map(), fitCLS_map()

Examples

```
# simulate dummy data
t = 1:30 # times series
Z = rnorm(30) # random independent variable
x = .2*Z + (.05*t) # generate dependent effects
x[2:30] = x[2:30] + .2*x[1:29] # add autocorrelation
x = x + rnorm(30, 0, .01)
df = data.frame(x, t, Z) # collect in data frame

# fit a CLS model with previous x, t, and Z as predictors
## note, this model does not follow the underlying process.
### See below for a better fit.
(CLS <- fitCLS(x ~ t + Z, data = df))

# extract other values
CLS$MSE #MSE
CLS$logLik #log-likelihood

# fit with no lag in independent variables (as simulated):
(CLS2 <- fitCLS(x ~ t + Z, df, lag.x = 0))
summary(CLS2)

# no lag in x
fitCLS(x ~ t + Z, df, lag.y = 0)

# visualize the lag
## large lag in x
fitCLS(x ~ t + Z, df, lag.y = 2, lag.x = 0, debug = TRUE)$lag
## large lag in Z
fitCLS(x ~ t + Z, df, lag.y = 0, lag.x = 2, debug = TRUE)$lag

# # throws errors (NOT RUN)
# fitCLS(x ~ t + Z, df, lag.y = 28) # longer lag than time
# fitCLS(cbind(x, rnorm(30)) ~ t + Z, df) # matrix response

## Methods
summary(CLS)
residuals(CLS)
```

---

fitCLS_map *Map-level CLS for time series*

Description

fitCLS_map is used to fit conditional least squares regression to each spatial location (pixel) within spatiotemporal data.

Usage

```
fitCLS_map(
  Y,
  coords,
  formula = "y ~ t",
  X.list = list(t = 1:ncol(Y)),
  lag.y = 1,
  lag.x = 0,
  resids.only = FALSE
)
```

Arguments

Y	a spatiotemporal response variable: a numeric matrix or data frame where columns correspond to time points and rows correspond to pixels.
coords	a numeric coordinate matrix or data frame, with two columns and rows corresponding to each pixel
formula	a model formula, passed to fitCLS(): the left side of the formula should always be "y" and the right hand side should refer to variables in X.list
X.list	a named list of temporal or spatiotemporal predictor variables: elements must be either numeric vectors with one element for each time point or a matrix/data frame with rows corresponding to pixels and columns corresponding to time point. These elements must be named and referred to in formula
lag.y	the lag between y and y.0, passed to fitCLS()
lag.x	the lag between y and predictor variables, passed to fitCLS()
resids.only	logical: should output beyond coordinates and residuals be withheld? Useful when passing output to fitCor()

Details

fitCLS_map is a wrapper function that applies fitCLS() to many pixels.

The function loops through the rows of Y, matched with rows of spatiotemporal predictor matrices. Purely temporal predictors, given by vectors, are used for all pixels. These predictor variables, given by the right side of formula are sourced from named elements in X.list.

Value

fitCLS_map returns a list object of class "mapTS".

The output will always contain at least these elements:

call the function call

coords the coordinate matrix or dataframe

residuals time series residuals: rows correspond to pixels (`coords`)

When `resids.only = FALSE`, the output will also contain the following components. Matrices have rows that correspond to pixels and columns that correspond to time points and vector elements correspond to pixels.

coefficients a numeric matrix of coefficients

SEs a numeric matrix of coefficient standard errors

tstats a numeric matrix of t-statistics for coefficients

pvals a numeric matrix of p-values for coefficients t-tests

MSEs a numeric vector of MSEs

logLiks a numeric vector of log-likelihoods

fitted.values a numeric matrix of fitted values

An attribute called "resids.only" is also set to match the value of `resids.only`

See Also

`fitCLS` for fitting CLS on individual time series and `fitAR` and `fitAR_map` for AR REML time series analysis.

Other remoteTS: `fitAR()`, `fitAR_map()`, `fitCLS()`

Examples

```
# simulate dummy data
time.points = 9 # time series length
map.width = 5 # square map width
coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix
## create empty spatiotemporal variables:
X <- matrix(NA, nrow = nrow(coords), ncol = time.points) # response
Z <- matrix(NA, nrow = nrow(coords), ncol = time.points) # predictor
# setup first time point:
Z[, 1] <- .05*coords[,"x"] + .2*coords[,"y"]
X[, 1] <- .5*Z[, 1] + rnorm(nrow(coords), 0, .05) #x at time t
## project through time:
for(t in 2:time.points){
  Z[, t] <- Z[, t-1] + rnorm(map.width^2)
  X[, t] <- .2*X[, t-1] + .1*Z[, t] + .05*t + rnorm(nrow(coords), 0 , .25)
}

# # visualize dummy data (NOT RUN)
# library(ggplot2);library(dplyr)
# data.frame(coords, X) %>%
#   reshape2::melt(id.vars = c("x", "y")) %>%
#   ggplot(aes(x = x, y = y, fill = value)) +
#   geom_tile() +
#   facet_wrap(~variable)
```

```
# fit CLS, showing all output
fitCLS_map(X, coords, formula = y ~ t, resids.only = TRUE)

# fit CLS with temporal and spatiotemporal predictors
(CLS.map <- fitCLS_map(X, coords, formula = y ~ t + Z,
                       X.list = list(t = 1:ncol(X), Z = Z),
                       resids.only = FALSE))
## extract some values
CLS.map$coefficients # coefficients
CLS.map$logLik # log-likelihoods

## Methods
summary(CLS.map)
residuals(CLS.map)
coefficients(CLS.map)
```

`fitCor` *Estimate spatial parameters from time series residuals*

Description

`fitCor()` estimates parameter values of a distance-based variance function from the pixel-wise correlations among time series residuals.

Usage

```
fitCor(
  resids,
  coords,
  distm_FUN = "distm_scaled",
  covar_FUN = "covar_exp",
  start = list(r = 0.1),
  fit.n = 1000,
  index,
  save_mod = TRUE,
  ...
)
```

Arguments

`resids`	a matrix of time series residuals, with rows corresponding to pixels and columns to time points
`coords`	a numeric coordinate matrix or data frame, with two columns and rows corresponding to each pixel
`distm_FUN`	a function to calculate a distance matrix from `coords`
`covar_FUN`	a function to estimate distance-based covariances

start	a named list of starting parameter values for covar_FUN, passed to nls
fit.n	an integer indicating how many pixels should be used to estimate parameters.
index	an optional index of pixels to use for parameter estimation
save_mod	logical: should the nls model be saved in the output?
...	additional arguments passed to nls.

