PBSsatellite 1.0: User's Guide

Nicholas R. Lefebvre
MacEwan University
lefebvren4@mymacewan.ca

Nicholas M. Boers
MacEwan University
boersn@macewan.ca

Lyse Godbout
Fisheries and Oceans Canada
Lyse.Godbout@dfo-mpo.gc.ca

Rowan Haigh
Fisheries and Oceans Canada
Rowan.Haigh@dfo-mpo.gc.ca

Technical Report
MACEWANU-CMPT-TR--2017-1
August 17, 2017

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 it uses PBSmapping, an existing R package, to aid in spatial analysis and the production of plots. The tools found in this package provide users with the functionality necessary to work with data from a variety of sources. Additionally, users are able to write their own data interpretation algorithms and provide them as arguments to some analysis functions within this package.

Acknowledgements

The MacEwan authors thank Dr. Lyse Godbout from Fisheries and Oceans Canada for approaching us with the business case that led to the PBSsatellite R package. Moreover, they thank both of their collaborators at Fisheries and Oceans Canada for their insight and assistance during the development of the software.

Contents

1 Introduction **3**
1.1 NetCDF dependency 3
1.2 Data sources 4

2 Data structures **7**
2.1 `ncdfData` data structure 7
2.1.1 Attributes 7
2.1.2 Structure details 8
2.2 TimeSeries data structure 12

3 Usage patterns **15**
3.1 Creating time series plots 15
3.2 Working with coastlines 17
3.3 Converting HDF to NetCDF to `ncdfData` 19
3.3.1 Conversion software 20
3.3.2 Before converting HDF files 20
3.3.3 Conversion process: HDF to NetCDF to `ncdfData` 21

4 PBSsatellite functions **27**
`PBSsatellite` 27
`assessMissingData` 28
`clipRegion` 30
`convert.ncdfData` 32
`create.ncdfData` 34
`extractSlices` 36
`extractTimeSeries` 38
`listToDF` 41
`ncdfData` 43
`plot.ncdfData` 44
`print.ncdfData` 46
`read.ncdfData` 47

removeAnomalousValues 49
scaleRegion 51
sst 54
to.EventData 55

References **57**

1 Introduction

When the development of the package PBSsatellite began in 2015, the existing R packages related to satellite data typically focused on importing, rather than analyzing, data. With these libraries, users were largely responsible for writing their own analysis functions. This package was created to address the analysis need. It provides a front end to the existing import tools, and it provides additional functions for satellite data analysis. In some cases, it complements PBSmapping, an existing R package that offers tools for spatial analysis and plotting.

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

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

Chapter 3 demonstrates complex applications of PBSsatellite's features. The first example shows the creation of a time series plot that compares, over time, sea surface temperatures for the Northern and Southern Hemispheres. The second harnesses the power of PBSmapping to create and subsequently use a complex polygon for selecting and plotting satellite data covering the BC coast. The final example guides a user through the conversion from an incompatible file format (HDF) to a NetCDF file that can be imported into PBSsatellite.

This report concludes with Ch. 4, which documents the functions found in PBSsatellite. This function documentation is also available within R's help system.

1.1 NetCDF dependency

Three formats are widely used for exchanging and storing meteorological data: (a) Extensible Markup Language (XML), (b) Network Common Data Format (NetCDF), and (c) Hierarchical Data Format (HDF) [1]. XML is substantially more verbose than the other two formats, and this verbosity leads to unnecessarily large files. With this being the case, XML was not seriously considered for integration into PBSsatellite. The two remaining formats, NetCDF and HDF, received further consideration.

We selected NetCDF over HDF primarily due to the availability and quality of R packages for importing these files. At the time of writing, the Comprehensive R Archive Network (CRAN) did not host any packages explicitly for importing HDF files. The package rgdal, available on CRAN[1], provides bindings for the Geospatial Data Abstraction Library, which can import HDF files when appropriately configured. Unfortunately, the available macOS and (reportedly) Windows versions are built *without* support for HDF. In contrast, three available NetCDF packages were hosted on CRAN: ncdf, ncdf4, and RNetCDF.[2] Therefore, we selected NetCDF over HDF for PBSsatellite.

