Package 'hero'

July 22, 2025

Type Package

Title Spatio-Temporal (Hero) Sandwich Smoother

Version 0.6

Author Joshua French

Maintainer Joshua French <joshua.french@ucdenver.edu>

Description An implementation of the sandwich smoother proposed in Fast Bivariate Penalized Splines by Xiao et al. (2012) <doi:10.1111/rssb.12007>. A hero is a specific type of sandwich. Dictionary.com (2018) <https://www.dictionary.com> describes a hero as: a large sandwich, usually consisting of a small loaf of bread or long roll cut in half lengthwise and containing a variety of ingredients, as meat, cheese, lettuce, and tomatoes. Also implements the spatio-temporal sandwich smoother of French and Kokoszka (2021) <doi:10.1016/j.spasta.2020.100413>.

Depends R (>= 2.10)

Imports Matrix, splines, optimx, pbapply, sf, sp, fields

Suggests autoimage, devtools, fda, igraph, testthat, future.apply, Rmpi

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-07-15 21:10:09 UTC

Contents

adjacent *Determine adjacent points*

Description

adjacent attempts to find the point(s) adjacent (closest) to each point. The data are implicitly assumed to be on a grid, otherwise this function isn't very useful. Distances between each point and other points in coords are computed and then rounded using the round function. Let k denote the minimum distance between a reference point and all other points. A point is adjacent to the reference point if (after rounding), it's distance from the reference point matches the minimum distance k.

Usage

```
adjacent(coords, longlat = FALSE, digits = 1)
```

Arguments

coords	A two-dimensional matrix-like object with non-NULL dimensions.
longlat	A logical value indicating whether Great Circle distances should be used (TRUE) or Euclidean distances (FALSE). The default is FALSE.
digits	The number of digits to use when applying round to the distances.

Details

digits is the number of digits used by round in the rounding process.

Value

A hero_adjacent object. This is simply a list with elements nbrs and coords. nbrs is a list specifying the adjacent points for each point. coords is simply the original coords supplied to the function and is retained for plotting purposes.

Examples

```
# basic coordinates
coords = expand.grid(1:4, 1:4)
# plot coordinates to see relationships
plot(coords, type = "n")
text(coords)
a = adjacent(coords, digits = 1)
plot(a)
```

as.starray *Convert array to* starray

Description

Convert a three-dimensional spatio-temporal array into an starray object. The first two dimensions are assumed to relate to gridded spatial positions.

Usage

```
as.starray(x)

starray(x)

as_starray(x)
```

Arguments

x	A three-dimensional array

Value

An starray object.

Examples

```
star = as.starray(tasmax)
class(star)
```

as.sts *Convert object to* sts *class*

Description

Convert a numeric three-dimensional array or two-dimensional matrix-like object to an sts (spatial time series) object. If x is a three-dimensional array, the first two dimensions are assumed to relate to gridded spatial positions. If x has only two dimensions, each row is a time series for a specific location. Each column is a realization of a geostatistical process at a specific time.

Usage

```
as.sts(x)

sts(x)

as_sts(x)
```

Arguments

x	A matrix-like object with 2 dimensions or an array with 3 dimensions.

Details

This method has been tested with objects of class matrix, data.frame, array, and Matrix-class. It should be possible for x to have a different class as long as the object has a loaded as.matrix method, which is used in this function.

Value

An sts object.

Examples

```
# 3d array to sts
sts = as.sts(tasmax)
class(sts)

# extract a subset of tasmax to produce an sts
x = matrix(c(tasmax[50:60, 50:60, ]), ncol = 30)
sts = as.sts(x)
class(sts)

sts = as.sts(as.array(x))
class(sts)

sts = as.sts(Matrix::Matrix(x))
class(sts)

sts = as.sts(as.data.frame(x))
class(sts)
```

---

assemble *Assemble spline ingredients for sandwich smooth*

Description

Assemble computations from a spline-related object x in order to implement the sandwich smoother. This is essentially an internal function, but could be useful for performing a manual implementation of the sandwich smoother.

Usage

```
assemble(object, ...)

## S3 method for class 'hero_bspline'
assemble(object, x, m = 2, sparse = TRUE, ...)
```

```
## S3 method for class 'hero_radspline'
assemble(object, x, m = 2, sparse = TRUE, spdiffpen = TRUE, digits = 1, ...)

## S3 method for class 'list'
assemble(object, x, m = 2, sparse = TRUE, spdiffpen = TRUE, digits = 1, ...)
```

Arguments

object	A spline-related object (e.g, a hero_bspline from the bspline function), or a list of spline-related objects.
...	Not implemented
x	Values at which to evaluate the basis functions. This should be a numeric vector if object is a hero_bspline. This should be a numeric matrix of coordinates if object is a hero_radspline. If object is a list comprised of hero_bspline and hero_radspline objects, then x should be a list where each element of the list corresponds to the appropriate x argument for each element.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
spdiffpen	A logical value indicating whether spdiffpen should be used to compute the difference penalty. The default is FALSE.
digits	The number of digits to use when applying round to the distances.

Value

A list with the necessary components (ingredients)

Examples

```
# construct b-spline
object1 = bspline(nbasis = 10)
# sequence to evaluate
x1 = seq(0, 1, len = 11)
# assemble b-spline information
spline1 = assemble(object1, x1)

# assemble radial spline information
border = border.grid(lon, lat)
object2 = radspline(nknots = 16, border)
x2 = cbind(c(lon), c(lat))
spline2 = assemble(object2, x = x2)

# assemble for list of splines
object = list(object1, object2)
x = list(x1, x2)
splines = assemble(object, x)
```

`border.grid` *Construct border for grid*

Description

`border.grid` determines the border for data on a grid. `x` and `y` must define a regular or irregular grid. See Details.

Usage

```
border.grid(x, y, proj4string)

border_grid(x, y, proj4string)

borderGrid(x, y, proj4string)

BorderGrid(x, y, proj4string)
```

Arguments

`x`	A vector or matrix of x coordinates. See Details.
`y`	A vector or matrix of y coordinates. See Details.
`proj4string`	A projection string of class `CRS-class`. If not provided, then default values are used. This should be changed with caution.

Details

A regular grid is defined by ascending numeric vectors `x` and `y`. A vector `x` is ascending if `x[i] < x[j]` for `i < j`.

An irregular grid is defind by ascending matrices. A matrix `x` is ascending if `x[i, j] < x[i, l]` for `j < l` and if `x[i, j] < x[k, j]` and `j < k`.

Value

A `SpatialPolygons` object.

Author(s)

Joshua French

Examples

```
# create x and y defining square border
x = seq(min(lon), max(lon), length = 60)
y = seq(min(lat), max(lat), length = 80)
border = border.grid(x, y)
sp::plot(border)
```

```
# use lon and lat to define border of an irregular grid
border2 = border.grid(lon, lat)
sp::plot(border2)
```

---

bspline *B-spline specification*

Description

bspline helps define the parameters necessary for constructing a B-spline but doesn't evaluate it.

Usage

```
bspline(rangeval = 0:1, nbasis, nknots, norder = 4, extend = FALSE, knots)
```

Arguments

rangeval	A numeric vector of length 2 defining the interval over which the functional data object can be evaulated. The default value is 0:1.
nbasis	An integer specifying the number of basis functions to construct. This is closely linked to the number of knots (nknots), and nknots = nbasis - norder.
nknots	The number of *interior* knots. See Details.
norder	An integer specifying the order of the B-splines, which is one higher than their degree. The default is 4, which corresponds to cubic splines.
extend	Should the knots stop at the endpoints specified by rangeval? Default is FALSE. See Details.
knots	A numeric vector with all knots (interior and exterior), including potentially replicated endpoints. See Details.

