PBSsatellite: User's Guide

Nicholas R. Lefebvre and Nicholas M. Boers
lefebvren4@mymacewan.ca and boersn@macewan.ca

Technical Report
TR-2015-001
July 21, 2015

Department of Computer Science
MacEwan University
Edmonton, Alberta, Canada

Abstract

This report describes the first version of PBSsatellite, software designed to simplify the extraction and statistical analysis of gridded satellite data. This software extends the R Project for Statistical Computing and uses PBSmapping, an existing R software package, to produce plots. The tools provided in this package provide users with the functionality necessary to accomplish statistical analysis of a myriad of satellite data sources. Additionally, users are able to produce their own functionality and data interpretation algorithms and incorporate these functions into the package's analysis methods. This provides users with a platform that simplifies the complex problem of satellite data analysis and visualization while providing the ability for personal research and method integration.

Acknowledgements

We thank Dr. Lyse Godbout from Fisheries and Oceans Canada for approaching us with the idea that led to the PBSsatellite R package. Moreover, we thank Rowan Haigh (Fisheries and Oceans Canada) and Lyse for their insight and assistance as we developed this software.

Contents

1 Introduction

This report describes the PBSsatellite software library that was designed to simplify the extraction and statistical analysis of gridded satellite data.

This chapter explores two fundamental aspects of the package: (a) the use of the NetCDF file format and the reasons for its adoption and (b) the data sources that were used to design the package's data structures.

Chapter 2 explains of the package's primary data structure (`ncdfData`). Most of the package's functions for the manipulation and extraction of data require objects of this structure. Additionally, this chapter explains the `TimeSeries` data structure in further detail. This latter object type simplifies the visualization of satellite data trends and the development of plots.

Chapter 3 displays complex applications of the PBSsatellite library's features. One situation, which will be further explored later, involves creating a complex polygon for the coast of British Columbia. This polygon can then be used with the function `extractTimeSeries` to create a `TimeSeries` object. Another example shows how one can go from a `TimeSeries` object to a time series plot. Time series plots provide visualization of satellite trends that can ultimately improve one's understanding of satellite data over time. The third and concluding situation guides a user step-by-step through the conversion of an incompatible file format (HDF) into a NetCDF file that can be imported into PBSsatellite. The process involves converting three HDF files into three NetCDF files and merging them into a single NetCDF file. With additional attributes, this particular NetCDF file will then become fully compatible with the package's functionality.

Chapter 4 documents the functions found in PBSsatellite. This function documentation is also available with R's help system.

1.1 NetCDF dependency

There are three widely used formats for exchanging and storing meteorological data: (a) Extensible Markup Language (XML), (b) Network Common Data Format (NetCDF), and (c) Hierarchical Data Format (HDF) [1]. The verbosity of XML-formatted documents leads to large file sizes, and thus it was not considered for this package. We evaluated the two remaining options, NetCDF and HDF, in the initial stages of designing PBSsatellite.

We selected NetCDF over HDF primarily due to the quality of available R packages on Comprehensive R Archive Network (CRAN), which is known to host well-maintained R software. At the time of writing, CRAN does not host any packages explicitly for importing HDF4 files. The hosted package rgdal provides bindings for the Geospatial Data Abstraction Library, which can import HDF4 files when appropriately configured. Unfortunately, the Mac OS X version and reportedly the Windows version[1] are built without support for HDF4. In contrast, three NetCDF packages are hosted on CRAN: ncdf, ncdf4, and RNetCDF. There, we selected the NetCDF format over HDF for PBSsatellite.

After focusing on NetCDF, we aimed to build upon the best of the three NetCDF packages available. The best package would maximize NetCDF functionality while minimizing additional system requirements (see Table 1.1).

Table 1.1: NetCDF Libraries

Library	**Advantages**	**Disadvantages**
ncdf	• minimal requirements • provides sufficient NetCDF functionality	• supports only NetCDF version 3
ncdf4	• supports NetCDF versions 3 & 4 • provides sufficient NetCDF functionality	• requires netcdf ($\geq$ 4.1)
RNetCDF	• none	• requires netcdf ($\geq$ 3.6), udunits ($\geq$ 1.11.7), or udunits2 ($\geq$ 2.1.22) • supports only NetCDF version 3

It was important to make PBSsatellite usable with as many data sets as possible. Given the information in Table 1.1, we chose ncdf4 for PBSsatellite. While ncdf4 adds one external dependency, it provides support for both versions 3 and 4 of NetCDF files and the ability to offset into NetCDF data files. When users are interested in only a subset of data within a large data set, the offset functionality can make processing significantly faster. If a user is interested in a section of data that does not begin at the start of a file, offset functionality can effectively skip this data and jump to the sections containing the desired data preventing the processing of undesired data.

1.2 Data Sources

Before we could design the `ncdfData` object (Section 2.1), we explored a variety of NetCDF data files. As previously stated, the ncdf4 library was selected (see Table 1.1) because it supported both NetCDF versions, and thus, the most NetCDF files. We faced a similar challenge when creating the `ncdfData` data object. It was important to be able to design `ncdfData` objects –the

[1] We did not test rgdal on Windows, but in online forums, a number of posts commented on rgdal's lack of HDF4 support on Windows.

Table 1.2: The NetCDF data sources initially selected prior to designing the `ncdfData` object. A data location of "error" indicates that the R ncdf4 library could not open the file, and such files were not used in the design stage.

Source	**Data Type**	**Data Location**
NOAA	Bedrock	error
NOAA	Sea Surface Temperature (Kelvin)	1
NOAA	Sea Surface Temperature (Celsius)	1
NOAA	Sea Surface Temperature	error
NOAA	Sea Surface Temperature (Kelvin)	1
JISAO	Chlorophyll Concentrations	1
USJGOFS	Chlorophyll Concentrations	2
CRU	Sea Surface Temperature (Kelvin)	1

backbone data structure for PBSsatellite– from multiple data sets and various NetCDF files. For this reason, a variety of data sets were analyzed for consistencies, particularly in data location and attribute naming.

If the source data sets named their attributes in a particular convention, we adopted this naming convention as our default naming for `ncdfData` attribute acquisition. For example, most NetCDF data sets use the names "lat" and "lon" to store attributes of latitude and longitude coordinate sequences. In order to accommodate data sets that did not follow this naming convention for particular attributes, the import functions allow attribute names to be overwritten. Whenever possible, PBSsatellite will locate fundamental attributes automatically. When a NetCDF data file deviates from the chosen naming standards, the user will need to provide attribute names to ensure that PBSsatellite locates the correct attributes.

1.2.1 Sources

Eight data sets were used in creating the `ncdfData` object: five data sets were from the National Oceanic and Atmospheric Administration (NOAA), one was from the Joint Institute for the Study of the Atmosphere and Ocean (JISAO), one was from the U.S. Joint Global Ocean Flux Study (USJGOFS), and the last was from Climate Research Unit (CRU) developed by the University of East Anglia. See Table 1.2 for more information.

1.2.2 Data

We identified some consistencies among various data sets, e.g., see Table 1.2. One major consistency was the location of the data variable ("Data Location" in Table 1.2), which was most commonly placed in the first variable of the NetCDF files. Some data sets, however, used a different location, e.g., the second data variable used by USJGOFS. For this reason the first variable is the default value when creating a `ncdfData` object. The default data variable parameter can be overwritten in situations if necessary e.g., if using a data set from USJGOFS, by specifying an alternate index when creating the `ncdfData` object i.e., 2.

Another commonality found between the NetCDF data files was their distribution of data along X and Y coordinates. Most of the data sets had increasing X coordinates (longitude) and decreasing Y coordinates (latitude). This led to another standard adopted by the `ncdfData` object. The `ncdfData` object ensures that all data is stored in this manner which creates the origin of a `ncdfData` object at the top-left corner of a map. While it is possible to reorder the `ncdfData` after creation, changing to make both X and Y coordinates increasing is a time consuming task. For these reasons we adopted the style of the most frequent type of NetCDF files we have encountered, which was to have increasing X values and decreasing Y values. When a NetCDF file does not follow this convention (X increasing Y decreasing format) the `ncdfData` will detect this and organize the X and Y coordinates accordingly.

It was also common to see a variety of time units within the NetCDF files. For example, some files strictly used seconds since an epoch to represent time, whereas others used minutes or hours since an epoch. The time attribute also displayed great variability among data sets e.g., seconds since 1981-01-01 00:00:00 or hours since 1997-1-1 1:0:0. When a `ncdfData` object is created, a date conversion is performed on these time attributes which creates standardized date time stamps that help simplify data extraction and comparisons. For example, before our date conversion, a date of 1 could be assigned to a sheet of data, this refers to 1 second since 1981-01-01 00:00:00. After the date conversion we would change this date from 1 to 1981-01-01 00:00:01.

It is common for NetCDF data sets to have missing values (see Section 2.1.3), and data sets may use varying placeholders to represent these missing values, e.g., -32767 or -99. The ncdf4 package's function `nc_open`, which is used by PBSsatellite to read NetCDF files, is documented to automatically detect the NetCDF's missing value attribute and to replace all occurrences the specified value with `NA`. In this case, the ncdf4 library manages this situation automatically when reading NetCDF files and makes all `ncdfData` use `NA` as the missing value placeholder.

Other inconsistencies included the temperature units in sea surface temperature data sets. As stated in Chapter 2, `ncdfData` objects contain an attribute that stores the `ncdfData`'s data units. In most cases, the attributes in `ncdfData` objects are detected upon import without input from the user. In cases where an attribute is not located or an incorrect value is selected, the user can manually change the unit's variable when creating the `ncdfData` object (the same applies with the data type variable).

