Package 'widals'

October 12, 2022

Type Package

Title Weighting by Inverse Distance with Adaptive Least Squares

Version 0.6.1

Date 2019-12-07

Author Dave Zes

Maintainer Dave Zes <zesdave@gmail.com>

Description
Computationally easy modeling, interpolation, forecasting of massive temporal-spacial data.

License GPL (>= 2)

Depends snowfall

Suggests SSsimple (>= 0.6.6)

Imports methods

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-12-07 22:20:02 UTC

R topics documented:

`widals-package` *Weighting by Inverse Distance with Adaptive Least Squares for Massive Space-Time Data*

Description

Fit, forecast, predict massive spacio-temporal data

Details

Package:	widals
Type:	Package
Version:	0.6.1
Date:	2019-12-07
License:	GPL (>=2)

The two essential functions are `widals.snow` and `widals.predict`, both contain an Adaptive Least Squares (ALS) prediction stage and complementary 'stochastic adjustment' stage. The function `H.als.b` solely fits with ALS.

This package offers the user a metaheuristic stochastic search to locate the scalar WIDALS hyperparameters. The function `MSS.snow` along with helper functions `fun.load` serve this end. In fairness, providing some useful amount of generality makes this aspect of `widals` a bit challenging to learn. The user new to this package should expect to spend a couple hours playing with the examples before effectively applying these functions to their own data.

Author(s)

Dave Zes

Maintainer: <zesdave@gmail.com>

See Also

Package `LatticeKrig`.

`applystnd.Hs` *Standardize Spacial Covariates with Existing Object*

Description

Standardize spacial covariates with respect to both the space and time dimensions

Usage

```
applystnd.Hs(Hs0, x)
```

Arguments

`Hs0`	Spacial covariates (of interpolation sites). An $n*$ x p_s numeric matrix.
`x`	Spacial standardization object, as created by `stnd.Hs`.

Value

An $n*$ x p_s matrix.

See Also

`stnd.Hst.ls`, `applystnd.Hst.ls`.

Examples

```
n.all <- 21
Hs.all <- cbind(1, rnorm(n.all, 1, 0.1), rnorm(n.all, -200, 21))

ndx.interp <- c(1,3,5)
ndx.support <- I(1:n.all)[ -ndx.interp ]


Hs <- Hs.all[ndx.support, , drop=FALSE]

xsns.obj <- stnd.Hs(Hs)

Hs0 <- Hs.all[ndx.interp, , drop=FALSE]
```

```
sHs0 <- applystnd.Hs(Hs0, xsns.obj)
sHs0

xsns.obj$sHs

crossprod(xsns.obj$sHs) / nrow(Hs)

crossprod(sHs0) / nrow(sHs0)


## The function is currently defined as
function (Hs0, x)
{
    sHs0 <- t((t(Hs0) - x$h.mean)/x$h.sd)
    if (x$intercept) {
        sHs0[, 1] <- 1/sqrt(x$n)
    }
    return(sHs0)
  }
```

applystnd.Hst.ls *Standardize Space-Time Covariates with Existing Object*

Description

Standardize spacio-temporal covariates with respect to both the space and time dimensions

Usage

```
applystnd.Hst.ls(Hst0.ls, x)
```

Arguments

`Hst0.ls`	Space-time covariates (of interpolation sites). A list of length τ, each element should be a $n*$ x p_st numeric matrix.
`x`	Space-time standardization object, as created by `stnd.Hst.ls`.

Value

An unnamed list of length τ, each element a $n*$ x p_st numeric matrix.

Examples

```
tau <- 20
n.all <- 10

Hst.ls.all <- list()
for(tt in 1:tau) {
```

```
Hst.ls.all[[tt]] <- cbind(rnorm(n.all, 1, 0.1), rnorm(n.all, -200, 21))
}

ndx.interp <- c(1,3,5)
ndx.support <- I(1:n.all)[ -ndx.interp ]

Hst.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.support)

xsnst.obj <- stnd.Hst.ls(Hst.ls)

Hst0.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.interp)

sHst0.ls <- applystnd.Hst.ls(Hst0.ls, xsnst.obj)



Hst.sumup(xsnst.obj$sHst.ls)

Hst.sumup(sHst0.ls)



## The function is currently defined as
function (Hst0.ls, x)
{
    tau <- length(Hst0.ls)
    sHst0.ls <- list()
    for (i in 1:tau) {
        sHst0.ls[[i]] <- t((t(Hst0.ls[[i]]) - x$h.mean)/x$h.sd)
    }
    return(sHst0.ls)
  }
```

`create.rm.ndx.ls` *Cross-Validation Indices*

Description

Create a list of vectors of indices to remove for *k*-fold cross-validation

Usage

```
create.rm.ndx.ls(n, xincmnt = 10)
```

Arguments

`n`	Number of sites. A scalar integer.
`xincmnt`	How many cv folds, i.e., *k*.

Details

The name of the object produced by this function is commonly `rm.ndx` in this documentation. See `MSS.snow` for a reminder that this object is passed out-of-scope when using `MSS.snow`.

In this package `rm.ndx` is used by `Hals.fastcv.snow` and `widals.snow`; however, creating this object as a list using this function is only necessary when using `widals.snow` with `cv=2` (i.e., 'true' cross-validation).

Value

An unnamed list of integer (>0) vectors.

Examples

```
n <- 100
xincmnt <- 7
rm.ndx <- create.rm.ndx.ls(n=n, xincmnt=xincmnt)
rm.ndx

######## if we want randomization of indices:
n <- 100
xincmnt <- 7
rm.ndx <- create.rm.ndx.ls(n=n, xincmnt=xincmnt)

rnd.ndx <- sample(I(1:n))
for(i in 1:length(rm.ndx)) { rm.ndx[[i]] <- rnd.ndx[rm.ndx[[i]]] }
rm.ndx

## The function is currently defined as
function (n, xincmnt = 10)
{
    rm.ndx.ls <- list()
    for (i in 1:xincmnt) {
        xrm.ndxs <- seq(i, n + xincmnt, by = xincmnt)
        xrm.ndxs <- xrm.ndxs[xrm.ndxs <= n]
        rm.ndx.ls[[i]] <- xrm.ndxs
    }
    return(rm.ndx.ls)
}
```

`crispify` *Observation-Space Stochastic Correction*

Description

Improve observation-space predictions using 'left over' spacial correlation between model residuals

Usage

```
crispify(locs1, locs2, Z.delta, z.lags.vec, geodesic, alpha, flatten, self.refs,
lags, stnd.d = FALSE, log10cutoff = -16)
```

Arguments

`locs1`	Locations of supporting sites. An n x 3 matrix, first column is spacial x, second column is spacial y, third column contains relative temporal 'distance'. If the `geodesic` is `TRUE`, make sure latitude is in the first column.
`locs2`	Locations of interpolation sites. An $n*$ x 3 matrix, where $n*$ is the number of interpolation sites. See `locs1` above.
`Z.delta`	Observed residuals. A τ x x matrix.
`z.lags.vec`	Temporal lags. An integer vector or scalar.
`geodesic`	Use geodesic distance? Boolean. If true, distance (used internally) is in units kilometers.
`alpha`	The WIDALS distance rate hyperparameter. A scalar non-negative number.
`flatten`	The WIDALS 'flattening' hyperparameter. A scalar non-negative number. Typically between 0 and some number slightly greater than 1. When 0, no crispification.
`self.refs`	Which sites are self-referencing? An integer vector of (zero-based) lag indices, OR a scalar set to `-1`. This argument only has meaning when `locs1` is identical to `locs2`. If the `lags` argument is, say, 0, then it would be pointless to smooth predictions with existing values. In this case, we can set `self.refs = 0`. If `locs1` is NOT the same as `locs2`, then set this argument to `-1`.
`lags`	Temporal lags. An integer vector or scalar. E.g., if the data's time increment is daily, then `lags = c(-1,0,1)` would have `crispify` smooth today's predictions using yesterdays, today's, and tomorrow's observed residuals.
`stnd.d`	Spacial compression. Boolean.
`log10cutoff`	Weight threshold. A scalar number. A value of, e.g., -10, will instruct `crispify` to ignore weights less than 10^(-10) when smoothing.

Details

This function is called inside `widals.predict` and `widals.snow`. It may be useful for the user in building their own WIDALS model extensions.

Value

A τ x x matrix.

See Also

`widals.predict`, `widals.snow`.