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

Table 1.1: NetCDF Packages

Package	**Advantages**	**Disadvantages**
ncdf	• minimal requirements	• supports only NetCDF version 3
ncdf4	• supports NetCDF versions 3 & 4 • supports offsetting into data files	• requires library netcdf ($\geq$ 4.1)
RNetCDF	• none for PBSsatellite	• supports only NetCDF version 3 • requires library netcdf ($\geq$ 3.6), udunits ($\geq$ 1.11.7), or udunits2 ($\geq$ 2.1.22)

It was important to make PBSsatellite compatible with as many data sets as possible. Given the information in Table 1.1, we chose ncdf4. While ncdf4 adds one external dependency,[3] it provides support for both NetCDF version 3 and 4 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 effectively skip irrelevant data to reach desired data, making processing significantly faster.

1.2 Data sources

Before designing the `ncdfData` object (Section 2.1), eight data sets were obtained and inspected. Five of these 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 the Cli-

[1]rgdal is available at `https://cran.r-project.org/web/packages/rgdal/index.html`.

[2]ncdf is no longer available on CRAN, ncdf4 is available at `https://cran.r-project.org/web/packages/ncdf4/index.html`, and RNetCDF is available at `https://cran.r-project.org/web/packages/RNetCDF/index.html`. Additional packages suitable for importing NetCDF files appear to now be available, too.

[3]This external library, netcdf, is available for macOS, Linux, and Windows from a variety of sources. As a starting point, see the official site at `http://www.unidata.ucar.edu/software/netcdf/`. Windows users can download a binary installer that will simplify the installation.

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 (degrees Kelvin)	1
NOAA	Sea Surface Temperature (degrees Celsius)	1
NOAA	Sea Surface Temperature	error
NOAA	Sea Surface Temperature (degrees Kelvin)	1
JISAO	Chlorophyll Concentrations	1
USJGOFS	Chlorophyll Concentrations	2
CRU	Sea Surface Temperature (degrees Kelvin)	1

mate Research Unit (CRU) at the University of East Anglia, UK. These data sets were analyzed for consistencies, particularly in data location and attribute naming.

Where data sets consistently named their attributes in a particular way, this naming convention became the default for `ncdfData` attribute acquisition. For example, most NetCDF data sets use the names “lat” and “lon” to store attributes for latitude and longitude coordinate sequences, respectively. Whenever possible, PBSsatellite locates such fundamental attributes automatically. When a NetCDF data file does not follow the expected naming conventions, the user must provide attribute names to the import functions to ensure that PBSsatellite locates the correct attributes.

The data variable most commonly appeared as the first variable (Table 1.2, “Data Location”). Some data sets, however, used a different location, e.g., the USJGOFS data set used the second data variable. For this reason, the first variable is the default when creating an `ncdfData` object, but it can be overridden when necessary using the `dataVariable` argument of a PBSsatellite function named `read.ncdfData`. For example, the user could pass `dataVariable=2` to `read.ncdfData` when loading the USJGOFS data set.

The sort order used for the X and Y coordinates was also common between several data sources. Most of the data sets had increasing X (longitude from west to east) and decreasing Y (latitude from north to south) coordinates. While it is possible to reorder these coordinates after creation, the operation can be time consuming. For that reason, we adopted the most frequently encountered order: increasing X and decreasing Y. When a NetCDF file does not follow this convention, `read.ncdfData` detects the situation and reorganizes the X and Y coordinates accordingly so that the resulting object is always consistent in its ordering.

The time units within NetCDF files vary. For example, some files strictly use seconds since an epoch, whereas others use minutes or hours since an epoch. The format of this time attribute also varies greatly between data sets, e.g., “seconds since 1981-01-01 00:00:00” and “hours since 1997-1-1 1:0:0”. When creating an `ncdfData` object, the import routine performs a date conversion on these time attributes to create consistent timestamps. The creation of consistent timestamps simplifies subsequent data extraction and comparison operations. For example, prior to date conversion, a sheet of data with the epoch “seconds since 1981-01-01 00:00:00” could have erroneously received a date of “1”. After introducing date conversion, however, it correctly receives the date “1981-01-01 00:00:0**1**”.