2 Data structures

PBSsatellite works with gridded satellite data in the NetCDF format and provides users with tools for manipulating, extracting, and analyzing information in a user-friendly manner. Given the variability in available NetCDF files (e.g., sea surface temperatures, chlorophyll concentrations, and ice concentrations), the package has one master data type (`ncdfData`) that standardizes the representation of the data. In addition to this new type, the `extractTimeSeries` function produces a data frame intended for statistical analyses. The sections that follow describe both of these data structures.

2.1 `ncdfData` Data Structure

2.1.1 Introduction

The `ncdfData` data structure is the primary one in PBSsatellite and is supported by most of the package functions. A `ncdfData` object is simply a list of named objects. Each object (a *slice*) represents satellite data from a point in time and is named with the date. A slice is a list of matrices, and each matrix is known as a *layer*. A slice always has a mandatory data layer that contains gridded satellite data from a specific time period, and this layer is always the first in the slice, i.e., the first element in the list of layers. In some situations, a slice has additional layers such as the missing and/or the error layer. These additional layers are created by the user using the PBSsatellite function `scaleRegion`, which is used to change the resolution of a `ncdfData` object. The `ncdfData` object contains a set of attributes that apply to the whole `ncdfData` object. These elements of a `ncdfData` object will be explained in further detail below.

Previously mentioned "with the exception of the data units with additional layers i.e., missing and/or error layers," but that doesn't make sense to me

2.1.2 Attributes

`ncdfData` objects must have the attributes listed in Table 2.1.

A `ncdfData` object requires at least one named *slice* (the "Names" attribute). A slice is a grid of satellite data captured at the same time and labelled by a time stamp. Naming slices in this way

Table 2.1: Required attributes for an `ncdfData` object.

Attribute	Description
`names`	Date names for each slice
`dataType`	Title of the data set
`dataUnits`	Units of the data set
`x`	Span of Longitude Coordinates
`y`	Span of Latitude Coordinates
`class`	name of the data structure (`ncdfData`)

allows for easy data extraction for both exact dates and date ranges. Slices are always stored in chronological order, i.e., the first slice in a `ncdfData` object is the oldest.

The "Data Type" attribute provides the title of the data being stored, e.g., "Long Term Mean of Sea Surface Temperature". Its value is often detected automatically when importing NetCDF data, but the user can override the title if desired.

The "Data Units" attribute refers to the actual units for the object's data component, e.g., "degC" or "Kelvin". As with "Data Type", it is often automatically detected and can be overridden. Within a `ncdfData` object, the units are always consistent, but they may vary between objects. Therefore, it is essential that all `ncdfData` objects are labelled with the unit.

The "X" attribute provides the sequence of longitude values (in degrees) for the X axis of the data in each slice. This sequence is always stored in ascending order, i.e., longitude values increase from left to right on a map. Note that internally, this attribute actually names the *rows* of a slice matrix rather than the columns. Storing the data in this manner allows software to use `[X, Y]` indexing of a matrix and achieve the expected result.

The "Y" attribute provides the sequence of latitude values (in degrees) for the Y axis of the data in each slice. This sequence is always stored in descending order, i.e., latitude values decrease from top to bottom on a map. Note that internally, this attribute actually names the *columns* of a slice matrix rather than the rows for the reason discussed earlier.

Given the sequence order for "X" and "Y", the top-left corner of a map is the origin. For example, consider a variable `s` that contains a matrix of data. The point `s[1, 1]` is thus located in the top-left corner of a plotted map.

In order to be considered a `ncdfData` object, an object's class must be `ncdfData`. When users manipulate or create `ncdfData` this indicates that the object is in fact a `ncdfData` object.

2.1.3 Structure Details

It is common for `ncdfData` to have multiple time slices. Slices can be retrieved via name or location in the list where they stored, i.e., slice 1 refers to the first slice in a list which is also the oldest. Slices are always required to have one component known as a layer. The data layer is required by all `ncdfData` objects and is where the data for each slice is located within the object. Slices may contain multiple layers such as the missing and/or the error layer. These two layers are optionally created when a `ncdfData` object is scaled down as indicated by the user. The missing

```
$names
[1] "1-02-01" "1-03-01" "1-04-01" "1-05-01"

$dataType
[1] "Long Term Mean of Sea Surface Temperature"

$dataUnits
[1] "degC"

$x
  [1]   0.5   1.5   2.5   3.5   ...    359.5

$y
  [1]  89.5  88.5  87.5  86.5   ...    -89.5

$class
[1] "ncdfData"
```

Figure 2.1: The attributes of a sample `ncdfData` object within R.

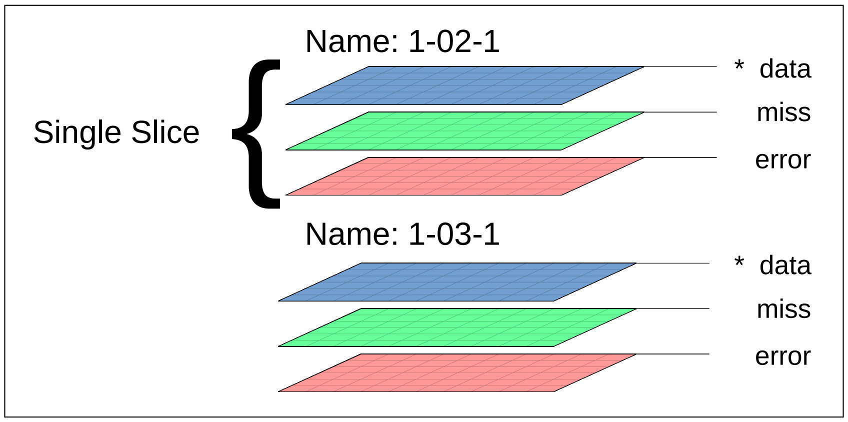


Figure 2.2: `ncdfData` Layers. (* = Required Layer)

layer contains a percentage of missing values that were used when scaling down a particular point in space. The error layer contains the percentage of error involved when scaling down a region. It is important to note that these additional layers must be spatially the same size as the data layer i.e., they must contain point for point as much data see figure 2.2.

Layers in a `ncdfData` object are stored as matrices. If there happens to be multiple layers in a slice, the layers can be retrieved by list notation similar to storing slices. The first layer in a slice always contains the slice's data layer.

In certain data sets such as the ones that pertain to SST (Sea Surface Temperature), it is common to have missing data values, e.g., data points on land. In a `ncdfData` object, a `NA` value is used to represent a location that is known to be missing from the data set. This can also happen due to cloud coverage where a satellite cannot measure an area.

Using matrices to store geographic data is problematic as they are limited to regions that are consistent in both height and width. It is essential to be able to use satellite data to perform analysis on particular geographic areas, most of which do not pertain to the dimensions of a

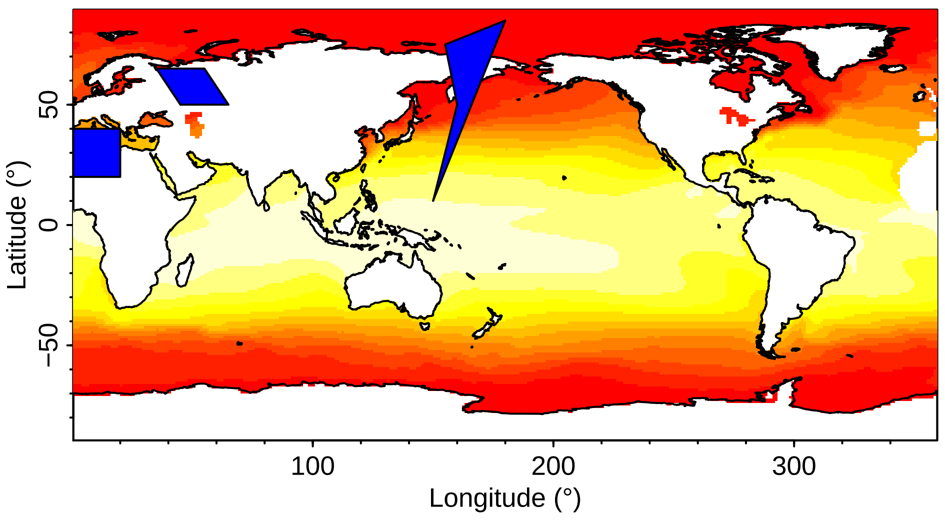


Figure 2.3: Three arbitrary polygons (blue) and the polygons from PBSmapping's `worldLL` data set (white) plotted on unclipped `ncdfData`.

matrix. A `ncdfData` object can be formatted to fit in a polygon's dimensions, which may often vary in length and width to accommodate complex region analysis; this is known as clipping. In order to store a clipped region in a matrix, removed areas due to clipping are stored with an alternative value to `NA`, we used `NaN` (Not a Number). This way a `ncdfData` can differentiate between areas missing due to the original data set (`NA`) and values missing due to a clipping operation (`NaN`). In order to save processing time and space for a `ncdfData` object that undergoes clipping, we will always use the smallest matrix in order to store all of the objects data (values that are not `NaN`). For an example, see Fig. 2.3 and Fig. 2.4.

Any rows or columns in a `ncdfData` that contain `NaN` values exclusively will be removed, as these regions are no longer of importance due to clipping.

A `ncdfData`'s attributes apply to all slices and layers (with the exception of data units for optional layers); therefore, all slices and layers in `ncdfData` contain matrices with the same dimensions and geographic representations.

2.2 TimeSeries Data Structure

A `TimeSeries` is essentially an analytical summary of a `ncdfData`'s slices over time. A user can create a `TimeSeries` object using the function `extractTimeSeries`. The `extractTimeSeries` function can use one of the standard R functions such as mean, sum, or sd (standard deviation) etc. The user can also create their own functions for different types of data analysis and `extractTimeSeries` will accept these functions as arguments. All of the provided summary functions will then be used to compute analysis on a `ncdfData` object for each sub region created by the `polygons` argument (see below for more information on the `polygons` argument).