Examples

```
######### here's an itty-bitty example

######### simulate itty-bitty data
```

```
tau <- 21 #### number of time points

d.alpha <- 2
R.scale <- 1
sigma2 <- 0.01
F <- 1
Q <- 0

n.all <- 14 ##### number of spacial locations

set.seed(9999)


library(SSsimple)

locs.all <- cbind(runif(n.all, -1, 1), runif(n.all, -1, 1)) #### random location of sensors
D.mx <- distance(locs.all, locs.all, FALSE) #### distance matrix

#### create measurement variance using distance and covariogram
R.all <- exp(-d.alpha*D.mx) + diag(sigma2, n.all)

Hs.all <- matrix(1, n.all, 1) #### constant mean function

##### use SSsimple to simulate system
xsssim <- SS.sim(F=F, H=Hs.all, Q=Q, R=R.all, length.out=tau, beta0=0)
Z.all <- xsssim$Z ###### system observation matrix


######## suppose use the global mean as a prediction

z.mean <- mean(Z.all)

Z.delta <- Z.all - z.mean


z.lags.vec <- rep(0, n.all)

geodesic <- FALSE
alpha <- 5
flatten <- 1

## emmulate cross-validation, i.e.,
## don't use observed site values to predict themselves (zero-based)
self.refs <- 0
lags <- 0

locs1 <- cbind(locs.all, rep(0, n.all))
locs2 <- cbind(locs.all, rep(0, n.all))

Z.adj <- crispify(locs1, locs2, Z.delta, z.lags.vec, geodesic, alpha,
    flatten, self.refs, lags, stnd.d = FALSE, log10cutoff = -16)

Z.adj
```

```
Z.hat <- z.mean + Z.adj

sqrt( mean( (Z.all - Z.hat)^2 ) )


######### set flatten to zero -- this means no crispification

Z.adj <- crispify(locs1, locs2, Z.delta, z.lags.vec, geodesic, alpha,
    flatten=0, self.refs, lags, stnd.d = FALSE, log10cutoff = -16)

Z.adj

Z.hat <- z.mean + Z.adj

sqrt( mean( (Z.all - Z.hat)^2 ) )
```

`distance` *Spacial Distance*

Description

Calculate spacial distance between two sets of locations (in two-space)

Usage

```
distance(locs1, locs2, geodesic = FALSE)
```

Arguments

`locs1`	First set of locations. E.g., supporting sites: An n x 2 matrix. If `geodesic` is set to true, make sure to place latitude in first column.
`locs2`	Second set of locations. E.g., interpolation sites: An n* x 2 matrix. If `geodesic` is set to true, make sure to place latitude in first column.
`geodesic`	Use geodesic distance? Boolean.

Details

If `geodesic` is set to `FALSE`, Euclidean distance is returned; if `TRUE`, Earth's geodesic distance is returned in units kilometers.

Value

An n x n* matrix.

Examples

```
locs1 <- cbind( c(-1, -1, 1, 1), c(-1, 1, -1, 1) )
locs2 <- cbind( c(0), c(0) )

distance(locs1, locs2)


locs1 <- cbind( c(32, 0), c(-114, -114) )
locs2 <- cbind( c(0), c(0) )

distance(locs1, locs2, TRUE)


####### separation of one deg long at 88 degs lat (near North-Pole) is (appx)
locs1 <- cbind( c(88), c(-114) )
locs2 <- cbind( c(88), c(-115) )
distance(locs1, locs2, TRUE)

####### separation of one deg long at 0 degs lat (Equator) is (appx)
locs1 <- cbind( c(0), c(-114) )
locs2 <- cbind( c(0), c(-115) )
distance(locs1, locs2, TRUE)





## The function is currently defined as
function (locs1, locs2, geodesic = FALSE)
{
#    dyn.load("~/Files/Creations/C/distance.so")
    n1 <- nrow(locs1)
    n2 <- nrow(locs2)
    d.out <- rep(0, n1 * n2)
    if (geodesic) {
        D.Mx <- .C("distance_geodesic_AB", as.double(locs1[,
            1] * pi/180), as.double(locs1[, 2] * pi/180), as.double(locs2[,
            1] * pi/180), as.double(locs2[, 2] * pi/180), as.double(d.out),
            as.integer(n1), as.integer(n2))[[5]]
    }
    else {
        D.Mx <- .C("distance_AB", as.double(locs1[, 1]), as.double(locs1[,
            2]), as.double(locs2[, 1]), as.double(locs2[, 2]),
            as.double(d.out), as.integer(n1), as.integer(n2))[[5]]
    }
    D.out <- matrix(D.Mx, n1, n2)
    return(D.out)
  }
```

dlog.norm *Local Search Function*

Description

Local hyperparameter exponentiated-normal search function

Usage

```
dlog.norm(n, center, sd)
```

Arguments

n	Sample size. A positive scalar integer.
center	Exponential of the mean. A numeric scalar (or vector).
sd	Standard deviation. A numeric scalar (or vector).

Details

This function can be used by `MSS.snow`.

Value

A numeric vector of length n.

See Also

`unif.mh`, `MSS.snow`, `fun.load`.

Examples

```
x <- dlog.norm(100, 1, 1)
hist(x)

## The function is currently defined as
function (n, center, sd)
{
    return(exp(rnorm(n, log(center), sd)))
  }
```

fun.load *Stochastic Search Helper Functions*

Description

Functions that assign values and functions needed by MSS.snow

Usage

```
fun.load.hals.a()
fun.load.hals.fill()
fun.load.widals.a()
fun.load.widals.fill()
```

Details

Please see MSS.snow and examples.

Value

Nothing. The central role of these functions is the creation of four functions required by MSS.snow: FUN.MH, FUN.GP, FUN.I, and FUN.EXIT. These four functions are assigned to the Global Environment. This fun.load suite of functions also passes needed objects (out-of-scope) to snowfall threads if the global user-made variable run.parallel is set to TRUE.

See Also

MSS.snow

Examples

```
### Here's an itty bitty example:
### we use stochastic search to find the minimum number in a vector
### GP isn't used here, and hence neither are p.ndx.ls nor f.d
### however, we still need to create them since MSS.snow requires their existence

## Not run:

fun.load.simpleExample <- function() {

   if( run.parallel ) {
         sfExport("xx")
    }

    p.ndx.ls <- list( c(1) )
```

```
        p.ndx.ls <<- p.ndx.ls

        f.d <- list( dlog.norm )

        f.d <<- f.d

        FUN.MH <- function(jj, GP.mx, X) {
            our.cost <- sample(xx, 1)
        }

        FUN.MH <<- FUN.MH


        FUN.GP <- NULL
        FUN.GP <<- FUN.GP


        FUN.I <- function(envmh, X) {
         cat( "Hello, I have found an even smaller number in xx ---> ", envmh$current.best, "\n" )
        }
        FUN.I <<- FUN.I

        FUN.EXIT <- function(envmh, X) {
            cat( "Done",   "\n" )
        }

        FUN.EXIT <<- FUN.EXIT

    }

    xx <- 1:600

    GP <- c(1)

    run.parallel <- TRUE
    sfInit(TRUE, 2)

    MH.source <- fun.load.simpleExample
    MH.source()

    MSS.snow(MH.source, Inf, p.ndx.ls, f.d, matrix(1, nrow=28), 28, 7)
    sfStop()




    ### Here's another itty bitty example:
    ### we use stochastic search to find the mean of a vector
    ### i.e., the argmin? of sum ( x - ? )^2

    fun.load.simpleExample2 <- function() {

       if( run.parallel ) {
            sfExport("xx")
        }

        p.ndx.ls <- list( c(1) )
        p.ndx.ls <<- p.ndx.ls

        f.d <- list( unif.mh )
        f.d <<- f.d

        FUN.MH <- function(jj, GP.mx, X) {
            our.cost <- sum( ( xx - GP.mx[jj, 1] )^2 )
            return(our.cost)
        }
        FUN.MH <<- FUN.MH

        FUN.GP <- NULL
        FUN.GP <<-  FUN.GP

        FUN.I <- function(envmh, X) {
            cat( "Improvement ---> ", envmh$current.best, " ---- " , envmh$GP, "\n" )
        }
        FUN.I <<- FUN.I

        FUN.EXIT <- function(envmh, X) {
            our.cost <- envmh$current.best
            GP <- envmh$GP
            cat( "Done",   "\n" )
            cat( envmh$GP, our.cost, "\n" )
        }
        FUN.EXIT <<- FUN.EXIT

    }

    ##set.seed(99999)
    xx <- rnorm(300, 5, 10)

    GP <- c(1)

    run.parallel <- TRUE
    sfInit(TRUE, 2)

    MH.source <- fun.load.simpleExample2
    MH.source()

    MSS.snow(MH.source, Inf, p.ndx.ls, f.d, matrix(1/10, nrow=140, ncol=length(GP)), 140, 14)
    sfStop()

    ##### in fact:
    mean(xx)


    ## End(Not run)
```

`fuse.Hst.ls` *Merge Contemporaneous Space-Time Covariates*

Description

Fuse together two lists of spacio-temporal covariates

Usage

```
fuse.Hst.ls(Hst.ls1, Hst.ls2)
```

Arguments

`Hst.ls1`	Space-time covariates. A list of length τ, each element should be a numeric n x p_1 matrix.
`Hst.ls2`	Space-time covariates. A list of length τ, each element should be a numeric n x p_2 matrix.