NetCDF data sets frequently have missing data (see Section 2.1.2), and the value used to represent them varies between data sets, e.g., one data set might use -32767 and another might use -99. The ncdf4 package's function `nc_open`, which is used by PBSsatellite to read NetCDF files, automatically detects the NetCDF's missing value attribute and replaces all occurrences of the specified value with `NA`. Given this ncdf4 functionality, missing data consistently appears as `NA` within `ncdfData` objects.

One final significant inconsistency occurs with temperature units in sea surface temperature data sets. Chapter 2 describes that `ncdfData` objects contain an attribute that stores the `ncdf-Data`'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 variety of and variability within NetCDF files, this package introduces a new data type, `ncdfData`, to make the representation of data consistent within PBSsatellite. In addition to this new type, the `extractTimeSeries` function produces a well-defined data frame intended for statistical analyses. The sections that follow describe both of these data structures.

2.1 `ncdfData` data structure

The `ncdfData` data structure is the primary type used by the functions in PBSsatellite. The structure is a list of named objects (*slices*), where each slice represents satellite data from a moment in time and is named with a timestamp (a required date and an optional time in the format YYYY-MM-DD HH:MM:SS). More specifically, each slice is a list of matrices, where each matrix represents a *layer* of information. One of these layers, the data layer, is mandatory, and it contains the gridded satellite data corresponding to the timestamp of the slice. 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 error layer. These additional layers are created by the `scaleRegion` function, which is used to change the resolution of an `ncdfData` object.

2.1.1 Attributes

In addition to the R objects (lists and matrices) that must appear in an `ncdfData` object, these objects must also have a set of attributes (Table 2.1). This section describes each attribute.

Recall that an `ncdfData` object is a list of slices. Since it is a list, the conventional attribute `names` refers to the character vector that provides each slice's name. As described above, the slice names are timestamps, and naming slices in this manner allows for easy data extraction of both exact dates and date ranges. Slices are always stored in chronological order, i.e., the first slice in an `ncdfData` object is the oldest.

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

Attribute	Description
`names`	Vector of timestamps, one per slice
`dataType`	Description of the data set
`dataUnits`	Units of the data set
`x`	Vector of longitude coordinates
`y`	Vector of latitude coordinates
`class`	Type of the data structure (`ncdfData`)

The `dataType` attribute describes the data being stored, e.g., "Long Term Mean of Sea Surface Temperature". Its value is often retrieved automatically when importing NetCDF data, but the user can override the description if desired.

The `dataUnits` attribute refers to the actual units for the object's data component, e.g., "degC" or "Kelvin". As with `dataType`, it is often retrieved automatically and can be overridden. Within an `ncdfData` object, the units are consistent, i.e., the `dataUnits` attribute applies to every slice within the object.

The `x` attribute provides a numeric vector 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 labels the *rows* of a slice matrix rather than the columns. Storing the data in this manner allows for the familiar ordering of X and Y when indexing a matrix, i.e., `[X, Y]`.

The `y` attribute provides a numeric vector 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 labels the *columns* of a slice matrix rather than the rows for the reason discussed earlier.

Given the sort order of the values in the `x` and `y` attributes, the top-left corner of a map is the origin. For example, if a variable named `m` contains a matrix of data, the point `m[1, 1]` is located in the top-left corner of a plotted map.

In addition to the above attributes, an `ncdfData` object must have the class `ncdfData`. Where PBSsatellite functions expect `ncdfData` objects, they may verify the existence of this class.

2.1.2 Structure details

As introduced in Sect. 2.1, an `ncdfData` object often contains multiple time slices. A single slice can be retrieved by either name or index, e.g., the syntax `sst$"2001-02-01"` and `sst[[1]]` both retrieve the first (and oldest) slice from the `sst` data set.[1] Each slice must always have a layer (`data`) containing the satellite data, and it may contain additional layers such as the missing and/or error layer. The `scaleRegion` function creates these two additional layers when (and only when) it scales down an `ncdfData` object. The missing layer (`miss`) contains the

[1]The `sst` data set comes with PBSsatellite, and it can be loaded with the command `data(sst)`.