If the argument `polygons` is provided to `extractTimeSeries` function will create subregions in each `ncdfData` slice. Data are then extracted from each of these subregions and processed by each function provided by the `functions` argument individually. If a `polygons` argument is

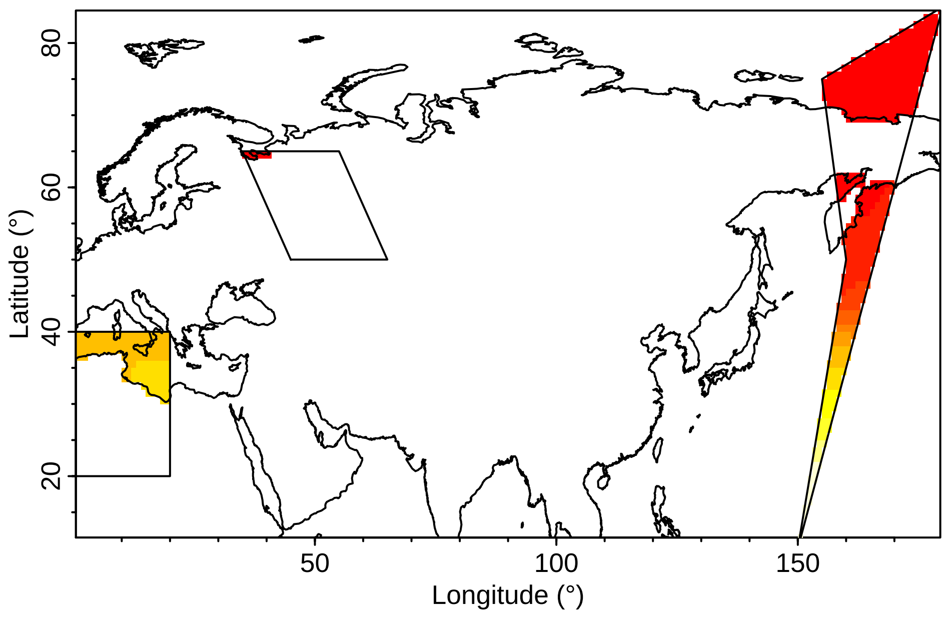


Figure 2.4: The result of clipping `ncdfData` using the polygons in Fig. 2.3. In the resulting `ncdfData` object, `NaN` values represent clipped areas outside of the three polygons and `NA` values represent areas inside the polygons that have a missing data component.

not provided the whole `ncdfData` is considered one large subregion and the entire object will be analyzed by the desired functions. This results in a `TimeSeries` object for the full `ncdfData` region.

A `TimeSeries` object is essentially a list of data frames. Each data frame summarizes a single slice of the original `ncdfData` object. Slices from the original `ncdfData` may be omitted from the `TimeSeries` object using `extractTimeSeries`'s `tlim` (time limit) argument. When the user provides a `tlim` argument, all slices that fall within this provided time span will be used to create the `TimeSeries` object.

Each list in this object contains a `DataFrame`. Different `TimeSeries` objects may have `DataFrame` objects that vary in rows and columns but within the `TimeSeries` object all `DataFrame` are uniform. Each `DataFrame` row is identified by a `PID` (polygon identifier). If a `polygons` argument is not provided to the `extractTimeSeries` function, the entire object will be given a `PID` of 1. In the event a `polygons` argument is provided, each polygon is given a number of 1 to n polygons in the order in which the polygons are stored in the `polygons` variable. With the use of the options `xlim` and `ylim`, it is possible to clip polygons during this process. In such cases, it is possible to have a data frame that contains missing `PID` values (Figs. 2.6a and b). Each data frame in the `TimeSeries` contains a column for each function that the user has provided in the `functions` argument. These columns store results for each given polygon function combination for all of the `ncdfData` slices see figure 2.6.

```
## load sample ncdfData object
data(sst)

## create a PolySet with two polygons
polys <- data.frame(
    PID=c(rep(1, 4), rep(2, 4)),
    POS=c(1:4, 1:4),
    X=c(155, 160, 150, 180,  0, 20, 20,  0),
    Y=c( 75,  50,  10,  85, 20, 20, 40, 40))
polys <- as.PolySet(
    polys, projection="LL")

## create a time series object that
## contains a summary for each of the
## two polygons
ts <- extractTimeSeries(sst, polygons=polys)
```

(a) The code to create a `TimeSeries` object. The function `extractTimeSeries` uses `sum`, `mean`, and `sd` by default.

```
$`2001-02-01`
  PID     sum      mean        sd
1   1 1190.98  2.802306 7.5412064
2   2 1729.63 15.040261 0.8559354

$`2001-03-01`
  PID     sum      mean        sd
1   1 1152.40  2.711529 7.4471030
2   2 1733.61 15.074869 0.7494353

$`2001-04-01`
  PID     sum      mean       sd
1   1 1254.69  2.952212 7.716375
2   2 1825.51 15.874000 0.721301

$`2001-05-01`
  PID     sum      mean        sd
1   1 1535.88  3.613835 8.1701254
2   2 2082.60 18.109565 0.5885894
```

(b) The resulting `TimeSeries` object created by the sample code in (a).

Figure 2.5: Sample code to create a `TimeSeries` object and the resulting object.

```
## load ncdfData object
data(sst)

## create a PolySet with three polygons
polys <- data.frame(
    PID=c(rep(1, 4), rep(2, 4), rep(3, 4)),
    POS=c(1:4, 1:4, 1:4),
    X=c(155, 160, 150, 180,   0,  20,  20,
          0,  45,  75,  65,  35),
    Y=c( 75,  50,  10,  85,  20,  20,  40,
         40, 80, 90, 75, 65))
polys <- as.PolySet(polys, projection="LL")

## create a time series object that contains
## one summary for each polygon that is not
## clipped by the xlim/ylim argument
ts <- extractTimeSeries(sst, polygons=polys,
    xlim=c(0, 100), ylim=c(35, 60))
```

(a) Code that creates a `TimeSeries` object. The function `extractTimeSeries` accounts for both PolySet's polygons and the `xlim` and `ylim` arguments.

```
$`2001-02-01`
  PID     sum     mean        sd
1   2 1196.96 14.59707 0.4590395

$`2001-03-01`
  PID     sum     mean        sd
1   2 1203.55 14.67744 0.3992889

$`2001-04-01`
  PID     sum     mean        sd
1   2 1269.96 15.48732 0.3930226

$`2001-05-01`
  PID     sum     mean        sd
1   2 1459.84 17.80293 0.3553732
```

(b) The resulting `TimeSeries` object created by the sample code in (a). Notice that the X/Y limits caused the first and third polygons to be clipped.

Figure 2.6: Sample code to create a `TimeSeries` object that combines a PolySet with `xlim` and `ylim` arguments.

3 Usage Patterns

This chapter describes some common usage patterns and aims to further explain the package's functionality. PBSsatellite's functionality greatly simplifies the otherwise complex operations.

The first example describes how to create a time series plot from imported satellite data to visualize changes over time. The second describes how to leverage a related package, PBSmapping, to create a complex polygon of British Columbia's coastline. After creating this polygon, it is possible to extract, from a `ncdfData` object, the data points that overlap with the ocean. Following the extraction, this resulting `ncdfData` object could be processed as in the first example to create a time series plot for the coastal region. Finally, the last example describes how to convert a sequence of files from version 4 of the Hierarchical Data Format (HDF) to NetCDF. Following the conversion, PBSsatellite can import the data into R.

3.1 Creating time series plots

When studying satellite data, the ability to quickly visualize trends for a specific geographic region can improve the efficiency of data analysis. In PBSsatellite, the `extractTimeSeries` function returns a `TimeSeries` object. This object has a straightforward structure (Section 2.2), and using the PBSsatellite function `listToDF`, can be converted into a data frame and easily plotted using built-in R functions.

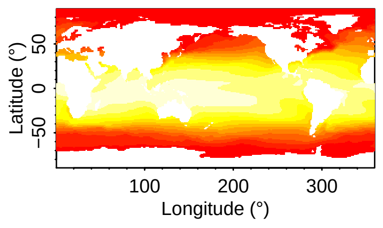
Consider the first slice of the sea-surface temperature data set (`sst`) included with the PBSsatellite package (Fig. 3.1a). Suppose that the user wants to extract time series data for each hemisphere and plot the mean sea surface temperature for each hemisphere. The PBSmapping commands

```
> g <- makeGrid(x=c(0, 360), y=c(-90, 0, 90), addSID=FALSE)
> print(g)
  PID POS   X   Y
1   1   1   0 -90
2   1   2 360 -90
3   1   3 360   0
4   1   4   0   0
5   2   1   0   0
6   2   2 360   0
7   2   3 360  90
8   2   4   0  90
```

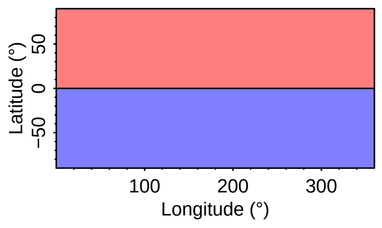
create a PolySet with one polygon for each hemisphere (Fig. 3.1b). Superimposing the two polygons (Fig. 3.1b) over the sea-surface temperatures (Fig. 3.1a) with the commands

```
> plot(sst, slice=1)
> addPolys(g, col=adjustcolor(c("blue", "red"), alpha.f=0.5))
```

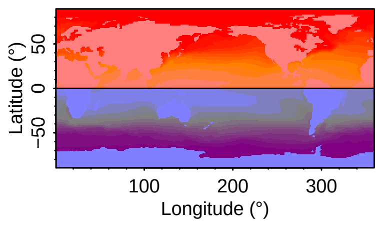
produces Fig. 3.1c and clearly shows the relationship between the polygons and the data set.