Value

An unnamed list of length τ, each element will be a numeric n x $(p_1 + p_2)$ matrix.

Examples

```
set.seed(9999)

tau <- 5
n <- 7

p1 <- 2
Hst.ls1 <- list()
for(i in 1:tau) { Hst.ls1[[i]] <- matrix(rnorm(n*p1), nrow=n) }

p2 <- 3
Hst.ls2 <- list()
for(i in 1:tau) { Hst.ls2[[i]] <- matrix(rnorm(n*p2), nrow=n) }

fuse.Hst.ls(Hst.ls1, Hst.ls2)


## The function is currently defined as
function (Hst.ls1, Hst.ls2)
{
    tau <- length(Hst.ls1)
    for (i in 1:tau) {
        Hst.ls1[[i]] <- cbind(Hst.ls1[[i]], Hst.ls2[[i]])
    }
    return(Hst.ls1)
  }
```

H.als.b *Adaptive Least Squares*

Description

Adaptive Least Squares expecially for large spacio-temporal data

Usage

```
H.als.b(Z, Hs, Ht, Hst.ls, rho, reg, b.lag = -1, Hs0 = NULL, Ht0 = NULL, Hst0.ls = NULL)
```

Arguments

`Z`	Space-time data. A τ x n numeric matrix.
`Hs`	Spacial covariates (of supporting sites). An n x p_s numeric matrix.
`Ht`	Temporal covariates (of supporting sites). A τ x p_t numeric matrix.
`Hst.ls`	Space-time covariates (of supporting sites). A list of length τ, each element should be a n x p_st numeric matrix.
`rho`	ALS signal-to-noise ratio (SNR). A non-negative scalar.
`reg`	ALS regularizer. A non-negative scalar.
`b.lag`	ALS lag. A scalar integer, typically -1 (*a-prior*), or 0 (*a-posteriori*).
`Hs0`	Spacial covariates (of interpolation sites). An n* x p_s matrix, or `NULL`.
`Ht0`	Temporal covariates (of interpolation sites). A τ x p_t matrix, or `NULL`. If not `NULL`, I cannot imagine a scenario where this shouldn't be `Ht`.
`Hst0.ls`	Space-time covariates (of interpolation sites). A list of length τ, each element should be a numeric n x p_st matrix.

Value

A named list.

`Z.hat`	A τ x *n* matrix, the *i*th row of which is the ALS prediction of the supporting sites at time *i*.
`B`	A τ x $(p_s + p_t + p_st)$ matrix, the *i*th row of which is the ALS state (partial slopes) prediction at time *i*.
`Z0.hat`	A τ x *n** matrix, the *i*th row of which is the ALS prediction of the interpolation sites at time *i*.
`inv.LHH`	A $(p_s + p_t + p_st)$ x $(p_s + p_t + p_st)$ matrix. This is the (ALS predicted) covariate precision matrix at time τ.
`ALS.g`	The ALS gain at time τ.

Examples

```
set.seed(99999)

library(SSsimple)

tau <- 70
n.all <- 14

Hs.all <- matrix(rnorm(n.all), nrow=n.all)
Ht <- matrix(rnorm(tau*2), nrow=tau)
Hst.ls.all <- list()
for(i in 1:tau) { Hst.ls.all[[i]] <- matrix(rnorm(n.all*2), nrow=n.all) }

Hst.combined <- list()
for(i in 1:tau) {
    Hst.combined[[i]] <- cbind( Hs.all, matrix(Ht[i, ], nrow=n.all, ncol=ncol(Ht),
    byrow=TRUE), Hst.ls.all[[i]] )
}

######## use SSsimple to simulate
sssim.obj <- SS.sim.tv( 0.999, Hst.combined, 0.01, diag(1, n.all), tau )


ndx.support <- 1:10
ndx.interp <- 11:14

Z.all <- sssim.obj$Z
Z <- Z.all[ , ndx.support]
Z0 <- Z.all[ , ndx.interp]

Hst.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.support)
Hst0.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.interp)

Hs <- Hs.all[ ndx.support, , drop=FALSE]
Hs0 <- Hs.all[ ndx.interp, , drop=FALSE]

xrho <- 1/10
xreg <- 1/10
xALS <- H.als.b(Z=Z, Hs=Hs, Ht=Ht, Hst.ls=Hst.ls, rho=xrho, reg=xreg, b.lag=-1,
Hs0=Hs0, Ht0=Ht, Hst0.ls=Hst0.ls)



test.rng <- 20:tau

errs.sq <- (Z0 - xALS$Z0.hat)^2
sqrt( mean(errs.sq[test.rng, ]) )


################ calculate the 'effective standard errors' (actually 'effective prediction
################ errors') of the ALS partial slopes
rmse <- sqrt(mean((Z[test.rng, ] - xALS$Z.hat[test.rng, ])^2))
rmse
als.se <- rmse * sqrt(xALS$ALS.g) * sqrt(diag(xALS$inv.LHH))
cbind(xALS$B[tau, ], als.se, xALS$B[tau, ]/als.se)
```

H.Earth.solar *Solar Radiation*

Description

Calculate Incident Solar Area (ISA)

Usage

```
H.Earth.solar(x, y, dateDate)
```

Arguments

x	Longitude. Numeric vector of length n.
y	Latitude. Numeric vector of length n.
dateDate	Posix date. Numeric vector of length τ.

Details

This function returns a spacio-temporal covariate list (Earth's ISA is space-time *non-seperable*). A negative value indicates that at that time (list index), and at that location (matrix row), the sun is below the horizon all day.

Value

An unnamed list of length τ, each element of which is an n x 1 matrix.

Examples

```
lat <- c(0, -88)
lon <- c(0, 0)
dateDate <- strptime( c('20120621', '20120320'), '%Y%m%d')

H.Earth.solar(lon, lat, dateDate)


## The function is currently defined as
function (x, y, dateDate)
{
    Hst.ls <- list()
    n <- length(y)
    tau <- length(dateDate)
    equinox <- strptime("20110320", "%Y%m%d")
    for (i in 1:tau) {
        this.date <- dateDate[i]
        dfe <- as.integer(difftime(this.date, equinox, units = "day"))
        dfe
        psi <- 23.5 * sin(2 * pi * dfe/365.25)
        psi
        eta <- 90 - (360/(2 * pi)) * acos(cos(2 * pi * y/360) *
            cos(2 * pi * psi/360) + sin(2 * pi * y/360) * sin(2 *
            pi * psi/360))
        surface.area <- sin(2 * pi * eta/360)
        surface.area
        Hst.ls[[i]] <- cbind(surface.area)
    }
    return(Hst.ls)
  }
```

`Hals.fastcv.snow` *ALS Spacial Cross-Validation*

Description

Fit Adaptive Least Squares with k-fold cross-validation

Usage

```
Hals.fastcv.snow(j, rm.ndx, Z, Hs, Ht, Hst.ls, GP.mx)
```

Arguments

`j`	Index used by `snowfall`. A scalar integer. Which row of `GP.mx` to use for the ALS hyperparameters, `GP`.
`rm.ndx`	A list of vectors of indices to remove for k-fold cross-validation.
`Z`	Data. A τ x n numeric matrix.
`Hs`	Spacial covariates. An n x p_s numeric matrix.
`Ht`	Temporal covariates. An τ x p_t numeric matrix.
`Hst.ls`	Space-time covariates. A list of length τ, each element containing a n x p_st numeric matrix.
`GP.mx`	Hyperparameters. A $k.glob$ x 2 non-negative matrix. See `MSS.snow`.

Value

A τ x n numeric matrix. The ALS cross-validated predictions of `Z`.

See Also

Hals.snow, MSS.snow.