Details

The knots are assumed to be equidistant and non-repeating (except possibly at the endpoints).

The number of knots (nknots) and the number of basis function (nbasis) are linked by the relation nknots = nbasis - norder.

If extend = TRUE, the interior knots are augmented by replicating the rangeval endpoints norder times. Otherwise, the interior knots are augmented by norder knots at each end corresponding to the spacing of the interior knots.

The knot placement mimics the behavior of create.bspline.basis when extend = FALSE. Note that the number of breaks specified by breaks in create.bspline.basis corresponds to the number of interior knots plus 2 (the interior knots plus the two endpoints).

If knots is specified, the function does minimial argument checking. This is provided (mostly) for testing purposes, though it can be used by individuals who want more customizability of knots locations than the equidistant spacing provided by default.

Value

An object of class `hero_bspline`. It is a list specifying the necessary B-spline parameters.

Author(s)

Joshua French

See Also

`knot.design`

Examples

```
bspline(nbasis = 10)
```

`circulate` *Circulate values of a vector*

Description

The first `n` values of `x` are circulated from the front of `x` to the back of `x`.

Usage

```
circulate(x, n = 1)
```

Arguments

`x`	vector of values
`n`	The number of values to circulate

Value

The circulated vector

Author(s)

Joshua French

Examples

```
circulate(1:10, n = 2)
circulate(as.list(1:10), n = 2)
```

connect *Connect* hero_radsplines

Description

connect joins multiple hero_radspline objects into a single hero_radspline. The e

Usage

```
connect(...)
```

Arguments

...	A sequence of hero_radspline objects from the radspline function.

Value

A combined hero_radspline

See Also

radspline

Examples

```
border = border.grid(lon, lat)
s1 = radspline(nknots = 36, border = border)
plot(s1)
s2 = radspline(nknots = 36 * 4, border = border,
               width = 6)
plot(s2)
par(mfrow = c(1, 2))
plot(s1)
plot(s2)
par(mfrow = c(1, 1))
s = connect(s1, s2)
plot(s)
```

create.prepared_list *Manually create a* prepared_list

Description

create.prepared_list creates a prepared_list manually. Typically, one would simply use the prepare.list, but there are situations where the data argument would be too large to read into memory.
This function assumes that the user has used the assemble function to construct a list of the relevant assembled_splines and manually computed Ytilde for a number of relevant data observations and stored them in a list. The user should also manually compute the sum of the squared data for each data observation. The user must also specify the dimensions of each data set (which are assumed to be the same) as a vector and provide the relevant set of values at which each data object is observed. See Examples.

Usage

```
create.prepared_list(assembled, x, Ytilde, sum_ysq, n)
```

Arguments

assembled	A list of assembled_splines. See Examples.
x	The list of arguments at which to evaluate each of the splines used to construct assembled.
Ytilde	A list of prepared_* objects.
sum_ysq	A vector with the sum of squared data objects used to construct Ytilde.
n	The dimensions of the data objects used to construct Ytilde.

Value

A prepared list.

Examples

```
# generate and prepare 3d data
set.seed(9)
dat = generate.data3d()

# list giving the locations to evaluate the basis functions
x = dat$x
# construct a set of basic B-splines for each dimension
splines = default.splines(x)

# construct assembled splines from splines list
a = assemble(splines, x)

# imagine there are 4 data obsevations we want to smooth
```

```
# but that they can't be loaded into memory
Ytilde = vector("list", 4)
sum_ysq = numeric(4)

# prepare each data set manually
# notice the use of the assembled arguments so that
# the splines are not "assembled" again for each data set
for(i in seq_along(Ytilde)) {
    data = generate.data3d()$data3d
    Ytilde[[i]] = prepare(data, x = x, splines = splines,
                          assembled = a)
    sum_ysq[i] = sum(data^2)
}
n = dim(data)
p = create.prepared_list(assembled = a, x = x,
                         Ytilde = Ytilde, sum_ysq = sum_ysq,
                         n = n)
```

`default.evalargs` *Construct default* `evalargs`

Description

Create a default `evalargs` object based on `data`. This is just a list of sequences. If `ni = dim(data)[i]`, then the sequence for dimension i is `seq(0, 1, len = ni)`.

Usage

```
default.evalargs(data)
```

Arguments

`data`	A matrix or array-like object

Value

A list of equidistance sequences between 0 and 1

Author(s)

Joshua French

Examples

```
a = array(rnorm(10 * 11 * 12), dim = 10:12)
default.evalargs(a)
```

`default.splines` *Construct default splines*

Description

Construct a list of `hero_bsplines` using the default values suggested by Ruppert, Wand, and Carroll (2003). Specifically, if `r = range(evalargs[[i]])` and `l = length(evalargs[[i]])`, then Ruppert, Wand, and Carroll (2003) suggest `nknots = min(ceiling(l/4), 35)` and the function returns the `hero_bspline` for that dimension as `bspline(r, nknots = nknots)`.

Usage

```
default.splines(evalargs)
```

Arguments

`evalargs`	A list of equidistant sequences.

Value

A list of `hero_bsplines`.

Author(s)

Joshua French

References

Ruppert, D., Wand, M. P., & Carroll, R. J. (2003). Semiparametric Regression. Cambridge University Press. <doi:10.1017/CBO9780511755453>

Examples

```
s1 = seq(0, 1, len = 10)
s2 = seq(0, 1, len = 20)
default.splines(list(s1, s2))
```

diffpen *P-spline difference penalty*

Description

P-spline difference penalty

Usage

```
diffpen(x, m = 2, sparse = TRUE)
```

Arguments

x	A hero_bspline object produced by bspline.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.

Value

A matrix or sparseMatrix-class object.

Author(s)

Joshua French

Examples

```
b = bspline(nbasis = 10)
diffpen(b)
```

enhance *Enhance penalty value*

Description

enhance enhances the sandwich smoother by choosing the optimal penalty value that minimizes the GCV statistic. The optimx function is used to do the optimization.

Usage

```
enhance(
  obj,
  par = rep(0, length(obj$n)),
  lower = rep(-20, length(par)),
  upper = rep(20, length(par)),
  method = "L-BFGS-B",
  control = list(),
  prepare = TRUE,
  loggcv = FALSE,
  ...
)
```

Arguments

obj	A prepared_* object from a prepare function.
par	a vector of initial values for the parameters for which optimal values are to be found. Names on the elements of this vector are preserved and used in the results data frame.
lower, upper	Bounds on the variables for methods such as "L-BFGS-B" that can handle box (or bounds) constraints.
method	The method to be used for optimization. The default is L-BFGS-B, which allows for constraints on the parameters to optimize. See optimx for all available methods.
control	A list of control parameters. See 'Details'.
prepare	A logical value. The default is TRUE, indicating that a prepared_data object should be returned. If FALSE, then the results of the call to the optimx function is returned.
loggcv	A logical value indicating whether the log of the GCV statistic should be used. Useful for very large data sets. Default is TRUE.
...	Additional arguments to pass to to the optimx function.

Details

Internally, the loglambda2gcv is used as the objective function for the optimx function. Many different optimization methods are available. The default is L-BFGS-B, which allows for constraints on the parameters to optimize. Another excellent choice is the nlminb algorithm, which also allows for parameter constraints.

Value

By default, a prepared_data object with the optimal loglambda values that minimize the GCV, along with an additional component, results, that contains the optimization results.