(a) The sst data set bundled with PBSsatellite.



(b) The PolySet generated by makeGrid.



(c) The PolySet representing the two hemispheres superimposed on the sst data set.

Figure 3.1: Input data used in this example.

Given this input data, the PBSsatellite function extractTimeSeries can extract summary data for each polygon from each slice of the sst object. The command

```
> extractTimeSeries(sst)
```

will generate Fig. 3.2a. Although such a TimeSeries object follows rather directly from the input data, it is not especially amenable to plotting and further analysis. The PBSsatellite function listToDF simplifies further processing by using the list element names to collapse the data frames into a single data frame while generating a new column for the names (Fig. 3.2b).

Given the data frame from listToDF, built-in R functions can generate a time series plot (Fig. 3.3). This figure provides code for the complete process that includes creating one polygon for each hemisphere, creating a TimeSeries object, and plotting the collapsed time series object.

```
$`2001-02-01`
  PID      sum     mean       sd
1   1 357483.4 15.46141 10.73330
2   2 246355.0 11.67670 11.95005

$`2001-03-01`
  PID      sum     mean       sd
1   1 355020.1 15.35488 10.92622
2   2 247695.1 11.74022 12.07278

$`2001-04-01`
  PID      sum     mean       sd
1   1 342895.3 14.83047 11.02855
2   2 255777.8 12.12332 12.28897

$`2001-05-01`
  PID      sum     mean       sd
1   1 325321.3 14.07038 10.92656
2   2 271396.7 12.86362 12.42489
```

(a) Sample list produced by `extract-TimeSeries`.

```
       names PID      sum     mean       sd
1 2001-02-01   1 357483.4 15.46141 10.73330
2 2001-02-01   2 246355.0 11.67670 11.95005
3 2001-03-01   1 355020.1 15.35488 10.92622
4 2001-03-01   2 247695.1 11.74022 12.07278
5 2001-04-01   1 342895.3 14.83047 11.02855
6 2001-04-01   2 255777.8 12.12332 12.28897
7 2001-05-01   1 325321.3 14.07038 10.92656
8 2001-05-01   2 271396.7 12.86362 12.42489
```

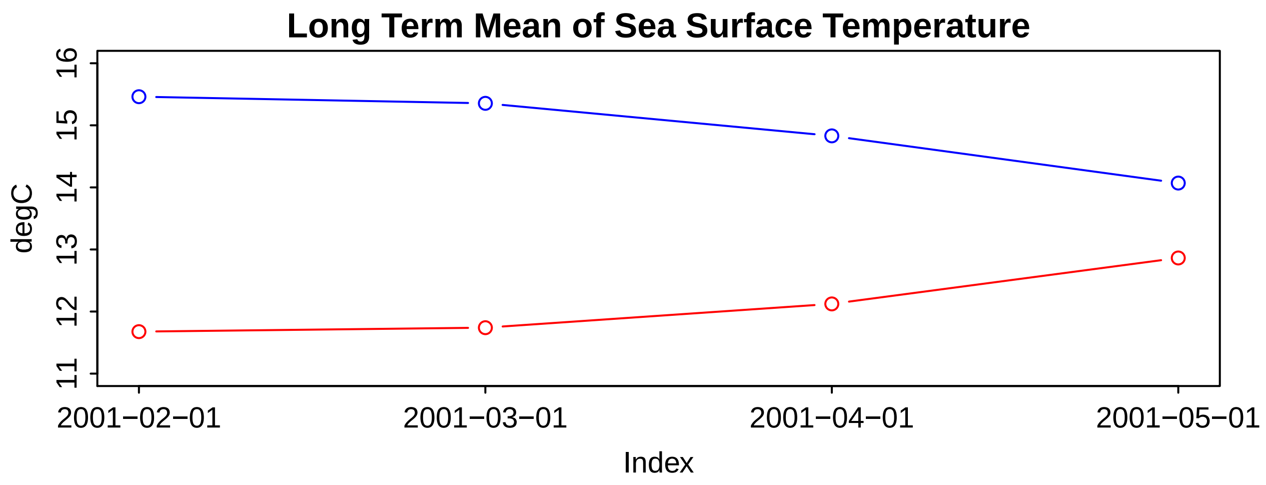
(b) Sample data frame returned from `listToDF` when given the list in (a).

Figure 3.2: Conversion from a list produced by `extractTimeSeries` to a data frame using the function `listToDF`.

3.2 Working with coastlines

Satellite data sets often include points spanning the entire globe, and in many cases, these readings include both land and water. On the other hand, analysis may focus on a specific geographic area, e.g., only measurements for the water within a region. This section provides an example of selecting a region of interest (a coastline) and excluding land measurements from the `ncdfData` object.

In this scenario, a user wants to perform sea-surface temperature analysis on the coastal region of British Columbia (BC). Consider the sea-surface temperature data set (`sst`) included with PBSsatellite, which was previously used in Sec. 3.1 (Fig. 3.1a). The PBSmapping commands



(a) A time series plot showing the mean sea surface temperature for data within the southern (blue) and northern (red) hemispheres.

```
## load ncdfData object
data(sst)

## create a PolySet with a polygon
## for each hemisphere
polys <- makeGrid(
    x=c(0, 360), y=c(-90, 0, 90),
    addSID=FALSE, projection="LL")

## create a time series object
tsList <- extractTimeSeries(
    sst, polygons=polys)

## convert the time series object into
## a data frame
tsDF <- listToDF(tsList)

## set up some plot parameters
par(mgp=c(1.7, 0.4, 0),
    mar=c(2.7, 2.7, 1.5, 1.5),
    tck=c(-0.02), cex=0.9)

## plot mean for the southern hemisphere
## first (without x axis)
plot(tsDF[tsDF$PID == 1, "mean"],
     type='b', xaxt='n', col='blue',
     ylim=c(11, 16),
     ylab=attributes(sst)$dataUnits)
lines(tsDF[tsDF$PID == 2, "mean"],
      type='b', col='red')

## create appropriate x-axis labels and
## add a title
axis(1,
     at=1:nrow(tsDF[tsDF$PID == 1, ]),
     lab=as.Date(names(tsList)))
title(main=attributes(sst)$dataType)
```

(b) The code used to generate the time series plot shown in (a).

Figure 3.3: Time series plot.

```
> bcCoast <- data.frame(PID=c(rep(1, 7)), POS=c(1:7),
>                      X=c(223, 226, 235, 238, 238, 226, 223),
>                      Y=c( 58,  53,  48,  48,  50,  60, 59.5))
> bcCoast <- as.PolySet(bcCoast, projection="LL")
> print(bcCoast)
  PID POS   X    Y
1   1   1 223 58.0
2   1   2 226 53.0
3   1   3 235 48.0
4   1   4 238 48.0
5   1   5 238 50.0
6   1   6 226 60.0
7   1   7 223 59.5
```

create a PolySet containing a polygon with seven vertices that overlies both land and sea within the BC coastal region (Fig. 3.4a). PBSmapping includes the joinPolys function, which can join one or more PolySets using a logical operation, and this function can be used to subtract the BC coastline and its islands from the seven-vertex polygon to exclude land readings. The following PBSmapping commands

```
> data(worldLLhigh)
> bcComplex <- joinPolys(bcCoast, worldLLhigh, operation="DIFF")
```

perform the subtraction and produce the polygon shown in Fig. 3.4b.

The PBSsatellite commands

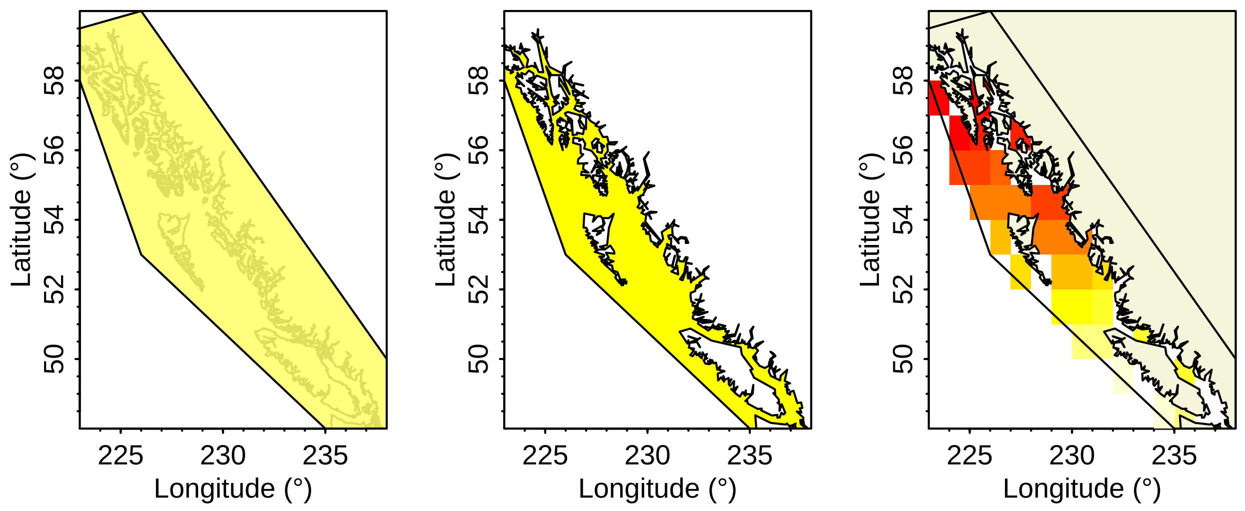
```
> data(sst)
> sstBcCoast <- clipRegion(sst, polygons=bcComplex)
```

clip data from the existing data set (sst). The resulting data set contains sea-surface temperature readings for the coloured portions of Fig. 3.4c. Note that regions without colour have no numerical value.