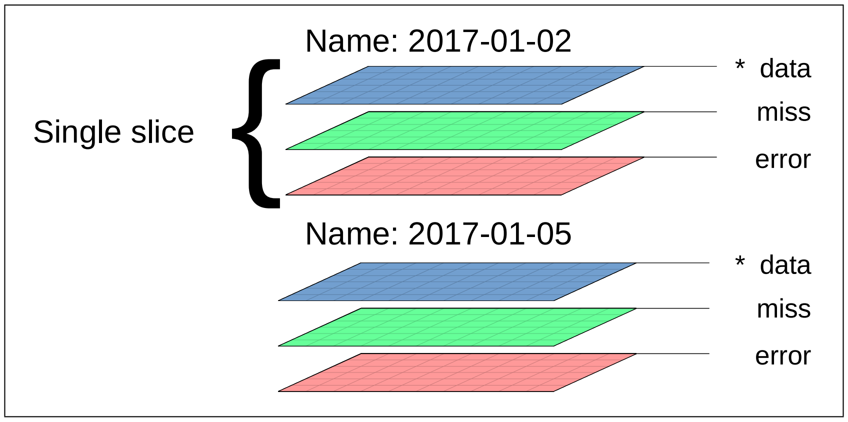


Figure 2.1: `ncdfData` Layers. (* denotes the required layer)

percentage of missing values encountered when scaling down a region. Similarly, the error layer (`error`) contains the percentage of error when scaling down a region. It is important to note that these additional layers must have the same dimensions as the data layer, i.e., they must contain point-for-point as much data (Fig. 2.1).

Each layer in an `ncdfData` object is stored as a matrix with dimensions `c(length(x), length(y))`. For slices with multiple layers, individual layers can be retrieved using list notation in the same way that slices can be retrieved.

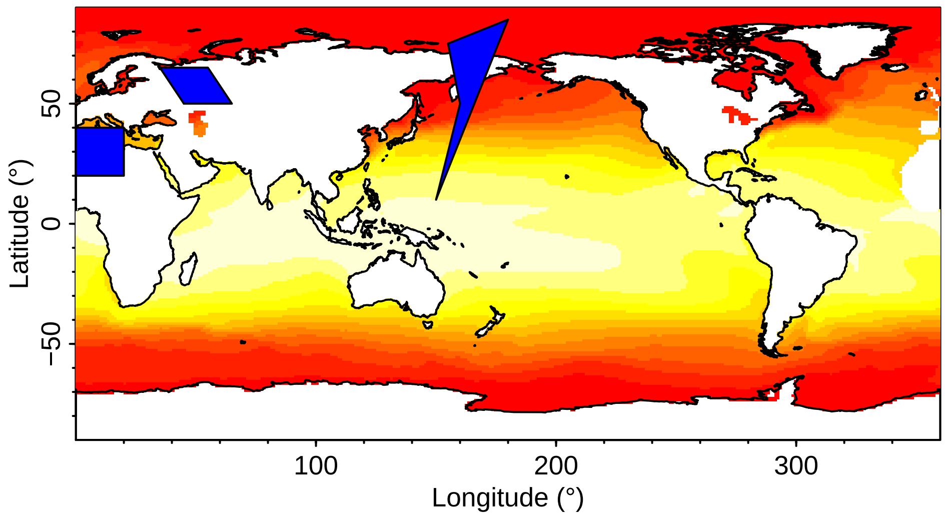
In certain data sets, such as the ones that pertain to SST (Sea Surface Temperature), it is common to have missing data values, e.g., data points on land or points obstructed by cloud cover. In an `ncdfData` object, an `NA` value is used to represent missing data.

Using matrices to store geographic data can be problematic as matrices are inherently rectangular and areas of interest may be non-rectangular. A non-rectangular region can be represented within rectangular matrices by assigning the value `NaN` (Not a Number) to points outside the region of interest.