Details

For accurate results, resids and coords must be paired matrices. Rows of both matrices should correspond to the same pixels.

Distances between sampled pixels are calculated with the function specified by distm_FUN. This function can be any that takes a coordinate matrix as input and returns a distance matrix between points. Some options provided by remotePARTS are distm_km(), which returns distances in kilometers and distm_scaled(), which returns distances scaled between 0 and 1.

covar_FUN can be any function that takes a vector of distances as its first argument, and at least one parameter as additional arguments. remotePARTS provides three suitable functions: covar_exp, covar_exppow, and covar_taper; but user-defined functions are also possible.

Parameters are estimated with stats::nls() by regressing correlations among time series residuals on a function of distances specified by covar_FUN.

start is used by nls to determine how many parameters need estimating, and starting values for those parameters. As such, it is important that start has named elements for each parameter in covar_FUN.

The fit will be performed for all pixels specified in index, if provided. Otherwise, a random sample of length fit.n is used. If fit.n exceeds the number of pixels, all pixels are used. When random pixels are used, parameter estimates will be different for each call of the function. For reproducible results, we recommend taking a random sample of pixels manually and passing in those values as index.

Caution: Note that a distance matrix, of size $n \times n$ must be fit to the sampled data where n is either fit.n or length(index). Take your computer's memory and processing time into consideration when choosing this size.

Parameter estimates are always returned in the same scale of distances calculated by distm_FUN. It is very important that these estimates are re-scaled by users if output of distm_FUN use units different from the desired scale. For example, if the function covar_FUN = function(d, r, a){-(d/r)^a} is used with distm_FUN = "distm_scaled", the estimated range parameter r will be based on a unit-map. Users will likely want to re-scaled it to map units by multiplying r by the maximum distance among points on your map.

If the distm_FUN is on the scale of your map (e.g., "distm_km"), re-scaling is not needed but may be preferable, since it is scaled to the maximum distance among the sampled data rather than the true maximum distance. For example, dividing the range parameter by max.distance and then multiplying it by the true max distance may provide a better range estimate.

Value

fitCor returns a list object of class "remoteCor", which contains these elements:

call the function call

mod the nls fit object, if save_mod=TRUE

spcor a vector of the estimated spatial correlation parameters

max.distance the maximum distance among the sampled pixels, as calculated by dist_FUN.

logLik the log-likelihood of the fit

Examples

```
# simulate dummy data
set.seed(19)
time.points = 30 # time series length
map.width = 8 # square map width
coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix

## create empty spatiotemporal variables:
X <- matrix(NA, nrow = nrow(coords), ncol = time.points) # response
Z <- matrix(NA, nrow = nrow(coords), ncol = time.points) # predictor

## setup first time point:
Z[, 1] <- .05*coords[,"x"] + .2*coords[,"y"]
X[, 1] <- .5*Z[, 1] + rnorm(nrow(coords), 0, .05) #x at time t

## project through time:
for(t in 2:time.points){
  Z[, t] <- Z[, t-1] + rnorm(map.width^2)
  X[, t] <- .2*X[, t-1] + .1*Z[, t] + .05*t + rnorm(nrow(coords), 0 , .25)
}

AR.map = fitAR_map(X, coords, formula = y ~ Z, X.list = list(Z = Z), resids.only = FALSE)

# using pre-defined covariance function
## exponential covariance
fitCor(AR.map$residuals, coords, covar_FUN = "covar_exp", start = list(range = .1))

## exponential-power covariance
fitCor(AR.map$residuals, coords, covar_FUN = "covar_exppow", start = list(range = .1, shape = .2))

# user-specified covariance function
fitCor(AR.map$residuals, coords, covar_FUN = function(d, r){d^r}, start = list(r = .1))

# un-scaled distances:
fitCor(AR.map$residuals, coords, distm_FUN = "distm_km", start = list(r = 106))

# specify which pixels to use, for reproducibility
fitCor(AR.map$residuals, coords, index = 1:64)$spcor #all
fitCor(AR.map$residuals, coords, index = 1:20)$spcor #first 20
fitCor(AR.map$residuals, coords, index = 21:64)$spcor # last 43
# randomly select pixels
fitCor(AR.map$residuals, coords, fit.n = 20)$spcor #random 20
fitCor(AR.map$residuals, coords, fit.n = 20)$spcor # different random 20
```

`fitGLS` *Fit a PARTS GLS model.*

Description

Fit a PARTS GLS model.

Usage

```
fitGLS(
  formula,
  data,
  V,
  nugget = 0,
  formula0 = NULL,
  save.xx = FALSE,
  save.invchol = FALSE,
  logLik.only = FALSE,
  no.F = FALSE,
  coords,
  distm_FUN,
  covar_FUN,
  covar.pars,
  invCholV,
  ncores = NA,
  suppress_compare_warning = FALSE,
  ...
)
```

Arguments

`formula`	a model formula
`data`	an optional data frame environment in which to search for variables given by `formula`
`V`	a covariance matrix, which must be positive definitive. This argument is optional if `coords`, `distm_FUN`, `covar_FUN`, and `covar.pars` are given instead.
`nugget`	an optional numeric nugget, must be positive
`formula0`	an optional formula for the null model to be compared with `formula` by an F-test
`save.xx`	logical: should information needed for cross-partition comparisons be returned?
`save.invchol`	logical: should the inverse of the Cholesky matrix be returned?
`logLik.only`	logical: should calculations stop after calculating partial log-likelihood?
`no.F`	logical: should F-test calculations be made?
`coords`	optional coordinate matrix for calculating `V` internally

distm_FUN	optional function for calculating a distance matrix from coords, when calculating V internally
covar_FUN	optional distance-based covariance function for calculating V internally
covar.pars	an optional named list of parameters passed to covar_FUN when calculating V internally
invCholV	optional pre-calculated inverse cholesky matrix to use in place of V
ncores	an optional integer indicating how many CPU threads to use for matrix calculations.
suppress_compare_warning	an optional variable to suppress warning that arises from identical formula and formula0.
...	additional arguments passed to optimize_nugget, which are only used if if nugget = NA

Details

conduct generalized least-squares regression of spatiotemporal trends

fitGLS fits a GLS model, using terms specified in formula. In the PARTS method, generally the left side of formula should be pixel-level trend estimates and the right side should be spatial predictors. The errors of the GLS are correlated according to covariance matrix V.

If nugget = NA, an ML nugget is estimated from the data using the optimize_nugget() function. Arguments additional arguments (...) are passed to optimize_nugget in this case. V must be provided for nugget optimization.