3.3 HDF to NetCDF to ncdfData Conversion

As stated in section 1.1 HDF file formats are lacking sufficient R libraries, for these reasons PBSsatellite is currently working with NetCDF files. On of the goals with PBSsatellite was to create a platform for satellite analysis that expands beyond the boundaries of a singular data format such as NetCDF. The more data sets that are compatible with this package the more powerful it will become, the quality of satellite analysis strongly depends on the quality and variety of satellite data available. For these reasons providing users with the ability to convert data sets into a supported file format is fundamental.

HDF files are known to provide high resolution data and they also offer a large variety of data, this made the HDF to ncdfData conversion an obvious choice for this usage pattern. Because NetCDF is currently the only supported file format for PBSsatellite we took a conversion approach to make HDF files compatible. By converting HDF to NetCDF files we are able to accommodate HDF data files without adding more library dependencies to the package. These



(a) Arbitrary polygon used in this example.

(b) Polygonal result of subtracting (a) from the BC coast (`worldLLhigh`) with PBSmapping's `joinPolys` function.

(c) Sea surface temperature (SST) data clipped using the polygon in (b).

Figure 3.4: The steps in isolating the sea surface temperatures (SSTs) within an arbitrary polygon.

converted NetCDF files can then be transformed into `ncdfData` objects.

3.3.1 Conversion Software

Here are two software packages that are necessary to complete the the HDF to `ncdfData` conversion.

Software designed by NCAR Command Language (NCL) provides NetCDF conversion functionality for multiple satellite files at once, including HDF to NetCDF conversation. This software also provides functionality for other file types to be converted to NetCDF.

A strong benefit of this software is that it provides version for Windows, Mac, and Linux systems with user friendly step by step installation instructions and examples. The software installation instructions can be found at `https://www.ncl.ucar.edu/Download/`. Navigate halfway down the page to a section labelled "Download source code or the appropriate binaries for your system."

The second software package that aids in the HDF to `ncdfData` conversion is developed by the NetCDF Operators (NCO) and provides NetCDF merging functionality. NCO additionally provides a variety of additional tools such as ones for renaming and creating dimensions which are fundamental for this conversion process.

As well as NCL, NCO also provides software versions for Windows, Mac, and Linux systems. Users can download NCO and access installation instructions here `http://nco.sourceforge.net/` find the section "Get NCO Binary Executables" which will walk you through a step by step installation process.

3.3.2 Problems with Converted NetCDF Files

Here is the list of problems that we will solve in our conversion process. The solutions to the following problems are documented in section 3.3.3.

HDF files contain information for one date at a time: This means if we converted these NetCDF files to `ncdfData` objects they only contain a single slice which creates limitations. Previous NetCDF files we used used contained data over a time period, the data for these dates were merged together and stored in an array. For example, the data component of a NetCDF file could access different slices in time with the following operation: data[, , 1] (first slice) or data[, , 2] (second slice). With the current conversion this data this component is lacking and will need to be created, this will be done by merging the new NetCDF files together.

Merged NetCDF files contain the following inconsistencies with when compared to native NetCDF files.

Time attributes are incorrectly formatted: The time unit attribute in these converted files is spread over multiple parameters such as a day, hour, minute, seconds. Where as the time attribute in native NetCDF files is formatted in a single time units attribute such as "days since 2006-01-01 00:00:00."

Time dimension is absent: With a correct time attribute a dimension must be specified. This dimension holds a 1:1 ratio per sheet in the NetCDF file, or for each file we have merged together i.e, if we have merged 3 hdf files together this array will be of size 3. It is mandatory to know when slice in a `ncdfData` has occurred, without a time dimension time series analysis is not possible.

Latitude and Longitude coordinate sequences are absent: The new NetCDF files do not contain proper X and Y sequence components see figure 2.1 for more information on these sequences. Without these components it is not possible to create a `ncdfData` because points of data are lacking locations in geographic space.

Missing value attribute is not properly formatted: The missing value argument, while it is included in the in the new NetCDF file it is not named properly. On import the ncdf4 library automatically detects the missing value argument and converts them all occurrences of these values to `NA`. In order for plots and data analysis to be accurate it is essential to properly format the missing values.

With the additional software package NCO we can successfully fix all of the above issues and are able to make the newly created NetCDF files fully functional with our package. This whole process is possible without adding extra R software or library dependencies.

3.3.3 HDF to NetCDF to `ncdfData` Conversion Process

This section provides a step-by-step description of how HDF data can be imported into PBSsatellite. The process involves converting the HDF files into NetCDF files, which are subsequently merged and used to create an `ncdfData` object.

This conversion is described in four parts. Parts one to three use the operating system's command line with the required software mentioned in section 3.3.1.[1] Part four uses R and completes the conversion by creating a `ncdfData` object.

At the start of the process, assume that three HDF files exist in a directory with the following filenames: `20150702.hdf`, `20150706.hdf`, and `20150709.hdf`.[2] These three related HDF files hold data separated by three days each, with the first file having the date July 2, 2015. It is important that all of the HDF files are from the same data source as attempts to merge files with varying resolutions and/or units will likely fail. If the file names did not contain necessary information for the conversion, e.g., dates or units, the command `ncl_filedump` can be used to gather more information about the file, e.g., Fig. 3.5.

```
$ ncl_filedump 20150702.hdf
 Copyright (C) 1995-2015 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 6.3.0
 The use of this software is governed by a License Agreement.
 See http://www.ncl.ucar.edu/ for more details.

Variable: f
Type: file
filename:        20150702
path:        20150702.hdf
   file global attributes:
      crwhdf_version : 1.0
      cwhdf_version : 3.4
      ...
      start_time : 14400
      start_time_unit : seconds since 00:00:00 UTC
      begin_date : 2015-06-29
      begin_time : 04:00:00 UTC
      stop_date : 2015-07-02
      stop_time : 04:00:00 UTC
      ...
```

Figure 3.5: Abbreviated output from the command `ncl_filedump` showing the date and time of collection.

Part 1: Convert HDF files to NetCDF

The program `ncl_convert2nc` performs the HDF to NetCDF conversion. Further documentation and examples can be found at https://www.ncl.ucar.edu/Document/Tools/ncl_convert2nc.shtml.

The following command

```
$ ncl_convert2nc *.hdf -c 'Comment: Converted 3 HDF version 4 files to NetCDF'
```

[1] We tested these commands on both an Ubuntu Linux and a Mac OS X machine. The commands may need to be adapted for a Windows machine.

[2] These files were downloaded from http://data.nodc.noaa.gov/crw/tsps50km/sst/2015/. The original filenames, e.g., `sst.night.field.50km.n19.20150702.hdf`, have been abbreviated for clarity.

will convert all of the files with the `hdf` file extension within the current directory, and it will produce NetCDF files within the same directory. The `-c` argument will create a comment within the new NetCDF files. In our example, the preceding command will convert `20150702.hdf`, `20150706.hdf`, and `20150709.hdf` to produce `20150702.nc`, `20150706.nc`, and `20150709.nc`, respectively.

Part 2: Merge New NetCDF Files

After converting individual HDF files to the NetCDF format, the individual NetCDF files must be merged into a single NetCDF file containing an array of slices (Section 3.3.2). To simplify the commands in Part 3, we recommend that you create the output file in a different directory than the input files, e.g., a subdirectory named `out`. The following command

```
$ ncecat 20150702.nc 20150706.nc 20150709.nc out/20150702-20150709.nc
```

will merge all of the files with the file extension `nc` within the current directory, and it will produce the output file `20150702-20150709.nc` within a directory named `out`. In the preceding command, the input filenames were listed explicitly to ensure that they appeared in the correct order. Depending on the filenames, you may be able to use a shortcut. If the following command

```
$ echo *.nc
20150702.nc 20150706.nc 20150709.nc
```

lists the files in the correct order, then the `ncecat` command line can safely be abbreviated to

```
$ ncecat *.nc 20150702-20150709.nc
```

Part 3: Add Missing Attributes

At this stage, we have a single NetCDF file (`20150702-20150709.nc`) that contains the data of all the source NetCDF files. Before we can import this file into an R-based `ncdfData` object, we must add some fundamental dimensions to it:

- an integer `time` dimension for units since an epoch,
- a `lon` dimension for longitude values in the grid,
- a `lat` dimension for latitude values in the grid, and
- an integer `missing_value` attribute that specifies the value used for missing data.

The integer `time` dimension will provide a timestamp for each slice within the merged file. We recommend that you use `ncl_filedump` to manually find the appropriate attributes by inspecting one of the input NetCDF files, e.g.,

```
$ ncl_filedump 20150702.nc
        ...
        pass_date_unit : days since 1 January 1970
        pass_date : 16615
        ...
```

In the sample above, the string `pass_date_unit` describes the epoch and `pass_date` provides an appropriate integer for the `time` dimension. At this time, note the string used for the epoch; you will not need it until Part 4. To extract all of the integer timestamps, use the following command

```
$ TIMES=($(ls *.nc | xargs -n1 ncl_filedump | grep "pass_date :" | egrep -o '+'))
```

and determine whether the command succeeded by listing the times with the command

```
$ echo $TIMES[@]
```

In our example, the command produces the output `16615 16618 16622`.