In order to save processing time and space for such an `ncdfData` object, functions that clip slices will always produce the lowest dimension matrices (considering all layers) that can store all of the object's data (values not encoded as `NaN`). In other words, any rows or columns in an `ncdfData` object that contain `NaN` values exclusively will be removed, as these regions are no longer of importance due to clipping. For an example, see Figs. 2.2 and 2.3.

```
## load the required data
data(sst)
data(worldLL)

## plot the first SST slice along with the worldLL polygons
plot(sst, slice=1, plt=c(.08, .98, .12, 1), mgp=c(1.7, 0.4, 0))
addPolys(worldLL)

## create 3 arbitrary 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, 65, 55,  35),
    Y=c( 75,  50,  10,  85, 20, 20,  40,  40,  50, 50, 65,  65))

## create and add the PolySet
polys <- as.PolySet(polys, projection="LL")
addPolys(polys, col="blue")
```

(a) R code used to generate the plot in (b).

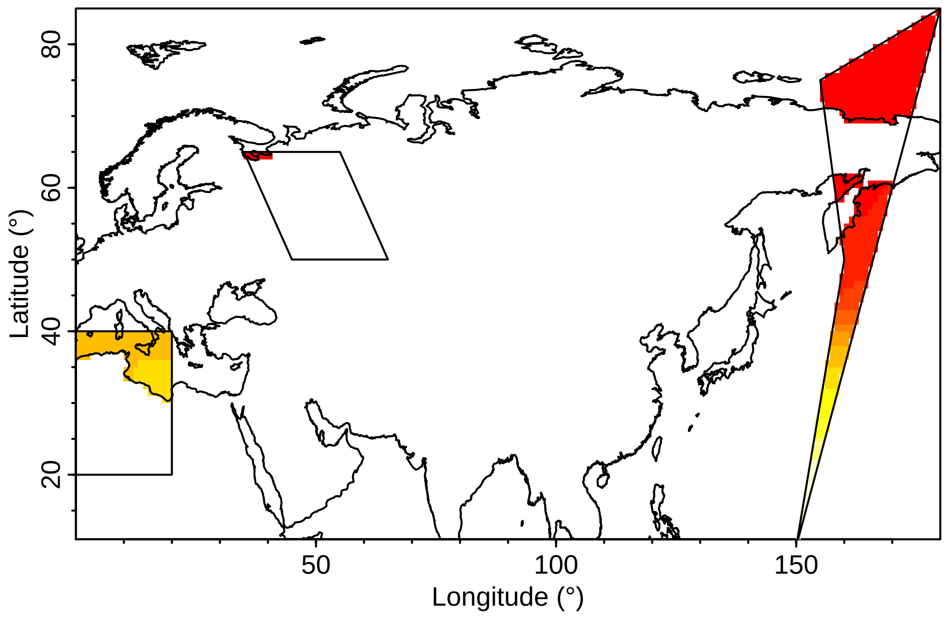


(b) Plot generated from the code in (a).

Figure 2.2: Three arbitrary polygons (blue) and the polygons from PBSmapping's `worldLL` data set (white) plotted on unclipped `ncdfData`.

```
## load the required data
data(sst)
data(worldLL)

## create 3 arbitrary 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, 65, 55,  35),
    Y=c( 75,  50,  10,  85, 20, 20,  40,  40,  50, 50, 65,  65))

## create a PolySet
polys <- as.PolySet(polys, projection="LL")

## clip the SST data set using the polygons
sst <- clipRegion(sst, polygons=polys)

## plot the first (clipped) SST slice along with the worldLL and
## arbitrary polygons
plot(sst, slice=1, plt=c(.08, .98, .12, 0.98), mgp=c(1.7, 0.4, 0))
addPolys(worldLL)
addPolys(polys)
```

(a) R code used to generate the plot in (b).



(b) Plot generated from the code in (a).

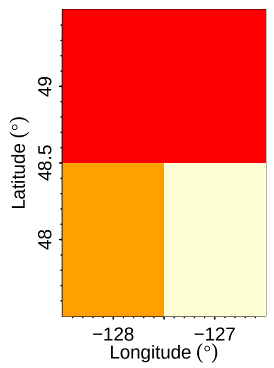
Figure 2.3: The result of clipping `ncdfData` using the polygons in Fig. 2.2. In the resulting `ncdfData` object, `NaN` values represent clipped areas outside of the three polygons and `NA` values represent areas inside the polygons that have a missing data component.