If formula0 is not specified, the default is to fit an intercept-only null model.

save.xx is included to allow for manually conducting a partitioned GLS analyses. Because most users will not need this feature, opting instead to use fitGLS_partition(), save.xx = FALSE by default.

Similarly, save.invchol is included to allow for recycling of the inverse cholesky matrix. Often, inverting the large cholesky matrix (i.e., invert_chol(V)) is the slowest part of GLS. This argument exists to allow users to recycle this process, though no remotePARTS function currently exists that can use invert_chol(V) to fit the GLS.

logLik.only = TRUE will return only the partial log-likelihood, which can minimized to obtain the maximum likelihood for a given set of data.

If no.F = TRUE, then the model given by formula is not compared to the model given by formula0.

If V is not provided, it can be fit internally by specifying all of coords, distm_FUN, covar_FUN, and covar.pars. The function given by distm_FUN will calculate a distance matrix from coords, which is then transformed into a distance-based covariance matrix with covar_FUN and parameters given by covar.pars.

This function uses C++ code that uses the Eigen matrix library (RcppEigen package) to fit models as efficiently as possible. As such, all available CPU cores are used for matrix calculations on systems with OpenMP support.

ncores is passed to the C++ code Eigen::setNpThreads() which sets the number of cores used for compatible Eigen matrix operations (when OpenMP is used).

Value

`fitGLS` returns a list object of class "remoteGLS", if `logLik.only = FALSE`. The list contains at least the following elements:

coefficients coefficient estimates for predictor variables

SSE sum of squares error

MSE mean squared error

SE standard errors

df_t degrees of freedom for the t-test

logDetV log-determinant of V

tstat t-test statistic

pval_t p-value of the t-statistic

logLik the Log-likelihood of the model

nugget the nugget used in fitting

covar_coef the covariance matrix of the coefficients

If `no.F = FALSE`, the following elements, corresponding to the null model and F-test are also calculated:

coefficients0 coefficient estimates for the null model

SSE0 sum of squares error for the null model

MSE0 mean squared error for the null model

SE0 the standard errors for null coefficients

MSR the regression mean square

df0 the null model F-test degrees of freedom

LL0 the log-likelihood of the null model

df_F the F-test degrees of freedom, for the main model

Fstat the F-statistic

pval_F the F-test p-value

formula the alternate formula used

formula0 the null formula used

An attribute called also set to `"no.F"` is set to the value of argument `no.F`, which signals to generic methods how to handle the output.

If `save.invchol = TRUE`, output also includes

invcholV the inverse of the Cholesky decomposition of the covariance matrix obtained with `invert_chol(V, nugget)`

If `save.xx = TRUE`, output also includes the following elements

xx the predictor variables `X`, from the right side of `formula`, transformed by the inverse cholesky matrix: xx = `invcholV %*% X`

xx0 the predictor variables `X0`, from the right side of `formula0`, transformed by the inverse cholesky matrix: xx0 = `invcholV %*% X0`

The primary use of `xx` and `xx0` is for use with `fitGLS_partition()`.

If `logLik.only = TRUE`, a single numeric output containing the log-likelihood is returned.

Examples

```
## read data
data(ndvi_AK10000)
df = ndvi_AK10000[seq_len(200), ] # first 200 rows

## fit covariance matrix
V = covar_exp(distm_scaled(cbind(df$lng, df$lat)), range = .01)

## run GLS
(GLS = fitGLS(CLS_coef ~ 0 + land, data = df, V = V))

## with F-test calculations to compare with the NULL model
(GLS.F = fitGLS(CLS_coef ~ 0 + land, data = df, V = V, no.F = FALSE))

## find ML nugget
fitGLS(CLS_coef ~ 0 + land, data = df, V = V, no.F = FALSE, nugget = NA)

## calculate V internally
coords = cbind(df$lng, df$lat)
fitGLS(CLS_coef ~ 0 + land, data = df, logLik.only = FALSE, coords = coords,
      distm_FUN = "distm_scaled", covar_FUN = "covar_exp", covar.pars = list(range = .01))

## use inverse cholesky
fitGLS(CLS_coef ~ 0 + land, data = df, invCholV = invert_chol(V))

## save inverse cholesky matrix
invchol = fitGLS(CLS_coef ~ 0 + land, data = df, V = V, save.invchol = TRUE)$invcholV

## re-use inverse cholesky instead of V
fitGLS(CLS_coef ~ 0 + land, data = df, invCholV = invchol)

## Log-likelihood (fast)
fitGLS(CLS_coef ~ 0 + land, data = df, V = V, logLik.only = TRUE)
```

`fitGLS_opt` *Fit a PARTS GLS model, with maximum likelihood spatial parameters*

Description

Fit a PARTS GLS model, with maximum likelihood spatial parameters

Usage

```
fitGLS_opt(
  formula,
  data = NULL,
  coords,
  distm_FUN = "distm_scaled",
  covar_FUN = "covar_exp",
  start = c(range = 0.01, nugget = 0),
  fixed = c(),
  opt.only = FALSE,
  formula0 = NULL,
  save.xx = FALSE,
  save.invchol = FALSE,
  no.F = TRUE,
  trans = list(),
  backtrans = list(),
  debug = TRUE,
  ncores = NA,
  ...
)
```

Arguments

formula	a model formula, passed to fitGLS
data	an optional data frame environment in which to search for variables given by formula; passed to fitGLS
coords	a numeric coordinate matrix or data frame, with two columns and rows corresponding to each pixel
distm_FUN	a function to calculate a distance matrix from coords
covar_FUN	a function to estimate distance-based covariances
start	a named vector of starting values for each parameter to be estimated; names must match the names of arguments in covar_FUN or "nugget"
fixed	an optional named vector of fixed parameter values; names must match the names of arguments in covar_FUN or "nugget"
opt.only	logical: if TRUE, execution will halt after estimating the parameters; a final GLS will not be fit with the estimated parameters
formula0, save.xx, save.invchol, no.F	arguments passed to fitGLS for final GLS output
trans	optional list of functions for transforming the values in start or fixed in order to constrain the parameter space within optim
backtrans	optional list of functions for back-transforming parameters to their correct scale (for use with trans)
debug	logical: debug mode (for use with trans and backtrans)
ncores	an optional integer indicating how many CPU threads to use for calculations.
...	additional arguments passed to stats::optim()

Details

Estimate spatial parameters, via maximum likelihood, from data rather than from time series residuals; Fit a GLS with these specifications.