Given that the variable `TIMES` now contains the timestamps, we can add them to the merged NetCDF file with the following command

```
$ ncap2 -Oh -s "defdim(\"time\", ${#TIMES[@]}); \
              time[time]={$(IFS=,; echo "${TIMES[*]}")};" \
    out/20150702-20150709.nc out/20150702-20150709.nc
```

In the preceding command, `defdim(\"time\", ${#TIMES[@]});` defines a new dimension named `time` with a fixed size equal to the number of times in the variable names `TIMES`. The fixed size should also correspond to the number of files that were merged together. The argument `out/20150702-20150709.nc` appears twice on the command line: the first occurrence refers to the input file and the second the output file. In this particular case, the command will overwrite the file `out/20150702-20150709.nc` with a new file containing the time dimension.

The next step in this part involves involves adding the longitude (`lon`) and latitude (`lat`) dimensions to the NetCDF file. We recommend that you use `ncl_filedump` to manually determine the distance between points in the newly created NetCDF file `out/20150702-20150709.nc`.

```
$ ncl_filedump out/20150702-20150709.nc
      ...
      easternmost_longitude : 179.75
      westernmost_longitude : 179.75
      northernmost_latitude : 85.25
      southernmost_latitude : -80.25
      spatial_description : The rows of the data array are
        oriented in west-east direction and columns in north-south
        direction.  Each element (pixel) is 0.5 by 0.5 degree in
        size. The first element (0,0) is at the northwest corner of
        the coverage area.  The southernmost_latitude,
        northernmost_latitude, westernmost_longitude, and
        easternmost_longitude attributes give the locations of the
        outer edges of the boundary pixels.
      ...
      spatial_resolution_row :  0.5
      spatial_resolution_column :  0.5
      ...
      latitude = 331
      longitude = 720
```

Note that the attribute names vary between NetCDF files. For example, we have observed `Longitude_Step` and `Latitude_Step` in place of `spatial_resolution_column` and `spatial_resolution_row`, respectively. Look for attributes that describe the data layout, too, e.g., the preceding `spatial_description` attribute.

In the output above, note that westernmost_longitude and easternmost_longitude contain an error. Given that these boundaries represent the outer edges of boundary pixels (spatial_description), the western-most pixel center would be -179.5 and the eastern-most pixel center would be 179.5. The sequence -179.5, -179.0, -169.5, ..., 179.5 contains 719 points, one less than the expected 720. To determine the correct range of longitude points, we revisited the data source web site: http://coralreefwatch.noaa.gov/satellite/metadata/crw_sst_50km_xml_2003_format_20110103.txt. This page describes

> Each grid is 0.5 degree latitude by 0.5 degree longitude in size, centered at latitudes of from 80.0S northward to 85.0N and at longitudes of from 180W eastward to 179.5E.

The following commands will create and display the size of longitude and latitude dimensions.

```
$ LON=($(seq -180 0.5 179.5))
$ echo ${#LON[@]}
720
$ LAT=($(seq 85 -0.5 -80))
$ echo ${#LAT[@]}
331
```

Once these values are correct, the commands

```
$ ncap2 -Oh -s "defdim(\"lon\", ${#LON[@]}); \
                lon[lon]={$(IFS=,; echo "${LON[*]}")};" \
   out/20150702-20150709.nc out/20150702-20150709.nc
$ ncap2 -Oh -s "defdim(\"lat\", ${#LAT[@]}); \
                lat[lat]={$(IFS=,; echo "${LAT[*]}")};" \
   out/20150702-20150709.nc out/20150702-20150709.nc
```

will create the dimensions in the NetCDF file.

In the last step of this part, an attribute named missing_value may need to be added to the NetCDF file. The utility ncl_filedump can help identify an appropriate value for this attribute:

```
$ ncl_filedump out/20150702-20150709.nc
         ...
         missing_value :        -7777
         ...
         _FillValue :        -7777
         ...
```

Note that the missing value attribute may appear under other names, too, e.g., Fill. In this particular case, the value -7777 is a placeholder for missing values in the data matrix.

After identifying the value for the missing attribute, the following command can be used to set the attribute:

```
$ ncatted -O -a missing_value,,c,i,"-7777" out/20150702-20150709.nc
```

Even if a correctly-named attribute exists, the preceding call to ncatted can be used without causing any harm. It is important to use the name missing_value because the R library ncdf4 will look such an attribute when converting missing values to NA.

Part 4: Create ncdfData object

The preceding parts have produced a NetCDF file, e.g., `out/20150702-20150709.nc` that can be imported into R. For this part, use the PBSsatellite package within R.

The following command will create a `ncdfData` and complete the conversation from HDF to `ncdfData`:

```
> ncdfData <- read.ncdfData("out/20150702-20150709.nc",
                            Ux="degrees", Uy="degrees",
                            Utime="days since 1970-01-01",
                            dataUnits="x100 Celsius",
                            dataType="SST")
```

Note that the units for both the X and Y dimension are specified using the argument `Uy` and `Uy`, respectively. The `Utime` argument uses the epoch noted earlier in subsection 3.3.3. In order for the library to correctly interpret the string, the date `days since 1 January 1970` must be rewritten as `days since 1970-01-01`. In some cases, the R package `ncdf4` cannot detect data units; for this reason, specify `dataUnits` and `dataType` to ensure `ncdfData` has the correct information. The `dataUnits` argument specifies the units for each point of data in the data set, and the `dataType` effectively specifies the title for the data set.

The output from the command-line program `ncl_filedump` can provide hints for how to set these various arguments. For more information about `ncdfData` attributes, see section 2.1.

4 PBSsatellite functions

`PBSsatellite`	*Plotting and Statistical Analysis of Satellite Data*

Description

This software creates a fundamental data object known as `ncdfData`. A `ncdfData` object is created from NetCDF satellite data files and introduces a standard in which all `ncdfData` objects follow. This standard allows for a myriad of NetCDF files the ability to become compatible with our packages functionality. NetCDF files can be converted into `ncdfData` objects regardless of differences in length, attributes, data type, and NetCDF version.

The `ncdfData` object allows users to manipulate satellite data in multiple ways such as resolution scaling, region clipping, and erroneous value removal. PBSsatellite simplifies data extraction allowing users to either indicate dates or date spans of interest. With the ease of data extraction users are able to use the PBSsatellite function `extractTimeSeries` which provides the ability for complex statistical analysis. Users can choose from both standard R functions such as sd, mean, sum etc. Additionally users have the ability to create functions locally that will create output more relevant to a specific area of research. PBSsatellite also offers visualization abilities such as creating time series plots and general satellite data plots.

Author(s)

Nicholas Lefebvre & Nicholas Boers

See Also

`PBSmapping`

`assessMissingData` *Assessment of Missing Data*

Description

Create an assessment of missing data in each `ncdfData` slice.

Usage

```
assessMissingData(ncdfData, tlim=NULL, xlim=NULL, ylim=NULL,
      polygons=NULL, include.lowest=TRUE)
```

Arguments

`ncdfData`	`ncdfData` used for missing data assessment (required).
`tlim`	start date and end date of assessment. If `tlim=NULL`, then assess all slices.
`xlim`	range of X-coordinates.
`ylim`	range of Y-coordinates.
`polygons`	PolySet containing a complex region to apply assessment.
`include.lowest`	see `clipRegion`.

Details

It is common with satellite data for data sets to have missing values. This function is useful to indicate to the user whether a given data set is relatively complete or incomplete. It is also common for satellites to have a bad reading for a given duration, e.g., a week, thus the function is applied to every slice. The user can determine which slices are complete enough for their usage.

Value

Numeric vector containing a percentage of missing data for each slice in `ncdfData`.

Author(s)

Nicholas Lefebvre

See Also

extractSlices, clipRegion.

Examples

```
## load ncdfData object
data(sst)
## get missing data assessment
md <- assessMissingData(sst)
print(md)

## user a polygon for missing data assessment
## create 2 polygons
polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
                   X=c(155, 160, 150, 180, 0, 20, 20, 0),
                   Y=c(75, 50, 10, 85, 20, 20, 40, 40))
md <- assessMissingData(sst, polygons=polys)
print(md)
```

clipRegion	*Clip an existing ncdfData object*

Description

Clip the region of a `ncdfData` object. Clipping is applied to all slices in `ncdfData` for the specified region.

Usage

```
clipRegion(ncdfData, xlim=NULL, ylim=NULL, polygons=NULL, include.lowest=TRUE)
```

Arguments

`ncdfData`	`ncdfData` to be clipped (required).
`xlim`	range of X-coordinates for data slice(s).
`ylim`	range of Y-coordinates for data slice(s).
`polygons`	data slice(s) dimensions will be created to match one or more `polygon` dimensions.
`include.lowest`	logical ignored unless user specifies `xlim/ylim`: if `TRUE`, includes points that fall on `min(xlim)` and/or `min(ylim)` but not points that fall on `max(xlim)` and/or `max(ylim)`. If `FALSE`, it does the opposite, includes points that fall on `max(xlim)` and/or `max(ylim)` but not points that fall on `min(xlim)` and/or `min(ylim)`. If `NULL`, includes all boundary points.

Details

In most cases data sets contain information spanning the whole world; therefore, it is useful for the user to be able select a region that better suits their needs. The user will be able to select a geographical area by specifying `xlim` and/or `ylim` arguments, e.g., Northeast Pacific. For more complex selection a user is able to select a geographical area based on `polygons`, e.g., Georgia Strait.

Clipping will be applied to every `ncdfData` slice.

In situations where `xlim` and/or `ylim` as well as a polygon are provided, `xlim` and `ylim` clipping with be clipped first from `ncdfData`, the `polygon` clipping will be applied to the result.

Value

The `clipRegion` method returns a new `ncdfData` object containing geographically modified slices from an existing `ncdfData`.