Author(s)

Joshua French

Examples

```
# create b-splines
x1 = bspline(nbasis = 10)
x2 = bspline(nbasis = 12)

# observed data locations
evalarg1 = seq(0, 1, len = 60)
evalarg2 = seq(0, 1, len = 80)

# construct "true" data
mu = matrix(0, nrow = 60, ncol = 80)
for(i in seq_len(60)) {
   for(j in seq_len(80)) {
      mu[i, j] =  sin(2*pi*(evalarg1[i]-.5)^3)*cos(4*pi*evalarg2[j])
   }
}
# construct noisy data
data = mu + rnorm(60 * 80)

obj = prepare(data, list(evalarg1, evalarg2), list(x1, x2))
enhance(obj)
```

`enhance.grid` *Enhance penalty value using grid search*

Description

`enhance.grid` enhances the sandwich smoother by choosing a optimal penalty value to lower the GCV statistic. A grid search algorithm is utilized based on the each row of `par`. The penalty values (assumed to be on the log scale) are passed to the `loglambda2gcv` function. If `prepare` is `TRUE`, then `obj` is returned with the penalty values that minimize the GCV statistic during the grid search. Otherwise, the complete results of the grid search are returned.

Usage

```
enhance.grid(obj, par, prepare = TRUE, loggcv = FALSE, ..., cl = NULL)
```

Arguments

`obj`	A `prepared_*` object from a `prepare` function.
`par`	A matrix-like object (i.e., `!is.null(dim(par))`). Each row contains a set of parameter values for which the GCV statistic should be computed. The number of columns of `par` should match the dimensionality of `obj`, i.e, should equal `length(obj)$n`. If missing, the default choices are a row of -20s, a row of 0s, and a row of 20s.
`prepare`	A logical value. The default is `TRUE`, indicating that a `prepared_data` object should be returned. If `FALSE`, then the results of the call to the `optimx` function is returned.

loggcv	A logical value indicating whether the log of the GCV statistic should be used. Useful for very large data sets. Default is TRUE.
...	Additional arguments to pass to to the loglambda2gcv function.
cl	A cluster object created by makeCluster, or an integer to indicate number of child-processes (integer values are ignored on Windows) for parallel evaluations (see Details on performance). It can also be "future" to use a future backend (see Details), NULL (default) refers to sequential evaluation.

Value

By default, a prepared_* object with the optimal loglambda values that minimize the GCV, along with an additional component, results, that contains the optimization results. Otherwise, the complete results of the grid search.

Author(s)

Joshua French

Examples

```
# create b-splines
b1 = bspline(nbasis = 10)
b2 = bspline(nbasis = 12)

# observed data locations
x1 = seq(0, 1, len = 60)
x2 = seq(0, 1, len = 80)

# construct "true" data
mu = matrix(0, nrow = 60, ncol = 80)
for(i in seq_len(60)) {
   for(j in seq_len(80)) {
      mu[i, j] =  sin(2*pi*(x1[i]-.5)^3)*cos(4*pi*x2[j])
   }
}
# construct noisy data
data = mu + rnorm(60 * 80)

obj = prepare(data, list(x1, x2), list(b1, b2))
enhance.grid(obj, prepare = FALSE)
```

---

enlarge *Enlarge spatial domain*

Description

Enlarge the spatial domain of a SpatialPolygons-class object. If width isn't specified, then 10% of the maximum distance between the points specified by the bounding box is used. The st_buffer function is used to enlarge x.

Usage

```
enlarge(x, width, ...)
```

Arguments

x	A SpatialPolygons-class object defining the border(s) of the spatial domain.
width	The width to enlarge the study area. Distance from original geometry to include in the new geometry. Negative values are allowed.
...	Additional arguments to pass to st_buffer.

Value

An object of class hero_enlarge. This is simply a list with eborder (the enlarged border), border (the border of the original coordinates), and the width of the enlargement. eborder and border are SpatialPolygons-class objects.

Author(s)

Joshua French

Examples

```
# enlarge regular grid
# create x and y defining square border
x = seq(min(lon), max(lon), length = 60)
y = seq(min(lat), max(lat), length = 80)
border = border.grid(x, y)
e = enlarge(border)
plot(e)

# create x and y defininging an irregular grid
border2 = border.grid(lon, lat)
e2 = enlarge(border2)
plot(e2)
```

---

generate.data2d *Generate 2d data*

Description

Generate two-dimensional data related to the f1 function of Lu et al. (2012) (code from author). Define n = c(60, 80). Then x[[i]] = (1:n[i])/n[i] -1/2/n[i]. These are the observed data locations. For i and j spanning the full length of each element of x, mu2d[i, j] = sin(2 * pi * (x[[1]][i] - .5) ^ 3) * cos(4 * pi * x[[2]][j]). Lastly, data2d = mu2d + rnorm(prod(n)).

Usage

```
generate.data2d()

generate_data2d()

generateData2d()

GenerateData2d()
```

Value

A list with components `x`, `mu2d`, and `data2d`. `x` is a list of sequences with length 60 and 80. `mu2d` and `data2d` are matrices of size 60 by 80.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

Examples

```
dat = generate.data2d()
```

`generate.data3d` *Generate 3d data*

Description

Generate data related to Section 7.2 of Lu et al. (2012) (code from author). Define `n = c(128, 128, 24)`. Then `x[[i]] = (1:n[i])/n[i] - 1/2/n[i]`. These are the observed data locations. For `i`, `j`, `k` spanning the full length of each element of `x`, `mu3d[i, j, k] = x[[1]][i]^2 + x[[2]][j]^2 + x[[3]][k]^2`. Lastly, `data3d = mu3d + 0.5 * rnorm(n[1] * n[2] * n[3])`.

Usage

```
generate.data3d()

generate_data3d()

generateData3d()

GenerateData3d()
```

Value

A list with components x, mu3d, and data3d. x is a list of sequences with length 128, 128, and 24. mu3d and data3d are arrays of size 128 by 128 by 24.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

Examples

```
dat = generate.data3d()
```

---

hero *Construct a hero sandwich smoother*

Description

hero constructs a hero sandwich smoother based off off a prepared data object coming from the prepare function.
Subclasses are added (e.g., hero_numeric, hero_matrix, hero_array, etc.) are added to the returned object for plotting purposes.
A list is returned (and the data locations are not) for hero.prepared_list. Each element of the list contains the coefficients and fitted values (if fitted is TRUE) for the respective data observation.

Usage

```
hero(x, ...)

## S3 method for class 'prepared_array'
hero(x, ...)

## S3 method for class 'prepared_list'
hero(x, ..., fitted = FALSE)

## S3 method for class 'prepared_matrix'
hero(x, ...)

## S3 method for class 'prepared_numeric'
hero(x, ...)

## S3 method for class 'prepared_sequential'
hero(
```

```
  x,
  ...,
  export_list,
  export_fun = base::saveRDS,
  package = "base",
  call_args = list()
)

## S3 method for class 'prepared_starray'
hero(x, ...)

## S3 method for class 'prepared_sts'
hero(x, ...)
```

Arguments

x	Data prepared via the prepare function.
...	Mostly not implemented. hero.prepared_list takes the fitted argument, specifying whether the fitted values should be returned.
fitted	A logical value indicating whether the fitted values should be computed. The default is FALSE.
export_list	A vector or list whose elements tell export_fun what files to export. The length must match the number of observations, i.e., the number of elements in x$Ytilde
export_fun	A function that will write the results for each observation to file using the names in export_list. Must only have the arguments object, which is what will be saved and computed internally, and file, which is the name of the file that will be saved. file will be one of the elements of export_list.
package	A character string indicating the approach to use for the computations. The choices are "base", "parallel", "pbapply", "future.apply", or "Rmpi". The default is "base". If package == "base", then mapply is used. If package == "parallel", then mcmapply is used. If package == "pbapply", then pblapply is used. If codepackage == "future.apply", then future_mapply is used. If codepackage == "Rmpi", then mpi.applyLB is used.
call_args	A named list providing relevant arguments to mcmapply, pblapply, future_mapply, or mpi.applyLB, depending on the package choice.