`fitGLS_opt` fits a GLS by estimating spatial parameters from data. `fitCor`, combined with `fitGLS(nugget = NA)`, gives better estimates of spatial parameters, but time-series residuals may not be available in all cases. In these cases, spatial parameters can be estimated from distances among points and a response vector. Mathematical optimization of the log likelihood of different GLS models are computed by calling `optim()` on `fitGLS`.

Distances are calculated with `distm_FUN` and a covariance matrix is calculated from these distances with `covar_FUN`. Arguments to to `covar_FUN`, except distances, are given by `start` and `fixed`. Parameters specified in `start` will be be estimated while those given by `fixed` will remain constant throughout fitting. Parameter names in `start` and `fixed` should exactly match the names of arguments in `covar_FUN` and should not overlap (though, `fixed` takes precedence).

In addition to arguments of `covar_FUN` a "nugget" component can also be occur in `start` or `fixed`. If "nugget" does not occur in either vector, the GLS are fit with `nugget = 0`. A zero nugget also allows much faster computation, through recycling the common inverse cholesky matrix in each GLS computation. A non-zero nugget requires inversion of a different matrix at each iteration, which can be substantially slower.

If `opt.only = FALSE`, the estimated parameters are used to fit the final maximum likelihood GLS solution with `fitGLS()` and arguments `formula0`, `save.xx`, `save.invchol`, and `no.F`.

Some parameter combinations may not produce valid covariance matrices. During the optimization step messages about non-positive definitive V may result on some iterations. These warnings are produced by `fitGLS` and NA log-likelihoods are returned in those cases.

Note that `fitGLS_opt` fits multiple GLS models, which requires inverting a large matrix for each one (unless a fixed 0 nugget is used). This process is very computationally intensive and may take a long time to finish depending upon your machine and the size of the data.

Sometimes `optim` can have a difficult time finding a reasonable solution and without any constraints on parameter space (with certain algorithms), results may even be nonsensical. To combat this, `fitGLS_opt` has the arguments `trans` and `backtrans` which allow you to transform (and back-transform) parameters to a different scale. For example, you may want to force the 'range' parameter between 0 and 1. The logit function can do just that, as its limits are -Inf and Inf as x approaches 0 and 1, respectively. So, we can set `trans` to the logit function: `trans = list(range = function(x)log(x/(1-x)))`. Then we need to set `backtrans` to the inverse logit function to return a parameter value between 0 and 1: `backtrans = list(range = function(x)1/(1+exp(-x)))`. This will force the optimizer to only search for the range parameter in the space from 0 to 1. Any other constraint function can be used for `trans` provided that there is a matching back-transformation.

Value

If `opt.only = TRUE`, `fitGLS_opt` returns the output from `stats::optim()`: see it's documentation for more details.

Otherwise, a list with two elements is returned:

opt output from `optim`, as above

GLS a "remoteGLS" object. See `fitGLS` for more details.

See Also

fitCor for estimating spatial parameters from time series residuals; fitGLS for fitting GLS and with the option of estimating the maximum-likelihood nugget component only.

Examples

```
## read data
data(ndvi_AK10000)
df = ndvi_AK10000[seq_len(200), ] # first 200 rows

## estimate nugget and range (very slow)
fitGLS_opt(formula = CLS_coef ~ 0 + land, data = df,
            coords = df[, c("lng", "lat")], start = c(range = .1, nugget = 0),
            opt.only = TRUE)

## estimate range only, fixed nugget at 0, and fit full GLS (slow)
fitGLS_opt(formula = CLS_coef ~ 0 + land, data = df,
             coords = df[, c("lng", "lat")],
             start = c(range = .1), fixed = c("nugget" = 0),
             method = "Brent", lower = 0, upper = 1)

## constrain nugget to 0 and 1
logit <- function(p) {log(p / (1 - p))}
inv_logit <- function(l) {1 / (1 + exp(-l))}

fitGLS_opt(formula = CLS_coef ~ 0 + land, data = df,
           coords = df[, c("lng", "lat")],
           start = c(range = .1, nugget = 1e-10),
           trans = list(nugget = logit), backtrans = list(nugget = inv_logit),
           opt.only = TRUE)
```

---

fitGLS_opt_FUN *Function that fitGLS_opt optimizes over*

Description

Function that fitGLS_opt optimizes over

Usage

```
fitGLS_opt_FUN(
  op,
  fp,
  formula,
  data = NULL,
  coords,
  covar_FUN = "covar_exp",
  distm_FUN = "distm_scaled",
```

```
  is.trans = FALSE,
  backtrans = list(),
  ncores = NA
)
```

Arguments

op	a named vector of parameters to be optimized
fp	a named vector of fixed parameters
formula	GLS model formula
data	data source
coords	a coordinate matrix
covar_FUN	a covariance function
distm_FUN	a distm function
is.trans	logical: are any of the values in op or fp transformed, needing back-transformation?
backtrans	optional: a named list of functions used to backtransform any element of op or fp. Names must correspond to names in op or fp.
ncores	an optional integer indicating how many CPU threads to use for calculations.

Value

fitGLS_opt_FUN returns the negative log likelihood of a GLS, given the parameters in op and fp

---

invert_chol *Invert the cholesky decomposition of V*

Description

Invert the cholesky decomposition of V

Usage

```
invert_chol(M, nugget = 0, ncores = NA)
```

Arguments

M	numeric (double), positive definite matrix
nugget	numeric (double) nugget to add to M
ncores	optional integer indicating how many cores to use during the inversion calculation

Details

Calculates the inverse of the Cholesky decomposition of M which should not be confused with the inverse of M *derived* from the Cholesky decomposition (i.e. ‘chol2inv(M)‘).

Value

numeric matrix: inverse of the Cholesky decomposition (lower triangle)

Examples

```
M <- crossprod(matrix(1:6, 3))

# without a nugget:
invert_chol(M)

# with a nugget:
invert_chol(M, nugget = 0.2)
```

---

logLik.mapTS *Log-likelihood of mapTS object*

Description

Log-likelihood of mapTS object

Usage

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

Arguments

object	mapTS object
...	additional arguments to pass to the generic method

---

logLik.remoteTS *Log likelihood of remoteTS objects*

Description

Log likelihood of remoteTS objects

Usage

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

Arguments

object	the remoteTS object
...	additional arguments passed to the generic logLik function

max_dist *calculate maximum distance among a table of coordinates*

Description

calculate maximum distance among a table of coordinates

Usage

```
max_dist(coords, dist_FUN = "distm_km")
```

Arguments

coords	the coordinate matrix (or dataframe) from which a maximum distance is desired.
dist_FUN	the distance function used to calculate distances

Details

First the outermost points are found by fitting a convex hull in Euclidean space. Then, the distances between these outer points is calculated with dist_FUN, and the maximum of these distances is returned

This is a fast, simple way of determining the maximum distance.