Examples

```
set.seed(99999)



library(SSsimple)

tau <- 70
n.all <- 14

Hs.all <- matrix(rnorm(n.all), nrow=n.all)
Ht <- matrix(rnorm(tau*2), nrow=tau)
Hst.ls.all <- list()
for(i in 1:tau) { Hst.ls.all[[i]] <- matrix(rnorm(n.all*2), nrow=n.all) }

Hst.combined <- list()
for(i in 1:tau) {
    Hst.combined[[i]] <- cbind( Hs.all, matrix(Ht[i, ], nrow=n.all,
    ncol=ncol(Ht), byrow=TRUE), Hst.ls.all[[i]] )
}

######## use SSsimple to simulate
sssim.obj <- SS.sim.tv( 0.999, Hst.combined, 0.01, diag(1, n.all), tau )



Z.all <- sssim.obj$Z
Z <- Z.all
n <- n.all

Hst.ls <- Hst.ls.all

Hs <- Hs.all

xrho <- 1/10
xreg <- 1/10

GP.mx <- matrix(c(xrho, xreg), nrow=1)

rm.ndx <- create.rm.ndx.ls(n, 10)

Zcv <- Hals.fastcv.snow(j=1, rm.ndx, Z, Hs, Ht, Hst.ls, GP.mx)



test.rng <- 20:tau
```

```
errs.sq <- (Z - Zcv)^2
sqrt( mean(errs.sq[test.rng, ]) )
```

`Hals.ses` *Effective Standard Errors*

Description

Calculate the ALS so-called 'effective standard errors'

Usage

```
Hals.ses(Z, Hs, Ht, Hst.ls, rho, reg, b.lag, test.rng)
```

Arguments

`Z`	Space-time data. A τ x n numeric matrix.
`Hs`	Spacial covariates (of supporting sites). An n x p_s numeric matrix.
`Ht`	Temporal covariates (of supporting sites). A τ x p_t numeric matrix.
`Hst.ls`	Space-time covariates (of supporting sites). A list of length τ, each element should be a numeric n x $p_s t$ matrix.
`rho`	ALS signal-to-noise ratio (SNR). A non-negative scalar.
`reg`	ALS regularizer. A non-negative scalar.
`b.lag`	ALS lag. A scalar integer, typically -1 (*a-prior*), or 0 (*a-posteriori*).
`test.rng`	Temporal test range. A vector of temporal indices of the model test range.

Value

A named list.

`estimates`	A p_s+p_t+$p_s t$ x 2 matrix, each row giving the ALS partial slope estimate/prediction at time τ, and the 'effective standard error (prediction error)' for the partial slope.
`inv.LHH`	A $(p_s + p_t + p_s t)$ x $(p_s + p_t + p_s t)$ matrix. This is the (ALS predicted) covariate precision matrix at time τ.
`ALS.g`	The ALS gain at time τ.

Examples

```
## Please see the example in H.als.b

## The function is currently defined as
function (Z, Hs, Ht, Hst.ls, rho, reg, b.lag, test.rng)
{
    tau <- nrow(Z)
    xALS <- H.als.b(Z = Z, Hs = Hs, Ht = Ht, Hst.ls = Hst.ls,
        rho = rho, reg = reg, b.lag = b.lag, Hs0 = NULL, Ht0 = NULL,
        Hst0.ls = NULL)
    rmse <- sqrt(mean((Z[test.rng, ] - xALS$Z.hat[test.rng, ])^2))
    rmse
    als.se <- rmse * sqrt(xALS$ALS.g) * sqrt(diag(xALS$inv.LHH))
    return(list(estimates = cbind(xALS$B[tau, ], als.se), inv.LHH = xALS$inv.LHH,
        ALS.g = xALS$ALS.g))
  }
```

`Hals.snow` *Fit ALS*

Description

Fit Adaptive Least Squares

Usage

```
Hals.snow(j, Z, Hs, Ht, Hst.ls, b.lag, GP.mx)
```

Arguments

`j`	Index used by `snowfall`. A scalar integer. Which row of `GP.mx` to use for the ALS hyperparameters, `GP`.
`Z`	Data. A τ x n numeric matrix.
`Hs`	Spacial covariates. An n x p_s numeric matrix.
`Ht`	Temporal covariates. An τ x p_t numeric matrix.
`Hst.ls`	Space-time covariates. A list of length τ, each element containing a n x p_st numeric matrix.
`b.lag`	ALS lag. A scalar integer, typically -1 (*a-prior*), or 0 (*a-posteriori*).
`GP.mx`	Hyperparameters. A $k.glob$ x 2 non-negative matrix. See `MSS.snow`.

Value

A τ x n numeric matrix. The ALS predictions of `Z`.

See Also

`Hals.fastcv.snow`, `MSS.snow`.

Examples

```
set.seed(9999)


library(SSsimple)

tau <- 280
n.all <- 35

Hs.all <- matrix(rnorm(n.all), nrow=n.all)
Ht <- matrix(rnorm(tau*2), nrow=tau)
Hst.ls.all <- list()
for(i in 1:tau) { Hst.ls.all[[i]] <- matrix(rnorm(n.all*3), nrow=n.all) }

Hst.combined <- list()
for(i in 1:tau) {
    Hst.combined[[i]] <- cbind( Hs.all, matrix(Ht[i, ], nrow=n.all,
    ncol=ncol(Ht), byrow=TRUE), Hst.ls.all[[i]] )
}

######## use SSsimple to simulate
sssim.obj <- SS.sim.tv( 0.999, Hst.combined, 0.1, diag(1, n.all), tau )



Z.all <- sssim.obj$Z
Z <- Z.all
n <- n.all

Hst.ls <- Hst.ls.all

Hs <- Hs.all

xrho <- 1/10
xreg <- 1/10
b.lag <- -1

GP.mx <- matrix(c(xrho, xreg), nrow=1)

Zcv <- Hals.snow(j=1, Z, Hs, Ht, Hst.ls, b.lag, GP.mx)


test.rng <- 20:tau

errs.sq <- (Z - Zcv)^2
sqrt( mean(errs.sq[test.rng, ]) )
```

`Hst.sumup` *Create Covariance Matrix*

Description

Calculate the covariance matrix of all model covariates

Usage

```
Hst.sumup(Hst.ls, Hs = NULL, Ht = NULL)
```

Arguments

`Hst.ls`	Space-time covariates. A list of length τ, each element containing a n x p_st numeric matrix.
`Hs`	Spacial covariates. An n x p_s numeric matrix.
`Ht`	Temporal covariates. An τ x p_t numeric matrix.

Details

Important: The order of the arguments in this function is NOT the same as in the returned covariance matrix. The order in the covariance matrix is the same as in other functions in this package: `Hs`, `Ht`, `Hst.ls`.

Value

A $(p_s + p_t + p_st)$ x $(p_s + p_t + p_st)$ numeric, symmetrix, non-negative definite matrix.

Examples

```
tau <- 20
n <- 10
Ht <- cbind(sin(1:tau), cos(1:tau))

Hs <- cbind(rnorm(10), rnorm(n, 5, 49))

Hst.ls <- list()
for(tt in 1:tau) {
Hst.ls[[tt]] <- cbind(rnorm(n, 1, 0.1), rnorm(n, -200, 21))
}


Hst.sumup(Hst.ls, Hs, Ht)



########### standardize all covariates

x1 <- stnd.Hst.ls(Hst.ls, NULL)$sHst.ls
x2 <- stnd.Hs(Hs, NULL, FALSE)$sHs
x3 <- stnd.Ht(Ht, n)
```

```
Hst.sumup(x1, x2, x3)



## The function is currently defined as
function (Hst.ls, Hs = NULL, Ht = NULL)
{
    tau <- length(Hst.ls)
    if(tau < 1) { tau <- nrow(Ht) }
    if(is.null(tau)) { tau <- 10 ; cat("tau assumed to be 10.", "\n") }
    n <- nrow(Hst.ls[[1]])
    if(is.null(n)) { n <- nrow(Hs) }
    big.sum <- 0
    for (i in 1:tau) {
        if (!is.null(Ht)) {
            Ht.mx <- matrix(Ht[i, ], n, ncol(Ht), byrow = TRUE)
        }
        else {
            Ht.mx <- NULL
        }
        big.sum <- big.sum + crossprod(cbind(Hs, Ht.mx, Hst.ls[[i]]))
    }
    return(big.sum)
  }
```

`load.Hst.ls.2Zs` *Load Observations into Space-Time Covariates*

Description

Insert an observation matrix into space-time covariates, but segregate based on missing values

Usage

```
load.Hst.ls.2Zs(Z, Z.na, Hst.ls.Z, xwhich, rgr.lags = c(0))
```

Arguments

`Z`	Observation data. A τ x n numeric matrix.
`Z.na`	Missing data indicator. A τ x n boolean matrix.
`Hst.ls.Z`	Space-time covariates. A list of length τ, each element should be a numeric n x p_st matrix.
`xwhich`	Which column-pair of `Hst.ls.Z[[i]]` to insert into the ith row of `Z`. A scalar positive integer. By 'column-pair', we mean, e.g., a value of 1 will fill columns 1 and 2, a value of 2 will fill columns 3 and 4, a value of 3 will fill columns 5 and 6, etc.
`rgr.lags`	Temporal lagging of `Z`. A scalar integer.