Value

A hero object with the smoothed data (fitted), the estimated coefficients for the basis functions (coefficients), and the locations of the original data (x).

Author(s)

Joshua French.

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

French, Joshua P., and Piotr S. Kokoszka. "A sandwich smoother for spatio-temporal functional data." Spatial Statistics 42 (2021): 100413.

Examples

```
# create b-splines
x1 = bspline(nbasis = 10)
x2 = bspline(nbasis = 12)

# observed data locations
evalarg1 = seq(0, 1, len = 60)
evalarg2 = seq(0, 1, len = 80)

# construct "true" data
mu = matrix(0, nrow = 60, ncol = 80)
for(i in seq_len(60)) {
   for(j in seq_len(80)) {
      mu[i, j] =  sin(2*pi*(evalarg1[i]-.5)^3)*cos(4*pi*evalarg2[j])
   }
}
# construct noisy data
data = mu + rnorm(60 * 80)

obj = prepare(data, list(evalarg1, evalarg2), list(x1, x2))
obj = enhance(obj)
sandmod = hero(obj)
plot(sandmod)
```

`knot.design` *Design knot/breakpoint spacing*

Description

See Details of `bspline` for additional information about arguments.

Usage

```
knot.design(
  rangeval = 0:1,
  nbasis,
  nknots,
  norder = 4,
  extend = FALSE,
  interior = FALSE
)
```

```
knot_design(
  rangeval = 0:1,
  nbasis,
  nknots,
  norder = 4,
  extend = FALSE,
  interior = FALSE
)

knotDesign(
  rangeval = 0:1,
  nbasis,
  nknots,
  norder = 4,
  extend = FALSE,
  interior = FALSE
)

KnotDesign(
  rangeval = 0:1,
  nbasis,
  nknots,
  norder = 4,
  extend = FALSE,
  interior = FALSE
)
```

Arguments

`rangeval`	A numeric vector of length 2 defining the interval over which the functional data object can be evaulated. The default value is `0:1`.
`nbasis`	An integer specifying the number of basis functions to construct. This is closely linked to the number of knots (`nknots`), and `nknots = nbasis - norder`.
`nknots`	The number of *interior* knots. See Details.
`norder`	An integer specifying the order of the B-splines, which is one higher than their degree. The default is 4, which corresponds to cubic splines.
`extend`	Should the knots stop at the endpoints specified by `rangeval`? Default is `FALSE`. See Details.
`interior`	A logical value specifying whether only interior knots should be returned. Default is `FALSE`.

Value

An ascending sequence of univarite knot locations.

Examples

```
if (requireNamespace("fda", quietly = TRUE)) {
b = fda::create.bspline.basis(nbasis = 10)
# interior knots only
bknots = b$params
# should match
knots = knot.design(nbasis = 10, interior = TRUE)
all.equal(bknots, knots)
}
```

`kronecker.seq` *A sequence of kronecker products*

Description

A sequence of kronecker products

Usage

```
kronecker.seq(X, FUN = "*", make.dimnames = FALSE, ...)

kronecker_seq(X, FUN = "*", make.dimnames = FALSE, ...)

kroneckerSeq(X, FUN = "*", make.dimnames = FALSE, ...)

KroneckerSeq(X, FUN = "*", make.dimnames = FALSE, ...)
```

Arguments

`X`	A list of numeric matrices or arrays
`FUN`	a function; it may be a quoted string.
`make.dimnames`	Provide dimnames that are the product of the dimnames of `X` and `Y`.
`...`	optional arguments to be passed to `FUN`.

Value

A matrix or array

Examples

```
x1 = matrix(rnorm(16), nrow = 4)
x2 = matrix(rnorm(25), nrow = 5)
x3 = matrix(rnorm(36), nrow = 6)
x4 = matrix(rnorm(49), nrow = 7)
p1 = x1 %x% x2 %x% x3 %x% x4
p2 = kronecker.seq(list(x1, x2, x3, x4))
all.equal(p1, p2)
```

loglambda2gcv *Determine GCV statistic*

Description

loglambda2gcv uses a vector of penalty values to evaluate the GCV statistic for a prepared_response object.

Usage

```
loglambda2gcv(loglambda, obj, loggcv = FALSE)
```

Arguments

loglambda	A vector of penalty values (assumed to be on a natural logarithmic scale) for computing the GCV.
obj	A prepared_* object from a prepare function.
loggcv	A logical value indicating whether the log of the GCV statistic should be returned. The default is FALSE.

Details

Though this function can be used by the user, it is basically an internal function used to find the value of loglambda minimizing the GCV statistic.

Value

The scalar GCV statistic

See Also

prepare

Examples

```
n1 = 10
b1 = bspline(nbasis = 10)
x1 = seq(0, 1, len = n1)
n2 = 20
x2 = seq(0, 1, len = n2)
b2 = bspline(nbasis = 12)
# construct "true" data
mu = matrix(0, nrow = n1, ncol = n2)
for(i in seq_len(n1)) {
   for(j in seq_len(n2)) {
      mu[i, j] =  sin(2*pi*(x1[i]-.5)^3)*cos(4*pi*x2[j])
   }
}
```

```
image(mu)
# construct noisy data
data = mu + rnorm(n1 * n2)
x = list(x1, x2)
splines = list(b1, b2)
obj = prepare(data, x, splines)
loglambda2gcv(c(0, 0), obj)
```

---

ludata *Data for f1 function from Lu et al. (2012)*

Description

Data related to the f1 function in Lu et al. (2012). Define `n1 = 60` and `x = seq_len(n1)/n1 - 1/2/n1`. Similarly, define `n2 = 80` and `z = seq_len(n2)/n2 - 1/2/n2`. The f1 function is defined as `sin(2 * pi * (x[i] - .5) ^ 3) * cos(4 * pi * z[j])`, where `i in seq_along(x)` and `j in seq_along(z)`. The result of this function is stored in `lutruef1`. Using `set.seed(3)` and adding `rnorm(60 * 80)` to `lutruef1` results in `lunoisyf1`.

Usage

```
data(ludata)
```

Format

The sequences `x` and `z`, the `lutruef1` data matrix with 60 rows and 80 columns, and the `lunoisyf1` data matrix with 60 rows and 80 columns.

---

plot.hero_adjacent *Plot a* hero_adjacent *object*

Description

Plot a `hero_adjacent` object. `x$nbrs` is used to construct a `sparseMatrix-class` object. The default behavior is to plot the sparse matrix using the `image` function. However, if the igraph package is installed, a graph is made using `graph_from_adjacency_matrix` and then plotted using `plot.igraph`.

Usage

```
## S3 method for class 'hero_adjacent'
plot(x, ...)
```

Arguments

`x`	A `hero_adjacent` object
`...`	Additional arguments passed to `image`, or if the igraph package is installed, `plot.igraph`.

Examples

```
coords = expand.grid(1:4, 1:4)
a = adjacent(coords, digits = 1)
plot(a)
```

`plot.hero_bspline` *Plot a* `hero_bspline` *object*

Description

Plots basis functions specified by results of `bspline`.