Author(s)

Nicholas Lefebvre

See Also

`extractSlices`.

Examples

```
## load ncdfData object
data(sst)
## load worldLL polygons for displaying
data(worldLL)

## clip region based on xlim and ylim
ncdfDataClip <-clipRegion(sst, xlim=c(190, 320), ylim=c(5, 80),
                                                 include.lowest=NULL)
## print newly clipped ncdfData object
print(ncdfDataClip)

## plot ncdfData object
plot(ncdfDataClip, slice=1)
addPolys(worldLL, col="beige")

## clip region based on xlim, ylim, and polygons
## create 2 polygons
polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
                   X=c(155, 160, 150, 180, 0, 20, 20, 0),
                   Y=c(75, 50, 10, 85, 20, 20, 40, 40))

ncdfDataClip <-clipRegion(sst, xlim=c(.5, 300), ylim=c(-50, 90),
polygons=polys, include.lowest=NULL)

## print newly clipped ncdfData object
print(ncdfDataClip)

## plot ncdfData object
plot(ncdfDataClip, slice=1)

## add some polygons to show the clipped region
addPolys(polys, border="blue", lwd=2)
addPolys(worldLL, border="gray")
```

extractSlices *Extract ncdfData slice(s)*

Description

Extract slices from a ncdfData object.

Usage

```
extractSlices(ncdfData, slices=NULL, dates=NULL, tlim=NULL)
```

Arguments

ncdfData	ncdfData from which to extract slices (required).
slices	numeric vector containing the slices to extract from ncdfData.
dates	vector of date strings containing specific dates of slices to extract from ncdfData.
tlim	vector of date strings (length 2) which creates a time range. Slice that fall within this time range will be extracted from ncdfData.

Details

User must specify one of the three arguments slices, dates, or tlim. A new ncdfData object will be created containing the slices that the user has indicated using one of the three extraction methods.

Value

A ncdfData object containing slices indicated by the extraction method.

Author(s)

Nicholas Lefebvre

See Also

assessMissingData, clipRegion.

Examples

```
## load ncdfData object
data(sst)

## extract slices based on date strings
## create dates object
dates <- c("2001-03-01", "2001-05-01")
newNcdfData <- extractSlices(sst, dates=dates)
print(newNcdfData)

## extract slices based date range/tlim
## create tlim object
tlim <- c("2001-02-04", "2001-07-02")
newNcdfData <- extractSlices(sst, tlim=tlim)
print(newNcdfData)

## extract slices based on slices
newNcdfData <- extractSlices(sst, slices=c(2,3))
print(newNcdfData)
```

`extractTimeSeries` *Extract ncdfData Time Series*

Description

Create a time series of `ncdfData` based on a given location, using specified functions.

Usage

```
extractTimeSeries (ncdfData, xlim=NULL, ylim=NULL,
                              polygons=NULL, functions=c("sum", "mean", "sd"),
                              na.rm=TRUE, tlim=NULL, combine=1, by=NULL,
                              include.lowest=TRUE)
```

Arguments

`ncdfData`	`ncdfData` to extract time series from (required).
`xlim`	range of X-coordinates.
`ylim`	range of Y-coordinates.
`polygons`	complex range of coordinates from which to extract time series. If more than one polygon a time series is created for each polygon. Polygons will be be specified by a PID (Polygon Identifier).
`functions`	vector of function strings, where each function accepts a numeric vector and produce a single numeric value.
`na.rm`	Boolean or Boolean vector that indicates whether NA's should be omitted; if a vector, should match one-to-one with the functions.
`tlim`	start date and end date of when to begin and end a time series.
`combine`	integer number of slices to be combined into each call to each function in `functions`, e.g., given `ncdfData` with 6 slices, a `combine` value of 2 will produce time series statistics for three times, where each of its times considers the data from two source slices.
`by`	integer number indicating whether the function should skip slices, e.g., produce time series statistics for every second slice in `ncdfData`.
`include.lowest`	see `clipRegion`.

Details

In the case of a `xlim/ylim` argument without a `polygons` argument, the resulting time series data will have a single subregion identifier equal to 1. In the case of `polygons` (one or more), the resulting time series will contain a subregion identifier equal to the corresponding PID from `polygons`. If `polygons` do not fall within the specified `xlim/ylim` arguments these subregions will be missing from the resulting time series.

For each slice in `ncdfData`, the function will determine which points fall within the region(s) of interest and will pass these points (as a vector) into each of the listed `functions`.

If a `combine` value is provided that is not a factor of `length(ncdfData)` (# of slices), slices will be removed from the tail of the `ncdfData` object in order to accommodate the `combine` value.

Value

A list of date named data frames. The data frames contain a PID for each `polygon` that exists inside of `xlim` and `ylim`. If `xlim` and/or `ylim=NULL functions` will take the full span of `ncdfData`. Each polygon contains a row in the data frame and will have columns for every function in `functions`.

If `polygons=NULL` there will be only one PID for the entire `xlim/ylim` of `ncdfData`.

Author(s)

Nicholas Lefebvre

See Also

`clipRegion`, `EventData`.

Examples

```
## load ncdfData object
data(sst)

## create a time series based on full map
ts <- extractTimeSeries(sst)
print(ts)

## create a time series based on 2 polygons
## create polygons
polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
                    X=c(155, 160, 150, 180, 0, 20, 20, 0),
                    Y=c(75, 50, 10, 85, 20, 20, 40, 40))
ts <- extractTimeSeries(sst, polygons=polys)
print(ts)
```

`listToDF`	*Convert a List to a Data Frame*

Description

Convert a list such as a TimeSeries into a data frame.

Usage

```
listToDF(lst, newColumnName="names")
```

Arguments

`lst`	`list` to be converted to a data frame.
`newColumnName`	name of the new column in the data frame.

Details

Converts a list into a data frame. The names of list elements are extracted from the list and are stored in a column, this column is named by the `newColumnName` argument. This simplifies a list into in a singular data frame which stores both the name of a list element and the data component associated with this list element into a data frame row.

Value

A data frame containing a row for each list element in `lst`.

Author(s)

Nicholas Boers

See Also

`data.frame`, `extractTimeSeries`.

Examples

```
## load ncdfData object
data(sst)

## create a time series based on 2 polygons
## create polygons
polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
                    X=c(155, 160, 150, 180, 0, 20, 20, 0),
                    Y=c(75, 50, 10, 85, 20, 20, 40, 40))
ts <- extractTimeSeries(sst, polygons=polys)

## turn TimeSeries list into a DataFrame
tsDF <- listToDF(ts)
print(tsDF)
```

`ncdfData` *NcdfData Object*

Description

PBSsatellite data object that contains satellite data for varying data types and spatial resolutions.

Details

A `ncdfData` object contains at least once slice. A slice contains a dated name and at least one layer known as the data layer. The data layer contains the values for the gridded satellite data in a matrix format, all data in a slice occurs at a singular time. A `ncdfData` object contains attributes for data type (title of data), sequences of X and Y coordinates, slice names, and data units. Slices can optionally hold additional layers of information that contain point for point the same data span as the the data layer, the data layer must always be the first layer. Additional layers are created with functions such as `scaleRegion` that gives the user the option to include a missing layer and/or an error layer when scaling down a `ncdfData` object.

When a `ncdfData` object has been clipped with a `polygon` argument a complex region will be represented by a `ncdfData` object. A complex region contains dimensions that vary in length of X and Y coordinates. In order to store a complex region in a matrix, slice layers use NaN as a place holder for data that has been clipped from an original `ncdfData` object. Data that is missing from the original data set and has not been clipped is represented as NA.

Value

A `ncdfData` object.

Author(s)

Nicholas Lefebvre & Nicholas Boers

See Also

`read.ncdfData`, `scaleRegion`, `clipRegion`.

plot.ncdfData	*Plot ncdfData Slice*

Description

Plot ncdfData slice.

Usage

```
  ## S3 method for class 'ncdfData'
plot(x, slice, layer = "data", xlim = NULL,
                          ylim = NULL, style = c("image", "contour"),
               projection = "LL", tck = -0.014,
               tckMinor = 0.5 * tck, ...)
```

Arguments

`x`	`ncdfData` object, location of `slice` to be plotted (required).
`slice`	`slice` to be plotted, if `NULL` first `slice` selected
`layer`	layer name to plot.
`xlim`	range of X-coordinates to plot.
`ylim`	range of Y-coordinates to plot.
`style`	`style` of the plot.
`projection`	desired projection when `PolySet` lacks a `projection` attribute; one of `"LL"`, `"UTM"`, or a numeric value. If Boolean, specifies whether to check `polys` for a `projection` attribute.
`tck`	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If `tckLab = TRUE`, these tick marks will be automatically labeled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
`tckMinor`	numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labeled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis.
`...`	additional arguments sent to `style` function.

Details

Plots a single layer of a ncdfData slice. If no slice is given assumes the first slice. If no layer is given assumes the "data" layer of ncdfData object.

Users can select a region to plot based on xlim and/or ylim arguments.

The user can select different plotting styles to plot ncdfData such as "image", or "contour".

Author(s)

Nicholas Boers

See Also

read.ncdfData, ncdfData, plotMap, addPolys.

Examples

```
## load ncdfData object
data(sst)
## load worldLL polygons from PBSmapping
data(worldLL)

## plot map using image functionality on the first slice
plot(sst, slice=1, style="image")
addPolys(worldLL)

## plot map using contour functionality on the first slice
plot(sst, slice=1, style="contour")
addPolys(worldLL)
```

print.ncdfData *Print ncdfData Object*

Description

Pring ncdfData object.

Usage

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

Arguments

x	ncdfData object to be printed (required).
...	additional printing arguments.

Details

Prints a ncdfData object.