Details

This function, along with load.Hst.ls.Z, allows the user to convert a set of observations into covariates for another set of observations. Unlike load.Hst.ls.Z, this function *splits* Z based on the argument Z.na. Values associated with FALSE elements of Z.na are placed into the first column of the specified column-pair of Hst.ls.Z, Values associated with TRUE elements of Z.na are placed into the second column of the specified column-pair of Hst.ls.Z (all other values in in the specified column-pair of Hst.ls.Z are zeroed).

Value

An unnamed list of length τ, each element will be a numeric n x $p_s t$ matrix.

See Also

load.Hst.ls.Z.

Examples

```
###### here's an itty-bitty example

tau <- 7
n <- 5

Z <- matrix(1, tau, n)

Z.na <- matrix(FALSE, tau, n)
Z.na[2:3, 4] <- TRUE

Z[Z.na] <- 2

Hst.ls <- list()
for(i in 1:tau) { Hst.ls[[i]] <- matrix(rnorm(n*4), nrow=n) }


load.Hst.ls.2Zs(Z, Z.na, Hst.ls.Z=Hst.ls, 1, 0)


########## insert into cols 3 and 4

load.Hst.ls.2Zs(Z, Z.na, Hst.ls.Z=Hst.ls, 2, 0)
```

`load.Hst.ls.Z` *Load Observations into Space-Time Covariates*

Description

Insert an observation matrix into space-time covariates

Usage

```
load.Hst.ls.Z(Z, Hst.ls.Z, xwhich, rgr.lags = c(0))
```

Arguments

`Z`	Observation data. A τ x n numeric matrix.
`Hst.ls.Z`	Space-time covariates. A list of length τ, each element should be a numeric n x p_st matrix.
`xwhich`	Which column of `Hst.ls.Z[[i]]` to insert into the ith row of `Z`. A scalar positive integer.
`rgr.lags`	Temporal lagging of `Z`. A scalar integer.

Details

This function, along with `load.Hst.ls.2Zs`, allows the user to convert a set of observations into covariates for another set of observations.

Value

An unnamed list of length τ, each element will be a numeric n x p_st matrix.

See Also

`load.Hst.ls.2Zs`.

Examples

```
###### here's an itty-bitty example

tau <- 7
n <- 5

Z <- matrix(1, tau, n)

Hst.ls <- list()
for(i in 1:tau) { Hst.ls[[i]] <- matrix(rnorm(n*4), nrow=n) }
```

```
load.Hst.ls.Z(Z, Hst.ls.Z=Hst.ls, 1, 0)


########## insert into col 3

load.Hst.ls.Z(Z, Hst.ls.Z=Hst.ls, 3, 0)



############ lag Z examples

Z <- matrix(1:tau, tau, n)

######### lag -1 Z

load.Hst.ls.Z(Z, Hst.ls.Z=Hst.ls, 1, -1)

######### lag 0 Z -- default

load.Hst.ls.Z(Z, Hst.ls.Z=Hst.ls, 1, 0)

######### lag +1 Z

load.Hst.ls.Z(Z, Hst.ls.Z=Hst.ls, 1, +1)
```

MSS.snow *Metaheuristic Stochastic Search*

Description

Locate WIDALS hyperparameters

Usage

```
MSS.snow(FUN.source, current.best, p.ndx.ls, f.d, sds.mx, k.glob, k.loc.coef, X = NULL)
```

Arguments

FUN.source	Search function definitions (see Details). A path to source code, or function, e.g., fun.load.widals.a.
current.best	An initial cost. A scalar. Setting to NA will cause MSS.snow to make an initial pass over the data to create an initial cost to beat.
p.ndx.ls	Hyperparameter indices (of GP) to search. A list of vectors. For example, list(c(1,2), c(3,4,5)) will instruct MSS.snow, for each local search, to search over the first two hyperparameters as a pair, then to search the last three as a group.

`f.d`	Local search functions. A list of functions (one for each element of GP). Typically, for WIDALS, all five will be `dlog.norm`.
`sds.mx`	The standard deviations for `f.d`. An *k.glob* x *q* matrix, where *q* is the number of hyperparameters, i.e., the length of GP.
`k.glob`	The number of global searches. A scalar integer.
`k.loc.coef`	The coeficient for the number of local searches to make. A scalar integer.
`X`	A placeholder for values to be passed between functions inside `MSS.snow` (see Details).

Details

This function requires the presence of a number of values and functions out-of-scope. It is assumed that these are available in the Global Environment. They are: `run.parallel` (boolean), `FUN.MH` (a function that creates, for a given `GP`, a cost), `FUN.GP` (a function that applies constraints to `GP`), `FUN.I` (a function that does something when local searches have reduced the cost), `FUN.EXIT` (a function that does something when `MSS.snow` is done).

Examine the code for `fun.load.widals.a` for an example of the four functions described above. Note that these four functions may themselves require objects out-of-scope.

In general, for a given `R` session, special care should be taken concerning the naming and assigning of the following objects: `Z` (the space-time data), `Z.na` (a boolean matrix indicating missing values in `Z`), `locs` (site locations), `Hs` (spacial covariates), `Ht` (temporal covariates), `Hst.ls` (space-time covariates), `lags` (temporal lag vector), `b.lag` (the ALS lag), `cv` (cross-validation switch), `xgeodesic` (boolean), `ltco` (weight cut-off), `GP` (hyperparameter vector), `run.parralel` (boolean), `stnd.d` (boolean), `train.rng` (time index vector), `test.rng` (time index vector).

Value

Nothing. After completion, the best hyperparameters, `GP`, are assigned to the Global Environment.

See Also

`Hals.fastcv.snow`, `Hals.snow`, `widals.snow`.