Usage

```
## S3 method for class 'hero_bspline'
plot(x, nderiv = 0L, type = "l", kcol = NULL, ...)
```

Arguments

`x`	An object of class `hero_bspline` to be plotted.
`nderiv`	An integer value specifying the derivative order of the B-splines. The default is 0.
`type`	character string (length 1 vector) or vector of 1-character strings indicating the type of plot for each column of `y`, see `plot` for all possible `types`. The first character of `type` defines the first plot, the second character the second, etc. Characters in `type` are cycled through; e.g., `"pl"` alternately plots points and lines.
`kcol`	Color for vertical lines drawn at interior knots. Default is `NULL`, meaning no lines are drawn.
`...`	Additional graphical parameters passed to `matplot` function.

See Also

`bspline`

Examples

```
x = bspline(nbasis = 10, extend = FALSE)
plot(x)
plot(x, nderiv = 1)
plot(x, kcol = "grey") # plot vertical lines at knots

# extend knots passed rangeval
x2 = bspline(nbasis = 10, extend = TRUE)
plot(x2, kcol = "grey")

# compare to plot.fd
```

```
if (requireNamespace("fda", quietly = TRUE)) {
   x3 = fda::create.bspline.basis(nbasis = 10)
   par(mfrow = c(2, 1))
   plot(x, kcol = "grey")
   title("plot.hero_bspline")
}
plot(x3)
title("plot.fd")
```

`plot.hero_enlarge` *Plot a* `hero_enlarge` *object*

Description

Plot the enlarged and original border defined be a set of coordinates.

Usage

```
## S3 method for class 'hero_enlarge'
plot(x, ..., blist = list(col = "grey"))
```

Arguments

`x`	An object of class `hero_enlarge`.
`...`	Additional graphical parameters passed to the plotting method for `SpatialPolygons-class` for `x$eborder`
`blist`	A list of additional graphical parameters passed to the plotting method for `SpatialPolygons-class` for `x$border`.

See Also

`SpatialPolygons-class`

Examples

```
b = border.grid(lon, lat)
e = enlarge(b)
plot(e)
```

`plot.hero_matrix` *Plot a* `hero` *object*

Description

Plot the smoothed data produced by the `hero` function. The behavior of the function changes depending on the subclass of the `hero` object. See Details.

Usage

```
## S3 method for class 'hero_matrix'
plot(x, xlab = "", ylab = "", ...)

## S3 method for class 'hero_numeric'
plot(x, xlab = "", ylab = "", type = "l", ...)
```

Arguments

`x`	An object of class `hero`.
`xlab`	x-axis label
`ylab`	y-axis label
`...`	Additional graphical parameters passed to the relevant plotting function. See Details.
`type`	The plot type (when `x` is of subclass `hero_numeric`). Default is `type = "l"`.

Details

If `x` has subclass `hero_numeric`, then the traditional `plot` function is used to plot the smoothed data, with `type = "l"`.

If `x` has subclass `hero_matrix`, then `image` is used to plot the smoothed data, or if the autoimage package is installed, `autoimage` is used to plot the smoothed data.

See Also

`hero`

Examples

```
# create b-splines
x1 = bspline(nbasis = 10)
x2 = bspline(nbasis = 12)

# observed data locations
evalarg1 = seq(0, 1, len = 60)
evalarg2 = seq(0, 1, len = 80)

# construct "true" data
```

```
mu = matrix(0, nrow = 60, ncol = 80)
for(i in seq_len(60)) {
   for(j in seq_len(80)) {
      mu[i, j] =  sin(2*pi*(evalarg1[i]-.5)^3)*cos(4*pi*evalarg2[j])
   }
}
# construct noisy data
data = mu + rnorm(60 * 80)

obj = prepare(data, list(evalarg1, evalarg2), list(x1, x2))
obj = enhance(obj)
sandmod = hero(obj)
plot(sandmod)
```

`plot.hero_radspline` *Plot a* `hero_radspline`

Description

Plot a `hero_radspline` to compare the knots to the observed data locations.

Usage

```
## S3 method for class 'hero_radspline'
plot(
  x,
  blist = list(col = "grey"),
  glist = list(col = seq_along(x$grid) + 1, pch = seq_along(x$grid)),
  ...
)
```

Arguments

`x`	A `hero_radspline` object.
`blist`	A list to pass the plot method associated with `SpatialPolygons-class` when plotting `x$border`. The default is a grey-colored polygon.
`glist`	A list to pass the plot method associated with `SpatialPoints-class` when plotting each element of the list `x$grid`. A basic color scheme and point style is automatically chosen if none is supplied.
`...`	Additional arguments to pass the plot method associated with `SpatialPolygons-class` when plotting `x$eborder`.

Details

If the default plotting styles for `x$grid` are to be changed, the user can either choose a single color/style that is replicated for each element of `x$grid` or supply a vector which has length matching `length{x$grid}`. See Examples.

Author(s)

Joshua French

See Also

radspline

Examples

```
border = border.grid(lon, lat)
r = radspline(nknots = c(36, 36 * 4), border = border)
# default color scheme
plot(r)
# change color and point styles of points,
# and background of original domain
plot(r, blist = list(col = "yellow"),
        glist = list(col = c("blue", "orange"),
                     pch =  3:4))
```

---

poly2SpatialPolygons *Convert simple polygon to a* SpatialPolygons *object*

Description

This function takes a simple polygon and attempts to convert it to a SpatialPolygons object. This list is assumed to have components x and y that define the boundary of the polygon.

Usage

```
poly2SpatialPolygons(x, ID = "border")
```

Arguments

x	A list with components x and y.
ID	The name of the resulting polygon. Default is "border".

Value

A SpatialPolygons object

Author(s)

Joshua French

Examples

```
angle = seq(0, 2 * pi, len = 100)
poly = list(x = cos(angle), y = sin(angle))
plot(poly, type = "l", asp = 1)
sppoly = poly2SpatialPolygons(poly)
library(sp)
plot(sppoly, axes = TRUE, asp = 1)
```

`precompute` *Precompute objects*

Description

This function is an internal function to compute objects needed for fast implementation of the sandwich smoother. It is meant to be an internal function, so use this at your own risk.

Usage

```
precompute(B, P, m)
```

Arguments

`B`	A matrix of basis functions
`P`	A penalty matrix
`m`	Difference order of P-spline

Value

A list of needed objects

Examples

```
object = bspline(nbasis = 10)
# sequence to evaluate
evalarg = seq(0, 1, len = 11)
# penalty matrix
D = diffpen(object)
P = Matrix::crossprod(D)
B = predict(object, evalarg)
stuff = precompute(B, P, m = 2)
```

predict.hero *Predict method for* hero *object*

Description

Predict new values based on object produced by the hero function.

Usage

```
## S3 method for class 'hero'
predict(object, newB, ...)
```

Arguments

object	A hero_bspline object created by bspline
newB	A vector or list containing the evaluated basis functions for the observations for which predictions are desired.
...	Not currently implemented.