Author(s)

Nicholas Boers

See Also

read.ncdfData, ncdfData, print.

Examples

```
## load ncdfData object
data(sst)

## print ncdfData object
print(sst)
```

read.ncdfData	*Create new ncdfData object*

Description

Create and return a `ncdfData` object. When possible, auto detect attribute names from file and inform the user. User has the ability to override any inconsistencies between NetCDF attribute names and the given `ncdfData` object attribute values.

Usage

```
read.ncdfData(filename, dataVariable=1, dataType=NULL, dataUnits=NULL,
        xlim=NULL, ylim=NULL, tlim=NULL, x=NULL, y=NULL, time=NULL,
         Ux=NULL, Uy=NULL, Utime=NULL)
```

Arguments

`filename`	path/filename of the NetCDF file to be read (required).
`dataVariable`	location of data variable within the NetCDF file `(default=1)`.
`dataType`	string representing type of data in the file (e.g., ”SST”, ”Chl”).
`dataUnits`	string representing units of dataType (e.g., ”Celsius”, ”mg m^3”).
`tlim`	range of time (slices) to import. If `tlim=NULL` then import all slices.
`xlim`	range of X-coordinates for data slice(s).
`ylim`	range of Y-coordinates for data slice(s).
`x`	name of x variable. If `xlim=NULL`, then `x="lon"`.
`y`	name of y variable. If `ylim=NULL`, then `y="lat"`.
`time`	name of time variable. If `time=NULL` then `time="time"`.
`Ux`	units of x variable.
`Uy`	units of y variable.
`Utime`	units of time variable.

Details

Creates a `ncdfData` object that can be used with other PBSsatellite functions. Where possible, this function attempts to read names from the data file, but it allows the user to override names to account for the inconsistencies between different NetCDF files. The function `read.ncdfData` makes it possible for a variety of different NetCDF formats with varying data types to become compatible.

Value

The `read.ncdfData` method creates and returns a `ncdfData` object containing attributes and data slices from a NetCDF file.

Author(s)

Nicholas Lefebvre

See Also

`clipRegion`, `extractSlices`.

Examples

```
## read in the whole NetCDF file containing the full region
path <- system.file("sst.ltm.1971-2000.nc", package="PBSsatellite")
ncdfData <- read.ncdfData(filename=path)
print(ncdfData)

## clipping the NetCDF file by dates
## create a tlim argument of date strings
dates <- c("1-02-01", "1-05-01")
ncdfData <- read.ncdfData(filename=path, tlim=dates)
print(ncdfData)

## clipping the NetCDF file by dates and region
ncdfData <- read.ncdfData(filename=path, tlim=dates, xlim=c(20, 80),
  ylim=c(-80, 10))
print(ncdfData)
```

removeAnomalousValues *Remove Anomalous Values from a ncdfData Object*

Description

Remove specified anomalies from ncdfData.

Usage

```
removeAnomalousValues(ncdfData, zlim)
```

Arguments

ncdfData	ncdfData from which to remove anomalies (required).
zlim	numeric vector containing a range of acceptable values in ncdfData slices. All values that do not fall in the range of zlim are removed from the data set and will be replaced with ncdfData objects missing value (required). NA can be used to omit part of a range in zlim.

Details

It is common with satellite data for data sets to contain values that are anomalous. This can happen due to a variety of environmental reasons. A zlim argument is required that contains a numeric vector containing a range of values that are considered valid.

Value

a new ncdfData object containing slices with removed anomalous values.

Author(s)

Nicholas Lefebvre

See Also

assessMissingData.

Examples

```
## load ncdfData object
data(sst)

## remove values less than -2 and greater than 25
newNcdfData <- removeAnomalousValues(sst, zlim=c(-2, 25))

## remove values greater than 25
newNcdfData <- removeAnomalousValues(sst, zlim=c(NA, 25))
```

`scaleRegion` *Scale ncdfData to a New Resolution*

Description

Scale ncdfData slices to a new resolution based on a scale factor.

Usage

```
scaleRegion(ncdfData, scaleFactor, fun="drop", placement="topleft",
      includeErrorMatrix=FALSE, includeMissMatrix=FALSE,
      remainder="crop", na.rm=TRUE)
```

Arguments

`ncdfData`	`ncdfData` which will be scaled by `scaleFactor` (required).
`scaleFactor`	positive or negative integer describing the scale factor. A positive integer will scale up a `ncdfData` object a negative integer will scale down a `ncdfData` object. All integers must be a power of two. A positive integer increases the number of data points by 1*`scaleFactor` in each axis, resulting in a total increase of 1*`scaleFactor^`2 data points. A negative integer reduces the number of data points to 1/`scaleFactor` in each axis for a total reduction of 1/`scaleFactor^`2 data points(required).
`fun`	string of a function used to scale down: `"mean"`, `"min"`, `"max"`, `"drop"`. When scaling up the `"repeat"` method is always used.
`placement`	string indicating placement for the computed data point: `"topleft"`, `"centre"`.
`includeErrorMatrix`	logical indicates whether an error matrix should paired with each data layer in the resulting `ncdfData` object.
`includeMissMatrix`	logical indicates whether a missing matrix should be paired with each data layer in the resulting `ncdfData` object.
`remainder`	string if `"crop"`: if `len(x)` of `ncdfData` and/or `len(y)` of `ncdfData` is not a factor of `scaleFactor` crop will remove rows and/or columns from `ncdfData` slices in order to make `len(x)` and/or `len(y)` a factor of `ncdfData`. If `"fill"`: rows and/or columns of NA values will be added on to `ncdfData` slices to make `len(x)` and/or `len(y)` a factor of `scaleFactor`.
`na.rm`	logical value which indicates if NA values in `ncdfData` slices should be removed before before calls to `fun`.

Details

It is common for satellite data to be in different resolutions. e.g., SST (1/4 degree) or Chl (1/8 degree) It is much easier to compare different data sets that are in a standardized resolution.

This function creates a new `ncdfData` object with slices converted into the new resolution. When computing new data points, the user may choose to have the computed data point placed at the `"topleft"` point's position or in the `"centre"` of the scaled points, these two options are only available for scaling down.

A negative `scaleFactor` argument can take a `fun` to perform the scaling operation. Drop is also an option that `scaleRegion` provides. `fun="drop"`: drop points that do not fall on the points of the new scaled down region.

A positive `scaleFactor` argument will use the `"repeat"` method. `fun="repeat"`: is the default and only function available for scaling up, repeat will repeat a points data 1 * `scaleFactor^2` times in order to properly increase the scale of `ncdfData` slices.

Value

A `ncdfData` object containing slices with the newly indicated resolution. If the user has specified `includeErrorMatrix=TRUE` and/or `includeMissMatrix=TRUE` the slices in the `ncdfData` object will now have an additional two layers (error and/or missing). These layers are of the same resolution (point for point) as the data layer. These additional layers will have a percent error and/or a percent of missing values that occurred when scaling down a region of data to a singular point. These optional layers are only available when a user performs a scale down operation (`scaleFactor` is negative).

A positive `scaleFactor` will increase the length of `x` and `y` attributes, a negative `scaleFactor` will decrease the length of `x` and `y` attributes. If a user provides a `placement="centre"` argument `x` and `y` attributes will be shifted towards the centre of the point when scaling down.

Author(s)

Nicholas Lefebvre

See Also

`read.ncdfData`.

Examples

```
## load ncdfData object
data(sst)

## scale down ncdfData slices by a factor of 2, using mean
```

```
sd <- scaleRegion(sst, scaleFactor=-2, fun="mean", remainder="fill")
print(sd)

## scale down ncdfData slices by a factor of 2, using drop, place result
## in the centre of the clip region
sd2 <- scaleRegion(sst, scaleFactor=-2, fun="drop", remainder="drop",
placement="centre")
print(sd2)

## scaling up ncdfData slices by a factor of 4
sd3 <- scaleRegion(sst, scaleFactor=4, fun="repeat")
print(sd3)
```

Description

This is a `ncdfData` object used in the PBSsatellite examples. It contains a data set of long term means of sea surface temperature for several months of 2001. This data set has grid spacing of 1.0 degree latitude and 1.0 degree longitude.

Format

`ncdfData`

Note

Names attribute on this ncdfData object were renamed from 01-MM-DD to 2001-MM-DD for easier understanding in example code.

Source

NOAA_OI_SST_V2 data provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their Web site at `http://www.esrl.noaa.gov/psd/`.

References

Reynolds, R.W., N.A. Rayner, T.M. Smith, D.C. Stokes, and W. Wang, 2002: An improved in situ and satellite SST analysis for climate. J. Climate, 15, 1609-1625.

`to.EventData` *Convert ncdfData Slice to EventData*

Description

Create `EventData` object from a `ncdfData` slice.

Usage

```
to.EventData(ncdfData, slice)
```

Arguments

`ncdfData`	`ncdfData` where `slice` is located. (required).
`slice`	date string or integer of `slice` location.

Details

Converts a `ncdfData` `slice` to `EventData`. `EventData` makes `ncdfData` compatible with `PBSmapping` functionality. `EventData` is used to find which data points are in a polygon and which points fall outside a polygon, known as the points in polygon problem.

Value

`EventData` with `ncdfData` `slice` information.

Author(s)

Nicholas Lefebvre

See Also

`extractTimeSeries`, `assessMissingData`, `clipRegion`, `PBSmapping`, `findPolys`.

Examples

```
## load ncdfData object
data(sst)

## convert slice to ncdfData
ed <- to.EventData(sst, slice=1)
```

References

[1] World Meteorological Organization. *Satellite Data Formats and Standards*. http://www.wmo.int/pages/prog/sat/formatsandstandards_en.php. last accessed July 9, 2015.