Value

The maximum distance between two points (units determined by dist_FUN)

MC_GLSpart *fit a parallel partitioned GLS*

Description

fit a GLS model to a large data set by partitioning the data into smaller pieces (partitions) and processing these pieces individually and summarizing output across partitions to conduct hypothesis tests.

Usage

```
MC_GLSpart(
  formula,
  partmat,
  formula0 = NULL,
  part_FUN = "part_data",
  distm_FUN = "distm_scaled",
  covar_FUN = "covar_exp",
```

```
  covar.pars = c(range = 0.1),
  nugget = NA,
  ncross = 6,
  save.GLS = FALSE,
  ncores = parallel::detectCores(logical = FALSE) - 1,
  debug = FALSE,
  ...
)

MCGLS_partsummary(
  MCpartGLS,
  covar.pars = c(range = 0.1),
  save.GLS = FALSE,
  partsize
)

multicore_fitGLS_partition(
  formula,
  partmat,
  formula0 = NULL,
  part_FUN = "part_data",
  distm_FUN = "distm_scaled",
  covar_FUN = "covar_exp",
  covar.pars = c(range = 0.1),
  nugget = NA,
  ncross = 6,
  save.GLS = FALSE,
  ncores = parallel::detectCores(logical = FALSE) - 1,
  do.t.test = TRUE,
  do.chisqr.test = TRUE,
  debug = FALSE,
  ...
)

fitGLS_partition(
  formula,
  partmat,
  formula0 = NULL,
  part_FUN = "part_data",
  distm_FUN = "distm_scaled",
  covar_FUN = "covar_exp",
  covar.pars = c(range = 0.1),
  nugget = NA,
  ncross = 6,
  save.GLS = FALSE,
  do.t.test = TRUE,
  do.chisqr.test = TRUE,
  progressbar = TRUE,
```

```
  debug = FALSE,
  ncores = NA,
  parallel = TRUE,
  ...
)

part_data(index, formula, data, formula0 = NULL, coord.names = c("lng", "lat"))

part_csv(index, formula, file, formula0 = NULL, coord.names = c("lng", "lat"))
```

Arguments

formula	a formula for the GLS model
partmat	a numeric partition matrix, with values containing indices of locations.
formula0	an optional formula for the null GLS model
part_FUN	a function to partition individual data. See details for more information about requirements for this function.
distm_FUN	a function to calculate distances from a coordinate matrix
covar_FUN	a function to calculate covariances from a distance matrix
covar.pars	a named list of parameters passed to covar_FUN
nugget	a numeric fixed nugget component: if NA, the nugget is estimated for each partition
ncross	an integer indicating the number of partitions used to calculate cross-partition statistics
save.GLS	logical: should full GLS output be saved for each partition?
ncores	an optional integer indicating how many CPU threads to use for calculations.
debug	logical debug mode
...	arguments passed to part_FUN
MCpartGLS	object resulting from MC_partGLS()
partsize	number of locations per partition
do.t.test	logical: should a t-test of the GLS coefficients be conducted?
do.chisqr.test	logical: should a correlated chi-squared test of the model fit be conducted?
progressbar	logical: should progress be tracked with a progress bar?
parallel	logical: should all calculations be done in parallel? See details for more information
index	a vector of pixels with which to subset the data
data	a data frame
coord.names	a vector containing names of spatial coordinate variables (x and y, respectively)
file	a text string indicating the csv file from which to read data

Details

The function specified by part_FUN is called internally to obtain properly formatted subsets of the full data (i.e., partitions). Two functions are provided in the remotePARTs package for this purpose: part_data and part_csv. Both of these functions have required arguments that must be specified through the call to fitGLS_partition (via ...). Check each function's argument list and see "part_FUN details" below for more information.

partmat is used to partition the data. partmat must be a complete matrix, without any missing or non-finite values. Columns of partmat are passed as the first argument part_FUN to obtain data, which is then passed to fitGLS. Users are encouraged to use sample_partitions() to obtain a valid partmat.

The specific dimensions of partmat can have a substantial effect on the efficiency of fitGLS_partition. For most systems, we do not recommend fitting with partitions exceeding 3000 locations or pixels (i.e., partmat(partsize = 3000, ...)). Any larger, and the covariance matrix inversions may become quite slow (or impossible for some machines). It may help performance to use smaller even partitions of around 1000-2000 locations.

ncross determines how many partitions are used to estimate cross-partition statistics. All partitions, up to ncross are compared with all others in a pairwise fashion. There is no hard rule for setting mincross. More crosses will ensure convergence, but we believe that the default of 6 (10 total comparisons) should be sufficient for most moderate-sized maps if 1500-3000 pixel partitions are used. This may require testing with each individual dataset to determine at what point convergence occurs.

Covariance matrices for each partition are calculated with covar_FUN from distances among points within the partition. Parameter values for covar_FUN are given by covar.pars.

The distances among points are calculated with distm_FUN. distm_FUN can be any function, modeled after geosphere::distm(), that satisfies both: 1) returns a distance matrix among points when a single coordinate matrix is given as first argument; and 2) returns a matrix containing distances between two coordinate matrices if given as the first and second arguments.

If nugget = NA, a ML nugget is obtained for each partition. Otherwise, a fixed nugget is used for all partitions.

It is not required to use all partitions for cross-partition calculations, nor is it recommended to do so for most large data sets.

If progressbar = TRUE a text progress bar shows the current status of the calculations in the console.

Value

a "MC_partGLS", which is a precursor to a "partGLS" object

a "partGLS" object

"partGLS" object

fitGLS_partition returns a list object of class "partGLS" which contains at least the following elements:

call the function call

GLS an optional list of "remoteGLS" objects, one for each partition

part statistics calculated from each partition: see below for further details

cross statistics calculated from each pair of crossed partitions, determined by `ncross`: see below for further details

overall summary statistics of the overall model: see below for further details

`part` is a sub-list containing the following elements

coefficients a numeric matrix of GLS coefficients for each partition

SEs a numeric matrix of coefficient standard errors

tstats a numeric matrix of coefficient t-statistics

pvals_t a numeric matrix of t-test pvalues

nuggets a numeric vector of nuggets for each partition

covar.pars `covar.pars` input vector

modstats a numeric matrix with rows corresponding to partitions and columns corresponding to log-likelihoods (`logLik`), sum of square error (`SSE`), mean-squared error (`MSE`), regression mean-square (`MSR`), F-statistics (`Fstat`), and p-values from F-tests (`pval_F`)