Value

A matrix of the appropriate size

Examples

```
# create b-splines
x1 = bspline(nbasis = 10)
x2 = bspline(nbasis = 12)

# observed data locations
evalarg1 = seq(0, 1, len = 60)
evalarg2 = seq(0, 1, len = 80)

# construct "true" data
mu = matrix(0, nrow = 60, ncol = 80)
for(i in seq_len(60)) {
   for(j in seq_len(80)) {
      mu[i, j] =  sin(2*pi*(evalarg1[i]-.5)^3)*cos(4*pi*evalarg2[j])
   }
}
# construct noisy data
data = mu + rnorm(60 * 80)

obj = prepare(data, list(evalarg1, evalarg2), list(x1, x2))
obj = enhance(obj)
sandmod = hero(obj)
plot(sandmod)
newb1 = predict(x1, newx = seq(0, 1, len = 100))
newb2 = predict(x2, newx = seq(0, 1, len = 100))
```

```
newB = list(newb1, newb2)
p = predict(sandmod, newB = list(newb1, newb2))
```

`predict.hero_bspline` *Predict method for* `hero_bspline` *object*

Description

Predicted values based on object created by `bspline`.

Usage

```
## S3 method for class 'hero_bspline'
predict(object, newx, nderiv = 0L, sparse = TRUE, ...)
```

Arguments

`object`	A `hero_bspline` object created by `bspline`
`newx`	A numeric vector of values at which to evaluate the B-spline functions or derivatives.
`nderiv`	An integer value specifying the derivative order of the B-splines. The default is 0.
`sparse`	A logical value indicating if the result should be a sparse version of the `Matrix-class`.
`...`	Not currently implemented.

Value

An $n \times k$ matrix (or `Matrix-class` object if `sparse = TRUE`), where n is the number of values in `newx` and k is the number of basis functions in `object`. Each row gives the predicted values of the basis functions for the appropriate value of `newx`.

See Also

`bspline`

Examples

```
b = bspline(nbasis = 10)
p = predict(b, newx = seq(0, 1, len = 101))
```

predict.hero_radspline *Predict method for a* hero_radspline

Description

Predicted values based on object created by radspline.

Usage

```
## S3 method for class 'hero_radspline'
predict(object, newx, sparse = TRUE, longlat = FALSE, join = TRUE, ...)
```

Arguments

object	A hero_radspline object created by radspline.
newx	A numeric matrix at which to evaluate the radial basis splines functions.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
longlat	Use Euclidean (FALSE) or Great Circle (WGS84 ellipsoid) distance (TRUE). Default is FALSE.
join	A logical value. TRUE, the default, indicates that the predictions from each set of radial basis functions should be joined column-wise. Otherwise, a list with the predictions from each set of basis functions is returned.
...	Not currently implemented.

Value

An $n \times k$ matrix (or Matrix-class object if sparse = TRUE), where n is the number of rows in newx and k is the number of basis functions in object. Each row gives the predicted values of each newx value evaluated at each of the basis functions.

See Also

radspline

Examples

```
border = border.grid(lon, lat)
r = radspline(nknots = c(36, 36 * 4), border = border)
newx = cbind(c(lon), c(lat))
p = predict(r, newx)
```

prepare *Prepare data for sandwich smooth*

Description

A generic function to prepare various types of data. See the functions linked in See Also.

Usage

```
prepare(data, ...)
```

Arguments

data	The data to prepare
...	Not implemented

Value

A prepared object

See Also

prepare.numeric, prepare.matrix, prepare.array, prepare.sts, prepare.starray

prepare.array *Prepare data array for sandwich smooth*

Description

prepare.array prepares a data matrix for the sandwich smooth. The dimensionality of data and the length of x must match. Specifically, length(dim(data)) must equal length(x). The dimensionality of data and the length of splines must match. Specifically, length(dim(data)) must equal length(splines).

Usage

```
## S3 method for class 'array'
prepare(data, x, splines, m = 2, sparse = TRUE, ...)
```

Arguments

data	A data array
x	A list of univariate, equidistant sequences. These should correspond to where the data are observed. Equidistant spacing between 0 and 1 is assumed if not supplied. See Details.
splines	A list of spline-related objects, e.g., produced by bspline. Splines are automatically created if not supplied. See Details.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
...	Not currently implemented.

Details

For a typical sandwich smooth, for data with d dimensions, `Y[i1, i2, ...,id]` is assumed to be observed at position `x[[1]][i1]`, `x[[2]][i2]`, ..., `x[[d]][id]`. Consequently, `dim(data)[i]` should equal `length(x[[i]])` for all `i` in `seq_len(d)`.

If `x` is not supplied, then `default.evalargs` is used to create it automatically.

If `splines` is not supplied, then a B-spline basis is automatically created for each dimension using `default.splines`.

Value

A `prepared_array` object.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

See Also

`bspline`, `default.evalargs`, `default.splines`

Examples

```
# generate and prepare 3d data
set.seed(9)
dat = generate.data3d()
obj = prepare(dat$data3d, x = dat$x)
```

`prepare.list` *Prepare data array for sandwich smooth*

Description

`prepare.list` prepares a list of data for the sandwich smooth. The class of each element of the list must be identical. The dimensionality of `data[[i]]` and the length of `x` must match. Specifically, `length(dim(data[[i]]))` must equal `length(x)`. The dimensionality of `data[[i]]` and the length of `splines` must match. Specifically, `length(dim(data[[i]]))` must equal `length(splines)`. Note: If the splines are preassembled, these can be passed using the argument `assembled` so that this computation is not reperformed.

Usage

```
## S3 method for class 'list'
prepare(data, x, splines, m = 2, sparse = TRUE, ...)
```

Arguments

`data`	A list of `numeric`, `matrix`, or `array` objects.
`x`	A list of values at which to evaluate the basis functions. See Examples and Details.
`splines`	A list of spline objects (`hero_bspline` and `hero_radspline`). See Examples and Details.
`m`	A positive integer indicating order of the difference penalty.
`sparse`	A logical value indicating if the result should be a sparse version of the `Matrix-class`.
`...`	Not currently implemented.

Details

This function applies the functions `prepare.numeric`, `prepare.matrix`, and `prepare.array` to each element of the list, so relevant restrictions in the arguments may be found there.

Value

A `prepared_list` object.

Author(s)

Joshua French.

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

See Also

prepare.numeric, prepare.matrix, prepare.array

Examples

```
# generate and prepare 3d data
set.seed(9)
dat = lapply(1:3, function (i) generate.data3d())
x = dat[[1]]$x
data = lapply(dat, getElement, name = "data3d")
obj = prepare(data, x = x)
h = hero(obj)
```

---

prepare.matrix *Prepare data matrix for sandwich smooth*

Description

prepare.matrix prepares a data matrix for the sandwich smooth. The dimensionality of data and the length of x must match. Specifically, length(dim(data)) must equal length(x). The dimensionality of data and the length of splines must match. Specifically, length(dim(data)) must equal length(splines).

Usage

```
## S3 method for class 'matrix'
prepare(
  data,
  x,
  splines,
  m = 2,
  sparse = TRUE,
  spdiffpen = TRUE,
  digits = 1,
  sts = FALSE,
  ...
)
```

Arguments

data	A data matrix.
x	A list of values at which to evaluate the basis functions. See Examples and Details.
splines	A list of spline objects (hero_bspline and hero_radspline). See Examples and Details.
m	A positive integer indicating order of the difference penalty.

sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
spdiffpen	A logical value indicating whether spdiffpen should be used to compute the difference penalty. The default is FALSE.
digits	The number of digits to use when applying round to the distances.
sts	A logical value indicating whether data is a spatial time series, in which each row of data corresponds to a distinct spatial location and each column corresponds to a distinct time.
...	Not currently implemented.

Details

For a typical sandwich smooth (sts = FALSE), for two-dimensional data, data[i, j] is assumed to be observed at position x[[1]][i], x[[2]][j]. If the data are a spatial time series, then the first dimension is assumed to refer to space, and the second dimension to time. In that case, data[i, j] is assumed to be observed at location x[[1]][i,] and time x[[2]][j].