Examples

```
##### simulate a state-space system (using pkg SSsimple)

## Not run:

### using dontrun because of excessive run time for CRAN submission

set.seed(9999)

library(SSsimple)


tau <- 77 #### number of time points
```

```
d.alpha <- 2
R.scale <- 1
sigma2 <- 0.01
F <- 0.999
Q <- 0.1

udom <- (0:300)/100
plot( udom,   R.scale * exp(-d.alpha*udom) ,  type="l", col="red" ) #### see the covariogram

n.all <- 70 ##### number of spacial locations

set.seed(9999)
locs.all <- cbind(runif(n.all, -1, 1), runif(n.all, -1, 1)) #### random location of sensors

D.mx <- distance(locs.all, locs.all, FALSE) #### distance matrix

#### create measurement variance using distance and covariogram
R.all <- exp(-d.alpha*D.mx) + diag(sigma2, n.all)

Hs.all <- matrix(1, n.all, 1) #### constant mean function

##### use SSsimple to simulate system
xsssim <- SS.sim(F=F, H=Hs.all, Q=Q, R=R.all, length.out=tau, beta0=0)

Z.all <- xsssim$Z ###### system observation matrix



########### now make assignments required by MSS.snow



##### randomly remove five sites to serve as interpolation points
ndx.interp <- sample(1:n.all, size=5)
ndx.support <- I(1:n.all)[ -ndx.interp ] ##### support sites



########### what follows are important assignments,
########### since MSS.snow and the four helper functions
########### will look for these in the Global Environment
########### to commence fitting the model (as noted in Details above)
train.rng <- 30:(tau) ; test.rng <- train.rng

Z <- Z.all[ , ndx.support ]
Hs <- Hs.all[ ndx.support, , drop=FALSE]
locs <- locs.all[ndx.support, , drop=FALSE]

Ht <- NULL
Hst.ls <- NULL

lags <- c(0)
b.lag <- c(-1)
```

```
cv <- -2
xgeodesic <- FALSE
stnd.d <- FALSE
ltco <- -10
GP <- c(1/10, 1, 20, 20, 1) ### -- initial hyperparameter values
run.parallel <- TRUE

if( cv==2 ) { rm.ndx <- create.rm.ndx.ls( nrow(Hs), 14 ) } else { rm.ndx <- 1:nrow(Hs) }
rgr.lower.limit <- 10^(-7) ; d.alpha.lower.limit <- 10^(-3) ; rho.upper.limit <- 10^(4)


############## tell snowfall to use two threads for local searches
sfInit(TRUE, cpus=2)
fun.load.widals.a()


######## now, finally, search for best fit over support
######## Note that p.ndx.ls and f.d are produced inside fun.load.widals.a()
MSS.snow(fun.load.widals.a, NA, p.ndx.ls, f.d, matrix(1/10, 10, length(GP)), 10, 7)
sfStop()

######## we can use these hyperparameters to interpolate to the
######## deliberately removed sites, and measure MSE, RMSE
Z0.hat <- widals.predict(Z, Hs, Ht, Hst.ls, locs, lags, b.lag,
Hs0=Hs.all[ ndx.interp, , drop=FALSE ],
Hst0.ls=NULL, locs0=locs.all[ ndx.interp, , drop=FALSE],
geodesic = xgeodesic, wrap.around = NULL, GP, stnd.d = stnd.d, ltco = ltco)

resids.wid <- ( Z.all[ , ndx.interp ] - Z0.hat )
mse.wid <- mean( resids.wid[ test.rng, ]^2 )
mse.wid
sqrt(mse.wid)





############################################ Simulated Imputation with WIDALS
Z.all <- xsssim$Z
Z.missing <- Z.all

Z.na.all <- matrix( sample(c(TRUE, FALSE), size=n.all*tau, prob=c(0.01, 0.99), replace=TRUE),
tau, n.all)
Z.missing[ Z.na.all ] <- NA


Z <- Z.missing
Z[ is.na(Z) ] <- mean(Z, na.rm=TRUE)
X <- list("Z.fill"=Z)

Z.na <- Z.na.all
Hs <- Hs.all
```

```
locs <- locs.all
Ht <- NULL
Hst.ls <- NULL
lags <- c(0)
b.lag <- c(-1)
cv <- -2
xgeodesic <- FALSE
ltco <- -10
if( cv==2 ) { rm.ndx <- create.rm.ndx.ls( nrow(Hs), 14 ) } else { rm.ndx <- 1:nrow(Hs) }

GP <- c(1/10, 1, 20, 20, 1)

rgr.lower.limit <- 10^(-7) ; d.alpha.lower.limit <- 10^(-3) ; rho.upper.limit <- 10^(4)

run.parallel <- TRUE

sfInit(TRUE, cpus=2)
fun.load.widals.fill()

MSS.snow(fun.load.widals.fill, NA, p.ndx.ls, f.d,
seq(2, 0.01, length=10)*matrix(1/10, 10, length(GP)), 10, 7, X=X)
sfStop()

sqrt(mean(( (Z.all[train.rng, ] - Z.fill[train.rng, ])^2 )[ Z.na[ train.rng, ] ]))







############################################ Now Try with ALS alone

Z.all <- xsssim$Z

GP <- c(1/10, 1) ### -- initial hyperparameter values

############## tell snowfall to use two threads for local searches
sfInit(TRUE, cpus=2)
fun.load.hals.a()

######## now, finally, search for best fit over support
######## Note that p.ndx.ls and f.d are produced inside fun.load.widals.a()
MSS.snow(fun.load.hals.a, NA, p.ndx.ls, f.d, matrix(1/10, 10, length(GP)), 10, 7)
sfStop()

######## we can use these hyperparameters to interpolate to the deliberately removed sites,
######## and measure MSE, RMSE
hals.obj <- H.als.b(Z, Hs, Ht, Hst.ls, rho=GP[1], reg=GP[2], b.lag = b.lag,
Hs0 = Hs.all[ ndx.interp, , drop=FALSE ], Ht0 = NULL, Hst0.ls = NULL)
Z0.hat <- hals.obj$Z0.hat
```

```
resids.als <- ( Z.all[ , ndx.interp ] - Z0.hat )
mse.als <- mean( resids.als[ test.rng, ]^2 )
mse.als
sqrt(mse.als)



########################################### Simulated Imputation with ALS
Z.all <- xsssim$Z
Z.missing <- Z.all

set.seed(99)
Z.na.all <- matrix( sample(c(TRUE, FALSE), size=n.all*tau, prob=c(0.03, 0.97), replace=TRUE),
tau, n.all)
Z.missing[ Z.na.all ] <- NA


Z <- Z.missing
Z[ is.na(Z) ] <- 0 #mean(Z, na.rm=TRUE)
X <- list("Z.fill"=Z)

Z.na <- Z.na.all

Hs <- Hs.all

GP <- c(1/10, 1) ### -- initial hyperparameter values

sfInit(TRUE, cpus=2)
fun.load.hals.fill()

MSS.snow(fun.load.hals.fill, NA, p.ndx.ls, f.d,
seq(3, 0.01, length=10)*matrix(1, 10, length(GP)), 10, 7, X=X)
sfStop()

sqrt(mean(( (Z.all[train.rng, ] - Z.fill[train.rng, ])^2 )[ Z.na[ train.rng, ] ]))


## End(Not run)
```

`O3` *California Ozone*

Description

Daily airborne Ozone concentrations (ppb) over California, 68 fixed sensors, 2005-2006

Usage

```
data(O3)
```

Format

The format is:

List of 5

`$Z` :'data.frame': 730 obs. of 68 variables: Ozone ppb for 68 sensors.

`$locs` :'data.frame': 68 obs. of 2 variables: longitude, latitude for 68 sites.

`$helevs` : elevations (meters) for 68 sites.

`$locs0` : 2358 obs. of 2 variables: longitude, latitude for interpolation grid.

`$helevs0`: elevations (meters) for 2358 interpolation grid sites.

Source

The Ozone data originates from the California Air Resources Board (CARB). The interpolation grid elevations originate from the Google Elevation API.

References

The O3 data comes from the California Air Resources Board (CARB).

Examples

```
data(O3)
```

`rm.cols.Hst.ls` *Remove Space-Time Covariates from Model*

Description

Remove spacial covariates from space-time covariate list

Usage

```
rm.cols.Hst.ls(Hst.ls, rm.col.ndx)
```

Arguments

`Hst.ls`	Space-time covariates (of supporting sites). A list of length τ, each element should be a numeric n x p_st matrix.
`rm.col.ndx`	Which columns of `Hst.ls` to remove. A positive scalar integer.

Value

An unnamed list of length τ, each element will be a numeric n x $p_st - p_rm$ matrix, where p_rm is the length of `rm.col.ndx`.

Examples

```
tau <- 21
n <- 7

pst <- 5
Hst.ls <- list()
for(i in 1:tau) { Hst.ls[[i]] <- matrix(1:pst, n, pst, byrow=TRUE) }

rm.cols.Hst.ls(Hst.ls, c(1,3))


## The function is currently defined as
function (Hst.ls, rm.col.ndx)
{
    tau <- length(Hst.ls)
    for (i in 1:tau) {
        Hst.ls[[i]] <- Hst.ls[[i]][, -rm.col.ndx, drop = FALSE]
    }
    return(Hst.ls)
  }
```

`stnd.Hs` *Standardize Spacial Covariates*

Description

Standardize spacial covariates with respect to both the space and time dimensions

Usage

```
stnd.Hs(Hs, Hs0 = NULL, intercept = TRUE)
```

Arguments

`Hs`	Spacial covariates (of supporting sites). An n x p_s numeric matrix.
`Hs0`	Spacial covariates (of interpolation sites). An $n*$ x p_s numeric matrix.
`intercept`	Include intercept term? Boolean.

Value

A named list.

`sHs`	An n x p_s numeric matrix.
`sHs0`	An $n*$ x p_s numeric matrix.
`h.mean`	The covariates' mean over space.
`h.sd`	The covariates' standard deviation over space.
`n`	Number of support sites.
`intercept`	The supplied intercept argument.

See Also

stnd.Ht, stnd.Hst.ls, applystnd.Hs.

Examples

```
##### Please see the examples in Hst.sumup



## The function is currently defined as
function (Hs, Hs0 = NULL, intercept = TRUE)
{
    n <- nrow(Hs)
    h.mean <- apply(Hs, 2, mean)
    h.sd <- apply(t(t(Hs) - h.mean), 2, function(x) {
        sqrt(sum(x^2))
    })
    h.sd[h.sd == 0] <- 1
    sHs <- t((t(Hs) - h.mean)/h.sd)
    if (intercept) {
        sHs[, 1] <- 1/sqrt(n)
    }
    sHs0 <- NULL
    if (!is.null(Hs0)) {
        sHs0 <- t((t(Hs0) - h.mean)/h.sd)
        if (intercept) {
            sHs0[, 1] <- 1/sqrt(n)
        }
    }
    ls.out <- list(sHs = sHs, sHs0 = sHs0, h.mean = h.mean, h.sd = h.sd,
        n = n, intercept = intercept)
    return(ls.out)
  }
```

---

stnd.Hst.ls *Standardize Space-Time Covariates*

Description

Standardize spacio-temporal covariates with respect to both the spacial and time dimensions

Usage

```
stnd.Hst.ls(Hst.ls, Hst0.ls = NULL)
```

Arguments

`Hst.ls`	Space-time covariates (of supporting sites). A list of length τ, each element should be a numeric n x p_st matrix.
`Hst0.ls`	Space-time covariates (of interpolation sites). A list of length τ, each element should be a numeric $n*$ x p_st matrix.