`cross` is a sub-list containing the following elements, which are use to calculate the combined (across partitions) standard errors of the coefficient estimates and statistical tests. See Ives et al. (2022).

rcoefs a numeric matrix of cross-partition correlations in the estimates of the coefficients

rSSRs a numeric vector of cross-partition correlations in the regression sum of squares

rSSEs a numeric vector of cross-partition correlations in the sum of squared errors

and `overall` is a sub-list containing the elements

coefficients a numeric vector of the average coefficient estimates across all partitions

rcoefficients a numeric vector of the average cross-partition coefficient from across all crosses

rSSR the average cross-partition correlation in the regression sum of squares

rSSE the average cross-partition correlation in the sum of squared errors

Fstat the average f-statistic across partitions

dfs degrees of freedom to be used with partitioned GLS f-test

partdims dimensions of `partmat`

pval.chisqr if `chisqr.test = TRUE`, a p-value for the correlated chi-squared test

t.test if `do.t.test = TRUE`, a table with t-test results, including the coefficient estimates, standard errors, t-statistics, and p-values

`part_data` and `part_csv` both return a list with two elements:

data a dataframe, containing the data subset

coords a coordinate matrix for the subset

parallel implementation

In order to be efficient and account for different user situations, parallel processing is available natively in `fitGLS_partition`. There are a few different specifications that will result in different behavior:

When `parallel = TRUE` and `ncores > 1`, all calculations are done completely in parallel (via `multicore_fitGLS_partition`). In this case, parallelization is implemented with the `parallel`, `doParallel`, and `foreach` packages. In this version, all matrix operations are serialized on each worker but multiple operations can occur simultaneously..

When `parallel = FALSE` and `ncores > 1`, then most calculations are done on a single core but matrix operations use multiple cores. In this case, `ncores` is passed to fitGLS. In this option, it is suggested to not exceed the number of physical cores (not threads).

When `ncores <= 1`, then the calculations are completely serialized

When `ncores = NA` (the default), only matrix multiplication is done in parallel and the number of cores determined by defaults used by `Eigen`. Typically, if `OpenMP` is available, all cores are used, otherwise only 1 is used. In this case, `parallel` is ignored (and treated as FALSE).

In the parallel implementation of this function, a progress bar is not possible, so `progressbar` is ignored.

`part_FUN` details

`part_FUN` can be any function that satisfies the following criteria

1. the first argument of `part_FUN` must accept an index of pixels by which to subset the data;

2. `part_FUN` must also accept `formula` and `formula0` from `fitGLS_partition`; and

3. the output of `part_FUN` must be a list with at least the following elements, which are passed to `fitGLS`;

data a data frame containing all variables given by `formula`. Rows should correspond to pixels specified by the first argument

coords a coordinate matrix or data frame. Rows should correspond to pixels specified by the first argument

Two functions that satisfy these criteria are provided by `remotePARTS`: `part_data` and `part_csv`.

`part_data` uses an in-memory data frame (`data`) as a data source. `part_csv`, instead reads data from a csv file (`file`), one partition at a time, for efficient memory usage. `part_csv` internally calls `sqldf::read.csv.sql()` for fast and efficient row extraction.

Both functions use `index` to subset rows of data and `formula` and `formula0` (optional) to determine which variables to select.

Both functions also use `coord.names` to indicate which variables contain spatial coordinates. The name of the x-coordinate column should always precede the y-coordinate column: `c("x", "y")`.

Users are encouraged to write their own `part_FUN` functions to meet their needs. For example, one might be interested in using data stored in a raster stack or any other file type. In this case, a user-defined `part_FUN` function allows access to `fitGLS_partition` without saving reformatted copies of data.

References

Ives, A. R., L. Zhu, F. Wang, J. Zhu, C. J. Morrow, and V. C. Radeloff. in review. Statistical tests for non-independent partitions of large autocorrelated datasets. MethodsX.

See Also

Other partitionedGLS: crosspart_GLS(), sample_partitions()

Other partitionedGLS: crosspart_GLS(), sample_partitions()

Other partitionedGLS: crosspart_GLS(), sample_partitions()

Examples

```
## read data
data(ndvi_AK10000)
df = ndvi_AK10000[seq_len(1000), ] # first 1000 rows

## create partition matrix
pm = sample_partitions(nrow(df), npart = 3)

## fit GLS with fixed nugget
partGLS = fitGLS_partition(formula = CLS_coef ~ 0 + land, partmat = pm,
                           data = df, nugget = 0, do.t.test = TRUE)

## hypothesis tests
chisqr(partGLS) # explanatory power of model
t.test(partGLS) # significance of predictors

## now with a numeric predictor
fitGLS_partition(formula = CLS_coef ~ lat, partmat = pm, data = df, nugget = 0)


## fit ML nugget for each partition (slow)
(partGLS.opt = fitGLS_partition(formula = CLS_coef ~ 0 + land, partmat = pm,
                                data = df, nugget = NA))
partGLS.opt$part$nuggets # ML nuggets

## Explicitly use multicore_fitGLS_partition()
(multicore_fitGLS_partition(formula = CLS_coef ~ 0 + land, partmat = pm,
                           data = df, nugget = 0, ncores = 2L))
(multicore_fitGLS_partition(formula = CLS_coef ~ 1, partmat = pm, ncores = 2L,
                            data = df, nugget = 0, do.chisqr.test = FALSE))

## fully parallel, using 2 cores
(MC_GLSpart = fitGLS_partition(formula = CLS_coef ~ 0 + land, partmat = pm, data = df, nugget = 0,
                 ncores = 2, parallel = TRUE, debug = FALSE))
fitGLS_partition(formula = CLS_coef ~ lat, partmat = pm, data = df, nugget = 0,
                 ncores = 2, parallel = TRUE, debug = FALSE)
fitGLS_partition(formula = CLS_coef ~ 1, partmat = pm, data = df, nugget = 0,
                 parallel = TRUE, ncores = 2, do.chisqr.test = FALSE)

# Certain model structures may not be useful:
```

```
## 0 intercept with numeric predictor (produces NAs) and gives a warning in statistical tests
fitGLS_partition(formula = CLS_coef ~ 0 + lat, partmat = pm, data = df, nugget = 0)

## intercept-only, gives warning
fitGLS_partition(formula = CLS_coef ~ 1, partmat = pm, data = df, nugget = 0,
                 do.chisqr.test = FALSE)

## part_data examples
part_data(1:20, CLS_coef ~ 0 + land, data = ndvi_AK10000)


## part_csv examples - ## CAUTION: examples for part_csv() include manipulation side-effects:
# first, create a .csv file from ndviAK
data(ndvi_AK10000)
file.path = "ndviAK10000-remotePARTS.csv"
write.csv(ndvi_AK10000, file = file.path)

# build a partition from the first 30 pixels in the file
part_csv(1:20, formula = CLS_coef ~ 0 + land, file = file.path)

# now with a random 20 pixels
part_csv(sample(3000, 20), formula = CLS_coef ~ 0 + land, file = file.path)

# remove the example csv file from disk
file.remove(file.path)
```

`ndvi_AK10000` *NDVI remote sensing data for 10,000 random pixels from Alaska, with rare land classes removed.*

Description

NDVI remote sensing data for 10,000 random pixels from Alaska, with rare land classes removed.