If sts = TRUE, then x[[1]] should be a matrix of spatial coordinates, with each row corresponding to a location, and x[[2]] should be a vector with the observation times.

If x is not supplied, then default.evalargs is used to create it automatically. This is only valid when sts = FALSE.

If splines is not supplied, then a B-spline basis is automatically created for each dimension using default.splines. This is only valid when sts = FALSE.

Value

A prepared_matrix object.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

See Also

bspline, radspline, diffpen, spdiffpen, default.evalargs, default.splines

Examples

```
# prepare Lu et al. (2012) noisy f1 data
data(ludata)
obj = prepare(lunoisyf1, x = list(x, z))
h = hero(obj)

# precompute some stuff
splines = default.splines(list(x, z))
```

```
l = assemble(splines, x = list(x, z))
obj2 = prepare(lunoisyf1, x = list(x, z),
               splines = splines, assembled = l)
h2 = hero(obj2)
all.equal(h, h2)
```

`prepare.numeric` *Prepare data vector for sandwich smooth*

Description

`prepare.vector` prepares a data vector for the sandwich smooth. Unlike the other `prepare.*` functions, `x` and `splines` do not need to be lists since the data are 1-dimensional.

Usage

```
## S3 method for class 'numeric'
prepare(data, x, splines, m = 2, sparse = TRUE, ...)
```

Arguments

`data`	A numeric data vector
`x`	A sequence of equidistant values corresponding to where the data are observed. Equidistant spacing between 0 and 1 is assumed if not supplied. See Details.
`splines`	A spline-related object, e.g., produced by `bspline`. A spline is automatically created if not supplied. See Details.
`m`	A positive integer indicating order of the difference penalty.
`sparse`	A logical value indicating if the result should be a sparse version of the `Matrix-class`.
`...`	Not currently implemented.

Details

If `x` is not supplied and `n` is the `length(data)`, then the function automatically sets `x = seq(0, 1, length = n)`.

If `splines` is not supplied, and `n` is the `length(data)`, then the function automatically sets `splines = bspline(range(x), nknots = min(ceiling(n/4), 35))`.

Value

A `prepared_numeric` object.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

Ruppert, D., Wand, M. P., & Carroll, R. J. (2003). Semiparametric Regression. Cambridge University Press. <doi:10.1017/CBO9780511755453>

See Also

bspline, default.evalargs, default.splines

Examples

```
# create data
n = 160
x = seq(0, 4 * pi, len = n)
# "true" data
mu = sin(x)
# plot true data
plot(x, mu, type = "l")
# construct noisy data
set.seed(4)
data = mu + rnorm(n)

# construct spline
splines = bspline(c(0, 4 * pi), nknots = 20)
# prepare/enhance data
obj = prepare(data, x, splines)
obj = enhance(obj)
sandmod = hero(obj)
plot(sandmod, ylim = range(data), lty = 2)
lines(x, data, col = "lightgrey")
lines(x, mu)
legend("bottomleft",
       legend = c("smoothed", "true", "observed"),
       lty = c(2, 1, 1),
       col = c("black", "black", "grey"))
```

---

prepare.starray *Prepare* starray *for sandwich smooth*

Description

prepare.starray prepares a spatio-temporal array for the sandwich smooth.

Usage

```
## S3 method for class 'starray'
prepare(data, x, y, times, rs, bs, m = 2, sparse = TRUE, spdiffpen = TRUE, ...)
```

Arguments

data	An starray
x	A vector or matrix of x coordinates. See Details.
y	A vector or matrix of y coordinates. See Details.
times	The vector of times at which the data were observed.
rs	A hero_radspline produced by the radspline or connect functions.
bs	A hero_bspline produced by the bspline function.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
spdiffpen	A logical value indicating whether spdiffpen should be used to compute the difference penalty. The default is FALSE.
...	Not currently implemented.

Value

A prepared_starray object.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

See Also

bspline, radspline

Examples

```
# construct basis functions
border = border.grid(lon, lat)
rs = radspline(nknots = 36, poverlap = 3,
               border = border, longlat = TRUE)
bs = bspline(c(1, 30), nbasis = 6)
data = starray(tasmax)
p = prepare(data, x = lon, y = lat, times = 1:30,
            rs = rs, bs = bs)
```

prepare.sts *Prepare* starray *for sandwich smooth*

Description

prepare.starray prepares a spatio-temporal array for the sandwich smooth.

Usage

```
## S3 method for class 'sts'
prepare(
  data,
  coords,
  times,
  rs,
  bs,
  m = 2,
  sparse = TRUE,
  spdiffpen = TRUE,
  ...
)
```

Arguments

data	An starray
coords	A two-dimensional matrix-like object with non-NULL dimensions.
times	The vector of times at which the data were observed.
rs	A hero_radspline produced by the radspline or connect functions.
bs	A hero_bspline produced by the bspline function.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
spdiffpen	A logical value indicating whether spdiffpen should be used to compute the difference penalty. The default is FALSE.
...	Not currently implemented.

Value

A prepared_sts object.

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

See Also

bspline, radspline

Examples

```
# construct basis functions
border = border.grid(lon, lat)
rs = radspline(nknots = 36, poverlap = 3,
               border = border, longlat = TRUE)
bs = bspline(c(1, 30), nbasis = 6)
splines = list(rs, bs)
data = as.sts(tasmax)
p = prepare(data, coords = cbind(c(lon), c(lat)),
            times = 1:30, rs = rs, bs = bs)
```

---

prepare_sequential *Sequentially prepare data for sandwich smooth*

Description

Sequentially prepare each observation for smoothing. It is assumed that each observation resides in its own file and that `do.call(import_fun,list(import_list[i]))` will import the data associated with observation `i` into memory. The `import_fun` argument should be a function after the style of readRDS, where the object can be assigned a name once it is read in. The `import_fun` argument should NOT be like load, where the object loaded has a preassigned name.

Usage

```
prepare_sequential(
  import_list,
  import_fun = base::readRDS,
  x,
  splines,
  assembled,
  package = "base",
  call_args = list(),
  ...
)
```

Arguments

import_list	A vector or list whose elements tell import_fun which files to import.
import_fun	A function that will read each observation into memory based on the elements of import_list.
x	The list of arguments at which to evaluate each of the splines used to construct assembled.
splines	A list of spline-related objects. Each element of splines corresponds to the set of splines for the corresponding element of x.
assembled	A list of assembled_splines. See Examples.
package	A character string indicating the package to use for the computations. The choices are "base", "parallel", "pbapply", "future.apply", and "Rmpi". The default is "base", in which case a standard for loop is used. If package == "parallel", then mclapply is used, which is only appropriate when mc.cores is integer-valued or NULL. If package == "pbapply", then pblapply is used, which automatically provides a progress bar. If package == "future.apply", then future_lapply is used. If package == "Rmpi", then mpi.applyLB is used.
call_args	A named list providing relevant arguments to the mclapply, pblapply, future_lapply, or mpi.applyLB depending on the value of package.
...	Not implemented

Value

A prepared_sequential object

Author(s)

Joshua P. French

See Also

prepare, mclapply, pblapply, future_lapply, mpi.applyLB

---

radspline *Radial basis spline specification*

Description

radspline specifies a set of radial basis splines. nknots is the approximate number of knots to sample in the (usually) enlarged study area. If eborder is not provided, then eborder is automatically constructed by enlarging the border object using the enlarge function and width. See Details for additional information about sampling the knot locations.