The code below shows creating, printing, and plotting a trivial ncdfData object.

```
d <- list()
d$`2017-06-16` <- list()
d$`2017-06-16`$data <-
  matrix(c(1, 1, 2, 3),
       nrow=2,
       byrow=FALSE)
attr(d, "x") <- c(-128, -127)
attr(d, "y") <- c(49, 48)
attr(d, "dataType") <- "Sample"
attr(d, "dataUnits") <- "none"
attr(d, "class") <- "ncdfData"
```

```
print(d)
NCDF data
  Data type:  Sample
  Data units: none
Slices:
  Count: 1
  First: 2017-06-16
  Last:  2017-06-16
Slice data:
  X: -128.000 to -127.000 by  1.000
  Y:   48.000 to   49.000 by  1.000
print(d$`2017-06-16`$data)
     [,1] [,2]
[1,]    1    2
[2,]    1    3
```

```
plot(d, slice=1)
```



2.2 TimeSeries data structure

A `TimeSeries` object is a list of data frames that contains an analytical summary of an ncdfData's slices over time. These objects can be created using the function extractTimeSeries.

```
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)
```

The extractTimeSeries function accepts both standard R summary functions (e.g., **sum**, **mean**, and **sd**) and user-defined summary functions. User-defined summary functions allow users to incorporate their own research and summary techniques into their analysis. The `functions` argument of extractTimeSeries determines the summary functions to apply when creating the `TimeSeries`.

Using the `polygons` argument to extractTimeSeries, a user can restrict the application of the summary function to subregions in each ncdfData slice. If the `polygons` argument is not provided, the whole ncdfData is considered one large subregion and the entire object will be summarized.

A `TimeSeries` object contains a list of data frames where each data frame summarizes a single slice of an ncdfData object. Each polygon from `polygons` results in an additional row identified by a `PID` (polygon identifier). Each summary function creates an additional column identified by the function name. Within a single `TimeSeries` object, all of the data frames are uniform.

The polygons object must be a `PolySet`.[2] Figure 2.4 shows the creation of a trivial PolySet and its use with extractTimeSeries.

Slices from the original ncdfData object may be omitted from the `TimeSeries` object using extractTimeSeries's `tlim` (time limit) argument. When the user provides a `tlim` argument,

[2]The `PolySet` object is defined by PBSmapping. After loading the PBSmapping package in R, view the `PolySet` documentation with the command `?PolySet`.

```r
## 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)

## display the resulting object (right)
print(ts)
```

(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 2.4: Sample code to create a `TimeSeries` object and the resulting object.

only the slices that fall within the provided `tlim` will be used to create the `TimeSeries` object.

The `xlim` and `ylim` arguments allow the user to limit the range of x and y coordinates that will be used in the `TimeSeries`. Similar to the `tlim` argument, only the coordinates that fall within the ranges of `xlim` and `ylim` will be used to create the `TimeSeries`. When using the `xlim` and `ylim` arguments combined with `polygons`, it is possible to clip out polygons entirely. When such clipping occurs, the resulting data frames do not contain rows for entirely clipped polygons, e.g., Fig. 2.5.

```r
## 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))

## display the resulting object (right)
print(ts)
```

(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 2.5: Sample code to create a `TimeSeries` object that combines a PolySet with `xlim` and `ylim` arguments. Note that two polygons (PIDs 1 and 3) are missing in the resulting `TimeSeries` due to a clipping operation specified by the `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 can greatly simplify 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 limit such a plot to only those data points that occur within a specific region. It leverages functionality from a related package, PBSmapping, to define a complex polygon for British Columbia's coastline. Given the polygon, it extracts the data from an `ncdfData` object that overlap with the ocean. It uses this subset of points to ultimately create a time series plot. In the last example, a sequence of files from version 4 of the Hierarchical Data Format (HDF) are converted 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 that combines time slices and can be easily plotted using built-in R functions.