Usage

```
ndvi_AK10000
```

Format

data frame with 10,000 rows corresponding to sites and 37 columns:

lng longitude of the pixel

lat latitude of the pixel

AR_coef pre-calculated AR REML coefficient standardized by mean ndvi values for each pixel

CLS_coef pre-calculated CLS coefficient standardized by mean ndvi values for each pixel

land dominant land class of the pixel

land logical: is this land class rare?

ndvi<t> ndvi value of the pixel during the year <t>

`optimize_nugget` *Find the maximum likelihood estimate of the nugget*

Description

Find the maximum likelihood estimate of the nugget

Usage

```
optimize_nugget(
  X,
  y,
  V,
  lower = 0.001,
  upper = 0.999,
  tol = .Machine$double.eps^0.25,
  debug = FALSE,
  ncores = NA
)
```

Arguments

`X`	numeric (double) nxp matrix
`y`	numeric (double) nx1 column vector
`V`	numeric (double) nxn matrix
`lower`	lower boundary for nugget search
`upper`	upper boundary for nugget search
`tol`	desired accuracy of nugget search
`debug`	logical: debug mode?
`ncores`	an optional integer indicating how many CPU threads to use for matrix calculations.

Details

Finds the maximum likelihood nugget estimate via mathematical optimization.

To maximize efficiency, `optimize_nugget()` is implemented entirely in C++. Optimization takes place via a C++ version of the `fmin` routine (Forsythe et al 1977). Translated from http://www.netlib.org/fmm/fmin.f

The function `LogLikGLS()` is optimized for `nugget`. Once the `LogLikGLS()` functionality is absorbed by `fitGLS()`, it will be used instead.

Value

maximum likelihood nugget estimate

See Also

```
?stats::optimize()
```

`part_chisqr` *Chisqr test for partitioned GLS*

Description

Chisqr test for partitioned GLS

Usage

```
part_chisqr(Fmean, rSSR, df1, npart)
```

Arguments

`Fmean`	mean value of F-statistic from correlated F-tests
`rSSR`	correlation among partition regression sum of squares
`df1`	first degree of freedom for F-tests
`npart`	number of partitions

Value

a p-value for the correlated chisqr test

`part_ttest` *Correlated t-test for partitioned GLS*

Description

Correlated t-test for partitioned GLS

Usage

```
part_ttest(coefs, part.covar_coef, rcoefficients, df2, npart)
```

Arguments

coefs	vector average GLS coefficients
part.covar_coef	an array of covar_coef from each partition
rcoefficients	an rcoefficients array, one for each partition
df2	second degree of freedom from partitioned GLS
npart	number of partitions

Value

a list whose first element is a coefficient table with estimates, standard errors, t-statistics, and p-values and whose second element is a matrix of correlations among coefficients.

---

partGLS_ndviAK *partitioned GLS results*

Description

Example output from fitGLS_partition() fit to the ndvi_AK data set

Usage

```
partGLS_ndviAK
```

Format

an S3 class "partGLS" object. See ?fitGLS_partition() for further details

---

print.partGLS *S3 print method for "partGLS" objects*

Description

S3 print method for "partGLS" objects

Usage

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

Arguments

x	"partGLS" object
...	additional arguments passed to print

Value

a print-formatted version of key elements of the "partGLS" object.

`print.remoteCor` *S3 print method for "remoteCor" class*

Description

S3 print method for "remoteCor" class

Usage

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

Arguments

`x`	remoteCor object to print
`...`	additional arguments passed to print()

Value

a print-formatted version of key elements of the "remoteCor" object.

`print.remoteGLS` *print method for remoteGLS*

Description

print method for remoteGLS

Usage

```
## S3 method for class 'remoteGLS'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`x`	remoteGLS object
`digits`	digits to print
`...`	additional arguments

Value

formatted output for remoteGLS object

`print.remoteTS` *S3 print method for remoteTS class*

Description

S3 print method for remoteTS class

S3 summary method for remoteTS class

S3 print method for mapTS class

S3 summary method for mapTS class

helper summary function (matrix)

helper summary function (vector)

Usage

```
## S3 method for class 'remoteTS'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

## S3 method for class 'remoteTS'
summary(
  object,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

## S3 method for class 'mapTS'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'mapTS'
summary(
  object,
  digits = max(3L, getOption("digits") - 3L),
  CL = 0.95,
  na.rm = TRUE,
  ...
)

smry_funM(x, CL = 0.95, na.rm = TRUE)

smry_funV(x, CL = 0.95, na.rm = TRUE)
```

Arguments

x	numeric matrix
digits	significant digits to show
signif.stars	logical, passed to stats::printCoefmat
...	additional parameters passed to further print methods
object	mapTS object
CL	confidence level (default = .95)
na.rm	logical, should observations with NA be removed?

Value

returns formatted output

returns formatted output, including summary stats

returns formatted output

returns formatted summary stats

summary statistics for each column including quartiles, mean, and upper and lower confidence levels (given by CL)

summary statistics including quartiles, mean, and upper and lower confidence levels (given by CL)

Examples

```
# simulate dummy data
 time.points = 9 # time series length
 map.width = 5 # square map width
 coords = expand.grid(x = 1:map.width, y = 1:map.width) # coordinate matrix
 ## create empty spatiotemporal variables:
 X <- matrix(NA, nrow = nrow(coords), ncol = time.points) # response
 Z <- matrix(NA, nrow = nrow(coords), ncol = time.points) # predictor
 # setup first time point:
 Z[, 1] <- .05*coords[,"x"] + .2*coords[,"y"]
 X[, 1] <- .5*Z[, 1] + rnorm(nrow(coords), 0, .05) #x at time t
 ## project through time:
 for(t in 2:time.points){
   Z[, t] <- Z[, t-1] + rnorm(map.width^2)
   X[, t] <- .2*X[, t-1] + .1*Z[, t] + .05*t + rnorm(nrow(coords), 0 , .25)
 }