Usage

```
radspline(
  nknots,
  border,
  poverlap = 2,
  k = 2,
  width,
  type = "hexagonal",
  longlat = FALSE,
  eborder,
  ...
)
```

Arguments

nknots	The approximate number of knots locations. Can be a vector of positive integers for successive samplings. See Details.
border	A SpatialPolygons-class object. If eborder is not supplied, this will be used to determine the sampling region for the knots. See Details.
poverlap	The proportional amount of overlap (>=1) beyond the nearest neighbor knots. Default is 2.
k	The order of the Wendland covariance function.
width	The width for the border enlargement.
type	The sampling type for spsample. The default is "hexagonal".
longlat	A logical value indicating whether Great Circle distances should be used (TRUE) or Euclidean distances (FALSE). The default is FALSE.
eborder	A SpatialPolygons-class object. The enlarged border from which the knots will be selected. If not supplied, this is automatically computed using border and width.
...	Additional arguments passed to spsample.

Details

The spsample function is used to "automatically" select the knot locations within eborder. nknots corresponds to the n argument in that function. A hexagonal sampling scheme is used by default, but other options are available.

Great circle distance IS NOT used in sampling from the regular grid. This is computationally expensive, so it has not been implemented. Great circle distance is only used when the constructed hero_radspline is evaluated (and longlat = TRUE).

Value

A hero_radspline object.

Examples

```
border = border.grid(lon, lat)
r = radspline(nknots = c(36, 36 * 4), border = border)
# default color scheme
plot(r)
# change color and point styles of points,
# and background of original domain
plot(r, blist = list(col = "yellow"),
        glist = list(col = c("blue", "orange"),
                     pch =  3:4))
```

---

rh *Rotated H-transform*

Description

A rotation of the H-transform of the array a by a matrix x.

Usage

```
rh(x, a, transpose = FALSE)
```

Arguments

x	A matrix-like object. See Details.
a	An d-dimensional array
transpose	A logical value. The Default is FALSE. If TRUE, then the transpose of A

Details

x should be matrix-like. This function has been tested when x is a `matrix` object or a `Matrix`.

Assuming a is of size $c_1 \times c_2 \times \cdots \times c_d$, then x is of size $r \times c_1$.

Value

A rotated, h-transformed array

Author(s)

Joshua French. Based off code by Luo Xiao (see References).

References

Currie, I. D., Durban, M. and Eilers, P. H. (2006), Generalized linear array models with applications to multidimensional smoothing. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 68: 259-280. <doi:10.1111/j.1467-9868.2006.00543.x>

Xiao, L. , Li, Y. and Ruppert, D. (2013), Fast bivariate P-splines: the sandwich smoother. J. R. Stat. Soc. B, 75: 577-599. <doi:10.1111/rssb.12007>

Examples

```
dim = c(10:12)
# construct random array
a = array(rnorm(prod(dim)), dim = dim)
# construct random matrix
x = matrix(rnorm(15 * dim[1]), nrow = 15)
rhxa = rh(x, a)
```

`rh.seq` *Apply* `rh` *sequentially*

Description

`rh.seq` sequentially applies the `rh` function to `a`. Specifically, if the length of `x` is `d`, then `rh.seq(x, a)` is equivalent to `rh(x[[d]], rh(x[[d - 1]], ..., rh(x[[2]], rh(x[[1]], a))..))`.

Usage

```
rh.seq(x, a, transpose = FALSE)

rh_seq(x, a, transpose = FALSE)

rhSeq(x, a, transpose = FALSE)

RhSeq(x, a, transpose = FALSE)
```

Arguments

`x`	A list of matrix-like objects
`a`	A matrix-like object (with dimensions)
`transpose`	A logical value. The Default is `FALSE`. If `TRUE`, then the transpose of `A`

Value

A `matrix` or `Matrix-class`.

Examples

```
# generate x, a
x = list(matrix(rnorm(100), nrow = 10),
         matrix(rnorm(100), nrow = 10))
a = matrix(rnorm(100), nrow = 10)

# three equivalent forms
rhs1 = rh.seq(x, a)
rhs2 = rh(x[[2]], rh(x[[1]], a))
rhs3 = x[[1]] %*% a %*% t(x[[2]])
```

```
# check equality
all.equal(rhs1, rhs2)
all.equal(rhs1, rhs3)
```

---

spdiffpen *Spatial difference penalty*

Description

spdiffpen computes the mth order spatial difference penalty for a set of coordinates.

Usage

```
spdiffpen(coords, m = 1, sparse = TRUE, longlat = FALSE, digits = 1)
```

Arguments

coords	A two-dimensional matrix-like object with non-NULL dimensions.
m	A positive integer indicating order of the difference penalty.
sparse	A logical value indicating if the result should be a sparse version of the Matrix-class.
longlat	A logical value indicating whether Great Circle distances should be used (TRUE) or Euclidean distances (FALSE). The default is FALSE.
digits	The number of digits to use when applying round to the distances.

Details

adjacent is used to determine the first-order neighbors of each point in coords. The difference penalties are then successively determined from that.

If sparse = TRUE, a sparseMatrix-class Matrix is returned when the penalty matrix is relatively sparse (typically, at least half the entries are zero). Otherwise, something of the more general Matrix-class is returned.

Value

A matrix or sparseMatrix-class object.

Examples

```
coords = expand.grid(1:4, 1:4)
# first order difference penalty
d1 = spdiffpen(coords, digits = 1)
# second order difference penalty
d2 = spdiffpen(coords, m = 2, digits = 1)
# third order difference penalty
d3 = spdiffpen(coords, m = 3, digits = 1)
```

tasmax *Computer-generated temperature data*

Description

The maximum daily surface air temperature (C) for the time period January 1, 1971 through January 30, 1971 for the ECP2-GFDL computer generated data made available through the North American Regional Climate Change Assessment Program (NARCCAP).

Usage

```
data(tasmax)
```

Format

Matrices `lon` and `lat` and array `tasmax`.

References

Mearns, L.O., et al., 2007, updated 2014. The North American Regional Climate Change Assessment Program dataset, National Center for Atmospheric Research Earth System Grid data portal, Boulder, CO. Data downloaded 2018-06-13. <doi:10.5065/D6RN35ST>.

Mearns, L. O., W. J. Gutowski, R. Jones, L.-Y. Leung, S. McGinnis, A. M. B. Nunes, and Y. Qian. "A regional climate change assessment program for North America." EOS, Vol. 90, No. 36, 8 September 2009, pp. 311-312. <doi:10.1029/2009EO360002>.

wrfg_cgcm3_tasmax *Computer-generated temperature data*

Description

The maximum daily surface air temperature (C) of land locations for the time period January 1, 2041 through January 30, 1941 for the WRFG-CGCM3 computer generated data made available through the North American Regional Climate Change Assessment Program (NARCCAP). The non-land locations are specified as NA.

Usage

```
data(wrfg_cgcm3_tasmax)
```

Format

Matrices `wrfg_lon` and `wrfg_lat` and array `wrfg_cgcm3_tasmax`.

References

Mearns, L.O., et al., 2007, updated 2014. The North American Regional Climate Change Assessment Program dataset, National Center for Atmospheric Research Earth System Grid data portal, Boulder, CO. Data downloaded 2018-06-13. <doi:10.5065/D6RN35ST>.

Mearns, L. O., W. J. Gutowski, R. Jones, L.-Y. Leung, S. McGinnis, A. M. B. Nunes, and Y. Qian. "A regional climate change assessment program for North America." EOS, Vol. 90, No. 36, 8 September 2009, pp. 311-312. <doi:10.1029/2009EO360002>.

Index