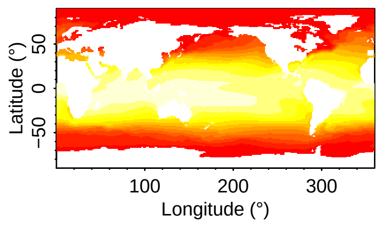
Consider the first slice of the sea surface temperature data set (`sst`) included with the PBSsatellite package (Fig. 3.1a). Suppose that the user wants to extract time series data for each hemisphere and plot the mean sea surface temperature for each hemisphere. The PBSmapping commands

```
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
```

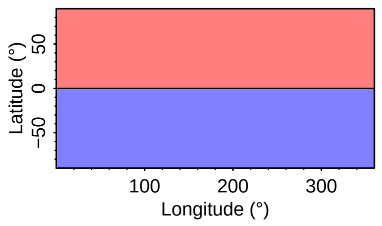
create a PolySet with one polygon for each hemisphere (Fig. 3.1b). Superimposing the two polygons (Fig. 3.1b) over the sea surface temperatures (Fig. 3.1a) with the commands

```
plot(sst, slice=1)
addPolys(g, col=adjustcolor(c("blue", "red"), alpha.f=0.2))
```

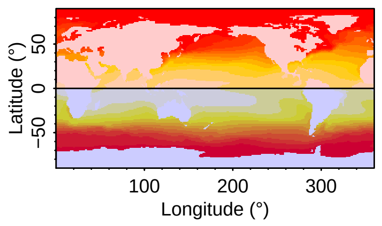
produces Fig. 3.1c and clearly shows the relationship between the polygons and the data set.



(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 attribute `names` to collapse the list of 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 `extractTime-Series`.

```
       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

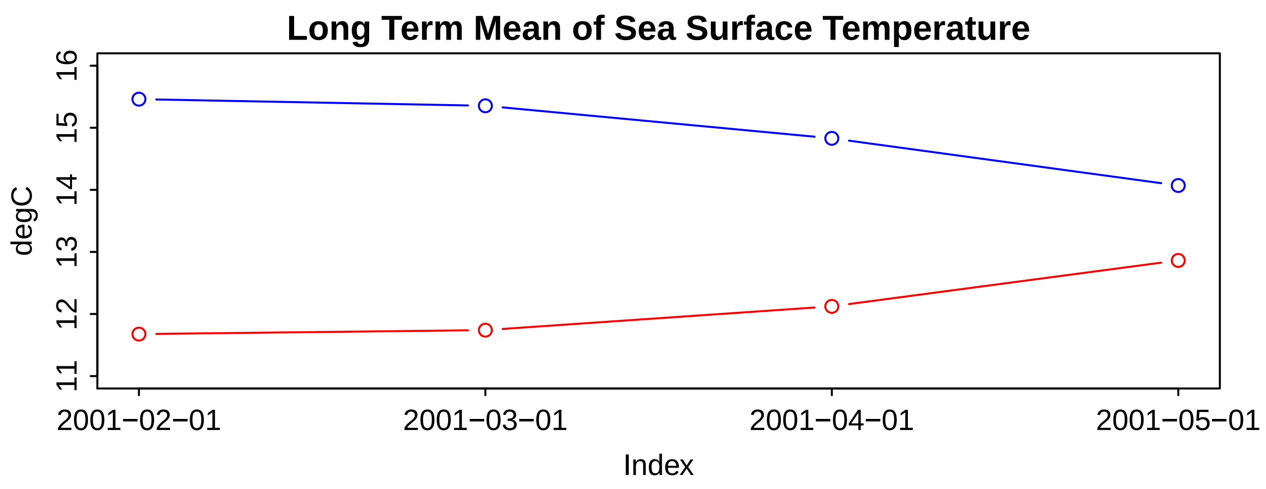
Satellite data sets often include points spanning the entire globe, and in many cases, they may contain readings from both land and water. Meanwhile, analysis may focus on a specific geographic area, e.g., only measurements for the water within a region. This section provides an example of selecting 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 Sect. 3.1 (Fig. 3.1a). The PBSmapping commands