 ## Pixel CLS
 tmp.df = data.frame(x = X[1, ], t = nrow(X), z = Z[1, ])
 CLS <- fitCLS(x ~ z, data = tmp.df)
 print(CLS)
 summary(CLS)
 residuals(CLS)
 coef(CLS)
 logLik(CLS)
 fitted(CLS)
 # plot(CLS) # doesn't work
```

```
## Pixel AR
AR <- fitAR(x ~ z, data = tmp.df)
print(AR)
summary(AR)
coef(AR)
residuals(AR)
logLik(AR)
fitted(AR)
# plot(AR) # doesn't work

## Map CLS
CLS.map <- fitCLS_map(X, coords, y ~ Z, X.list = list(Z = Z), lag.x = 0, resids.only = TRUE)
print(CLS.map)
summary(CLS.map)
residuals(CLS.map)
# plot(CLS.map)# doesn't work

CLS.map <- fitCLS_map(X, coords, y ~ Z, X.list = list(Z = Z), lag.x = 0, resids.only = FALSE)
print(CLS.map)
summary(CLS.map)
coef(CLS.map)
residuals(CLS.map)
# logLik(CLS.map) # doesn't work
fitted(CLS.map)
# plot(CLS.map) # doesn't work

## Map AR
AR.map <- fitAR_map(X, coords, y ~ Z, X.list = list(Z = Z), resids.only = TRUE)
print(AR.map)
summary(AR.map)
residuals(AR.map)
# plot(AR.map)# doesn't work

AR.map <- fitAR_map(X, coords, y ~ Z, X.list = list(Z = Z), resids.only = FALSE)
print(AR.map)
summary(AR.map)
coef(AR.map)
residuals(AR.map)
# logLik(AR.map) # doesn't work
fitted(AR.map)
# plot(AR.map) # doesn't work
```

`remoteGLS` *remoteGLS constructor (S3)*

Description

remoteGLS constructor (S3)

Usage

```
remoteGLS(formula, formula0, no.F = FALSE)
```

Arguments

formula	optional argument specifying the GLS formula
formula0	optional argument specifying the null GLS formula
no.F	optional argument specifying the no.F attribute

Value

an empty S3 object of class "remoteGLS"

---

sample_partitions *Randomly sample a partition matrix for partitioned GLS*

Description

Create a matrix whose columns contain indices of non-overlapping random samples.

Usage

```
sample_partitions(
  npix,
  npart = 10,
  partsize = NA,
  pixels = NA,
  verbose = FALSE
)
```

Arguments

npix	number of pixels in full dataset
npart	number of partitions to create
partsize	size of each partition
pixels	vector of pixel indexes to sample from
verbose	logical: TRUE prints additional info

Details

If both npart and partsize is specified, a partition matrix with these dimensions is returned. If only npart, is specified, partsize is selected as the largest integer possible that creates equal sized partitions. Similarly, if only npart = NA, then npart is selected to obtain as many partitions as possible.

Value

`sample_partitions` returns a matrix with `partsize` rows and `npart` columns. Columns contain random, non-overlapping samples from `1:npix`

See Also

Other partitionedGLS: `MC_GLSpart()`, `crosspart_GLS()`

Examples

```
# dummy data with 100 pixels and 20 time points
dat.M <- matrix(rnorm(100*20), ncol = 20)

# 4 partitions (exhaustive)
sample_partitions(npix = nrow(dat.M), npart = 4)

# partitions with 10 pixels each (exhaustive)
sample_partitions(npix = nrow(dat.M), partsize = 10)

# 4 partitions each with 10 pixels (non-exhaustive, produces warning)
sample_partitions(npix = nrow(dat.M), npart = 4, partsize = 10)

# index of 50 pixels to use as subset
sub.indx <- c(1:10, 21:25, 30:62, 70:71)

# 5 partitions (exhaustive) from only the specified pixel subset
sample_partitions(npix = nrow(dat.M), npart = 5, pixels = sub.indx)
```

---

t.test.partGLS *Conduct a t-test of "partGLS" object*

Description

Conduct a correlated t-test of a partitioned GLS

Usage

```
## S3 method for class 'partGLS'
t.test(x, ...)
```

Arguments

x	"partGLS" object
...	additional arguments passed to print

Value

a list whose first element is a coefficient table with estimates, standard errors, t-statistics, and p-values and whose second element is a matrix of correlations among coefficients.

Index

∗ **datasets**
ndvi_AK10000, 37
partGLS_ndviAK, 40
∗ **partitionedGLS**
crosspart_GLS, 6
MC_GLSpart, 30
sample_partitions, 45
∗ **remoteTS**
fitAR, 9
fitAR_map, 11
fitCLS, 13
fitCLS_map, 15

AR_fun *(*fitAR*)*, 9

calc_dfpart, 2
check_posdef, 3
chisqr, 4
chisqr.partGLS, 4
covar_exp *(*covar_taper*)*, 5
covar_exppow *(*covar_taper*)*, 5
covar_taper, 5
crosspart_GLS, 6, *36*, *46*

distm_km, 8
distm_scaled *(*distm_km*)*, 8

fitAR, 9, *12*, *15*, *17*
fitAR_map, *10*, 11, *15*, *17*
fitCLS, *10*, *12*, 13, *17*
fitCLS_map, *10*, *12*, 15, *15*
fitCor, 18, *26, 27*
fitGLS, 21, *26, 27*
fitGLS_opt, 24
fitGLS_opt_FUN, 27
fitGLS_partition *(*MC_GLSpart*)*, 30

invert_chol, 28

logLik.mapTS, 29
logLik.remoteTS, 29

max_dist, 30
MC_GLSpart, *8*, 30, *46*
MCGLS_partsummary *(*MC_GLSpart*)*, 30
multicore_fitGLS_partition *(*MC_GLSpart*)*, 30

ndvi_AK10000, 37

optimize_nugget, 38

part_chisqr, 39
part_csv *(*MC_GLSpart*)*, 30
part_data *(*MC_GLSpart*)*, 30
part_ttest, 39
partGLS_ndviAK, 40
print.mapTS *(*print.remoteTS*)*, 42
print.partGLS, 40
print.remoteCor, 41
print.remoteGLS, 41
print.remoteTS, 42

remoteGLS, 44

sample_partitions, *8*, *36*, 45
smry_funM *(*print.remoteTS*)*, 42
smry_funV *(*print.remoteTS*)*, 42
summary.mapTS *(*print.remoteTS*)*, 42
summary.remoteTS *(*print.remoteTS*)*, 42

t.test.partGLS, 46