Value

A named list.

`sHst.ls`	A list of length τ, each element a numeric n x p_st matrix.
`sHst0.ls`	A list of length τ, each element a $n*$ x p_st matrix
`h.mean`	The covariates' mean over space-time.
`h.sd`	The covariates' standard deviation over space-time.

See Also

`stnd.Ht`, `stnd.Hs`, `applystnd.Hst.ls`.

Examples

```
##### Please see the examples in Hst.sumup

## The function is currently defined as
function (Hst.ls, Hst0.ls = NULL)
{
    tau <- length(Hst.ls)
    big.sum <- 0
    for (i in 1:tau) {
        big.sum <- big.sum + apply(Hst.ls[[i]], 2, mean)
    }
    h.mean <- big.sum/tau
    sHst.ls <- list()
    big.sum.mx <- 0
    for (i in 1:tau) {
        sHst.ls[[i]] <- t(t(Hst.ls[[i]]) - h.mean)
        big.sum.mx <- big.sum.mx + crossprod(sHst.ls[[i]])
    }
    cov.mx <- big.sum.mx/tau
    sqrtXX <- 1/sqrt(diag(cov.mx))
    for (i in 1:tau) {
        sHst.ls[[i]] <- t(t(sHst.ls[[i]]) * sqrtXX)
    }
    sHst0.ls <- NULL
    if (!is.null(Hst0.ls)) {
        sHst0.ls <- list()
        for (i in 1:tau) {
            sHst0.ls[[i]] <- t((t(Hst0.ls[[i]]) - h.mean) * sqrtXX)
        }
    }
    ls.out <- list(sHst.ls = sHst.ls, sHst0.ls = sHst0.ls, h.mean = h.mean,
        h.sd = 1/sqrtXX)
    return(ls.out)
  }
```

---

stnd.Ht *Standardize Temporal Covariates*

Description

Standardize temporal covariates with respect to both the spacial and time dimensions

Usage

```
stnd.Ht(Ht, n)
```

Arguments

Ht	Temporal covariates (of supporting sites). A τ x p_t numeric matrix.
n	Number of sites. A positive scalar integer.

Value

A τ x p_t numeric matrix.

See Also

stnd.Hs, stnd.Hst.ls.

Examples

```
##### Please see the examples in Hst.sumup


## The function is currently defined as
function (Ht, n)
{
    h.mean <- apply(Ht, 2, mean)
    sHt <- t(t(Ht) - h.mean)
    sHt <- t(t(sHt)/apply(sHt, 2, function(x) {
        sqrt(sum(x^2))
    }))
    sHt <- sHt * sqrt(nrow(Ht)/n)
    return(sHt)
  }
```

`subsetsites.Hst.ls` *Site-Wise Extract Space-Time Covariates*

Description

Extract space-time covariates by site

Usage

```
subsetsites.Hst.ls(Hst.ls, xmask)
```

Arguments

`Hst.ls`	Space-time covariates. A list of length τ, each element should be a n x $p_s t$ numeric matrix.
`xmask`	Which sites to remove from `Hst.ls`. A boolean vector of length n, or a vector of spacial indices.

Value

Space-time covariates. A list of length τ, each element a c x $p_s t$ numeric matrix, where c is the number of `TRUE`'s in boolean `xmask`, or length of index `xmask`.

Examples

```

tau <- 70
n <- 28

Hst.ls <- list()
for(i in 1:tau) { Hst.ls[[i]] <- matrix(rnorm(n*4), nrow=n) }

subsetsites.Hst.ls(Hst.ls, c(1,3,10))


subsetsites.Hst.ls(Hst.ls, c(TRUE, TRUE, rep(FALSE, n-2)))


## The function is currently defined as
function (Hst.ls, xmask)
{
    tau <- length(Hst.ls)
    Hst.ls.out <- list()
    for (i in 1:tau) {
        Hst.ls.out[[i]] <- Hst.ls[[i]][xmask, , drop = FALSE]
    }
    return(Hst.ls.out)
  }
```

unif.mh *Local Search Function*

Description

Search function

Usage

```
unif.mh(n, center, sd)
```

Arguments

n	Sample size. A positive scalar integer.
center	Mean. A numeric scalar (or vector).
sd	Standard deviation. A numeric scalar (or vector).

Value

A numeric vector of length n.

See Also

dlog.norm, MSS.snow.

Examples

```

x <- unif.mh(100, 1, 1)
hist(x)

## The function is currently defined as
function (n, center, sd)
{
    w <- sd * sqrt(3)
    a <- center - w
    b <- center + w
    x <- runif(n, a, b)
    return(x)
  }
```

`unload.Hst.ls` *Convert a Space-Time Covariate into Data*

Description

Convert a spacio-temporal covariate into contemporaneous data

Usage

```
unload.Hst.ls(Hst.ls, which.col, rgr.lags)
```

Arguments

`Hst.ls`	Space-time covariates. A list of length τ, each element should be a n x $p_s t$ numeric matrix.
`which.col`	Which column of `Hst.ls[[i]]` to insert into the ith row of `Z`. A scalar positive integer.
`rgr.lags`	Temporal lagging of `Z`. A scalar integer.

Value

A numeric τ x n matrix.

See Also

`load.Hst.ls.Z`, `load.Hst.ls.2Zs`.

Examples

```
###### here's an itty-bitty example

tau <- 7
n <- 5

Hst.ls <- list()
for(i in 1:tau) { Hst.ls[[i]] <- matrix(rnorm(n*4), nrow=n) }

Zh <- unload.Hst.ls(Hst.ls, 1, 0)


## The function is currently defined as
function (Hst.ls, which.col, rgr.lags)
{
    n <- nrow(Hst.ls[[1]])
    tau <- length(Hst.ls)
    Z.out <- matrix(NA, tau, n)
    min.ndx <- max(1, -min(rgr.lags) + 1)
    max.ndx <- min(tau, tau - max(rgr.lags))
    for (i in min.ndx:max.ndx) {
        Z.out[i - rgr.lags, ] <- Hst.ls[[i]][, which.col]
    }
    return(Z.out)
  }
```

`widals.predict` *WIDALS Interpolation*

Description

Interpolate to unmonitored sites using WIDALS

Usage

```
widals.predict(Z, Hs, Ht, Hst.ls, locs, lags, b.lag, Hs0, Hst0.ls, locs0,
geodesic = FALSE, wrap.around = NULL, GP, stnd.d = FALSE, ltco = -16)
```

Arguments

`Z`	Space-time data. A τ x n numeric matrix.
`Hs`	Spacial covariates (of supporting sites). An n x p_s numeric matrix.
`Ht`	Temporal covariates (of supporting sites). A τ x p_t numeric matrix.
`Hst.ls`	Space-time covariates (of supporting sites). A list of length τ, each element should be a n x p_st numeric matrix.
`locs`	Locations of supporting sites. An n x 2 numeric matrix, first column is spacial x, second column is spacial y. If the `geodesic` is `TRUE`, make sure latitude is in the first column.
`lags`	Temporal lags for stochastic smoothing. An integer vector or scalar. E.g., if the data's time increment is daily, then `lags = c(-1,0,1)` would tell the enclosed function `crispify` smooth today's predictions using yesterdays, today's, and tomorrow's observed residuals.
`b.lag`	ALS lag. A scalar integer, typically -1 (*a-prior*), or 0 (*a-posteriori*).
`Hs0`	Spacial covariates (of interpolation sites). An n* x p_s matrix, or `NULL`.
`Hst0.ls`	Space-time covariates (of interpolation sites). A list of length τ, each element should be a numeric n x p_st matrix.
`locs0`	Locations of interpolation sites. An *n** x 2 numeric matrix. See `locs` argument above.
`geodesic`	Use geodesic distance? Boolean. If true, distance (used internally) is in units kilometers.
`wrap.around`	**Unused.
`GP`	Widals hyperparameters. A non-negative vector.
`stnd.d`	Spacial compression. Boolean.
`ltco`	Weight threshold. A scalar number. A value of, e.g., -10, will instruct `crispify` to ignore weights less than 10^(-10) when smoothing.

Value

A τ x n^* matrix. The WIDALS predictions at `locs0`.

See Also

`crispify`, `H.als.b`, `widals.snow`.

Examples

```
#### similar to example provided in H.als.b.

set.seed(99999)


library(SSsimple)

tau <- 70
n.all <- 14

Hs.all <- matrix(rnorm(n.all), nrow=n.all)
Ht <- matrix(rnorm(tau*2), nrow=tau)
Hst.ls.all <- list()
for(i in 1:tau) { Hst.ls.all[[i]] <- matrix(rnorm(n.all*2), nrow=n.all) }

Hst.combined <- list()
for(i in 1:tau) {
    Hst.combined[[i]] <- cbind( Hs.all, matrix(Ht[i, ], nrow=n.all, ncol=ncol(Ht),
    byrow=TRUE), Hst.ls.all[[i]] )
}

locs.all <- cbind(runif(n.all, -1, 1), runif(n.all, -1, 1))
D.mx.all <- distance(locs.all, locs.all, FALSE)
R.all <- exp(-2*D.mx.all) + diag(0.01, n.all)

######## use SSsimple to simulate
sssim.obj <- SS.sim.tv( 0.999, Hst.combined, 0.01, R.all, tau )


ndx.support <- 1:10
ndx.interp <- 11:14

locs <- locs.all[ndx.support, ]
locs0 <- locs.all[ndx.interp, ]

Z.all <- sssim.obj$Z
Z <- Z.all[ , ndx.support]
Z0 <- Z.all[ , ndx.interp]

Hst.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.support)
Hst0.ls <- subsetsites.Hst.ls(Hst.ls.all, ndx.interp)
```

```
Hs <- Hs.all[ ndx.support, , drop=FALSE]
Hs0 <- Hs.all[ ndx.interp, , drop=FALSE]

test.rng <- 20:tau


################## use ALS
xrho <- 1/10
xreg <- 1/10
xALS <- H.als.b(Z=Z, Hs=Hs, Ht=Ht, Hst.ls=Hst.ls, rho=xrho, reg=xreg,
b.lag=-1, Hs0=Hs0, Ht0=Ht, Hst0.ls=Hst0.ls)

errs.sq <- (Z0 - xALS$Z0.hat)^2
sqrt( mean(errs.sq[test.rng, ]) )

################## now use WIDALS

GP <- c(1/10, 1/10, 2, 0, 1)
Zwid <- widals.predict(Z=Z, Hs=Hs, Ht=Ht, Hst.ls=Hst.ls, locs=locs, lags=c(0),
b.lag=-1, Hs0=Hs0, Hst0.ls=Hst0.ls, locs0=locs0, FALSE, NULL, GP)

errs.sq <- (Z0 - Zwid)^2
sqrt( mean(errs.sq[test.rng, ]) )
```

widals.snow *Fit WIDALS*

Description

Locate the WIDALS hyperparameters

Usage

```
widals.snow(j, rm.ndx, Z, Hs, Ht, Hst.ls, locs, lags, b.lag, cv = 0,
geodesic = FALSE, wrap.around = NULL, GP.mx, stnd.d = FALSE, ltco = -16)
```

Arguments

j	Index used by snowfall. A scalar integer. Which row of GP.mx to use for the ALS hyperparameters, GP.
rm.ndx	A list of vectors of indices to remove for *k*-fold cross-validation.
Z	Data. A τ x n numeric matrix.
Hs	Spacial covariates. An n x p_s numeric matrix.
Ht	Temporal covariates. A τ x p_t numeric matrix.
Hst.ls	Space-time covariates. A list of length τ, each element containing a n x $p_s t$ numeric matrix.

`locs`	Locations of supporting sites. An *n* x 2 numeric matrix, first column is spacial x, second column is spacial y. If the `geodesic` is `TRUE`, make sure latitude is in the first column.
`lags`	Temporal lags for stochastic smoothing. An integer vector or scalar. E.g., if the data's time increment is daily, then `lags = c(-1,0,1)` would tell the enclosed function `crispify` smooth today's predictions using yesterdays, today's, and tomorrow's observed residuals.
`b.lag`	ALS lag. A scalar integer, typically -1 (*a-prior*), or 0 (*a-posteriori*).
`cv`	Cross-validation switch. Currently takes on a value of `-2` or `2`. See Details below.
`geodesic`	Use geodesic distance? Boolean. If true, distance (used internally) is in units kilometers.
`wrap.around`	**Unused.
`GP.mx`	Hyperparameters. A $k.glob$ x 2 non-negative matrix. See `MSS.snow`.
`stnd.d`	Spacial compression. Boolean.
`ltco`	Weight threshold. A scalar number. A value of, e.g., -10, will instruct `crispify` to ignore weights less than 10^(-10) when smoothing.

Details

When the `cv` is set to 2, then this function uses spacial *k*-fold validation, according to the site indices present in `rm.ndx`. When `cv` is set to -2, self-referencing sites are given zero-weight, i.e., a site's value is not allowed to contribute to its predicted value.

Value

A τ x n matrix. The WIDALS predictions at `locs`.

See Also

`crispify`, `H.als.b`, `widals.predict`.

Examples

```
set.seed(99999)

library(SSsimple)

tau <- 100
n.all <- 35

Hs.all <- matrix(rnorm(n.all), nrow=n.all)
Ht <- matrix(rnorm(tau*2), nrow=tau)
Hst.ls.all <- list()
for(i in 1:tau) { Hst.ls.all[[i]] <- matrix(rnorm(n.all*2), nrow=n.all) }

Hst.combined <- list()
```

```
for(i in 1:tau) {
    Hst.combined[[i]] <- cbind( Hs.all, matrix(Ht[i, ], nrow=n.all, ncol=ncol(Ht),
    byrow=TRUE), Hst.ls.all[[i]] )
}

locs.all <- cbind(runif(n.all, -1, 1), runif(n.all, -1, 1))
D.mx.all <- distance(locs.all, locs.all, FALSE)
R.all <- exp(-2*D.mx.all) + diag(0.01, n.all)

######## use SSsimple to simulate
sssim.obj <- SS.sim.tv( 0.999, Hst.combined, 0.01, R.all, tau )


n <- n.all
locs <- locs.all

Z.all <- sssim.obj$Z
Z <- Z.all


Hst.ls <- Hst.ls.all
Hs <- Hs.all

test.rng <- 20:tau

################  WIDALS, true cross-validation

rm.ndx <- create.rm.ndx.ls(n, 10)

cv <- 2
lags <- c(0)
b.lag <- 0

GP <- c(1/8, 1/12, 5, 0, 1)
GP.mx <- matrix(GP, ncol=length(GP))
Zwid <- widals.snow(j=1, rm.ndx, Z, Hs, Ht, Hst.ls, locs, lags, b.lag, cv = cv,
geodesic = FALSE, wrap.around = NULL, GP.mx, stnd.d = FALSE, ltco = -16)

errs.sq <- (Z - Zwid)^2
sqrt( mean(errs.sq[test.rng, ]) )


################  WIDALS, pseudo cross-validation

rm.ndx <- I(1:n)

cv <- -2
lags <- c(0)
b.lag <- -1

GP <- c(1/8, 1/12, 5, 0, 1)
GP.mx <- matrix(GP, ncol=length(GP))
Zwid <- widals.snow(j=1, rm.ndx, Z, Hs, Ht, Hst.ls, locs, lags, b.lag, cv = cv,
geodesic = FALSE, wrap.around = NULL, GP.mx, stnd.d = FALSE, ltco = -16)

errs.sq <- (Z - Zwid)^2
sqrt( mean(errs.sq[test.rng, ]) )
```

`Z.clean.up` *Clean Data*

Description

A crude, brute-force way to destroy bad values in data.

Usage

```
Z.clean.up(Z)
```

Arguments

`Z`	Data. A τ x n matrix.

Details

This function replaces intractable values, e.g., `NA`, or `-Inf`, in data, with the global mean.

Value

A τ x n numeric matrix.

Examples

```
tau <- 10
n <- 7

Z <- matrix(1, tau, n)
Z[2,4] <- -Inf
Z[3,4] <- Inf
Z[4,4] <- NA
Z[5,4] <- log(-1)
Z

Z.clean.up(Z)




## The function is currently defined as
```

```
function (Z)
{
    Z[Z == Inf | Z == -Inf] <- NA
    Z[is.na(Z) | is.nan(Z)] <- mean(Z, na.rm = TRUE)
    return(Z)
  }
```

Index