```
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
```

```
## 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)
```

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



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

Figure 3.3: Time series plot.

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 set theoretic operations, i.e., difference, intersection, union, and exclusive-or. This function can be used to subtract the BC coastline and its islands from the seven-vertex polygon to generate a new PolySet with polygons that overlie only the water within our region of interest. The following PBSmapping commands

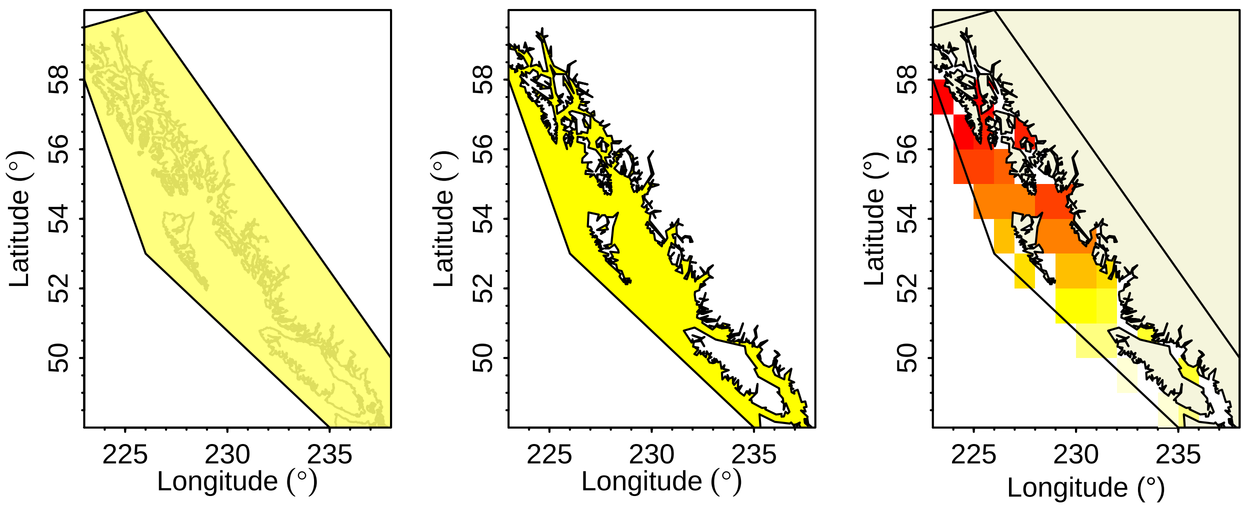
```
data(worldLLhigh)
bcComplex <- joinPolys(bcCoast, worldLLhigh, operation="DIFF")
```

perform the subtraction and produce the PolySet 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 numeric value, i.e., their value is `NaN`.



(a) Arbitrary polygon used in this example.

(b) PolySet result of subtracting the BC coast (`worldLLhigh`) from (a) 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.

3.3 Converting HDF to NetCDF to `ncdfData`

As stated in Sect. 1.1, R lacks adequate library support for importing files in the HDF file format. For that reason, PBSsatellite can only directly import NetCDF data files. At the same time, important satellite data sets are available as HDF files, and PBSsatellite users should be able to use them. This section describes how PBSsatellite users can convert HDF data sets to NetCDF data sets outside of the PBSsatellite package.

3.3.1 Conversion software

Two software packages are required to complete the the HDF to `ncdfData` conversion.

The University Corporation for Atmospheric Research provides software named the NCAR Command Language (NCL). This software supports file format conversion between many satellite data formats including from HDF to NetCDF.

The software is available as a binary download for Windows, macOS, and Linux systems with step-by-step installation instructions and examples. These installation instructions are available at `https://www.ncl.ucar.edu/Download/`. Look for the section titled "Download source code or the appropriate binaries for your system." Note that running NCL on Windows can only be done under the *Windows 10 Subsystem for Linux* or under *Cygwin/X (32-bit)*.

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 tools that allow for both renaming and creating dimensions, which are fundamental in this conversion process.

The software is available as a binary download for Windows, macOS, and Linux systems. Users can download NCO and access installation instructions at `http://nco.sourceforge.net/`. Click on the link "Executables" to see the available pre-built executables.

3.3.2 Before converting HDF files

This section provides a list of issues that must be resolved during the conversion progress. The next section, Sect. 3.3.3, documents solutions to them.

Each HDF file typically contains data from one moment in time. Thus, creating an `ncdfData` object from such a file (after converting it to a NetCDF file) would produce an object containing only a single slice. If multiple NetCDF files are merged before they are imported into PBSsatellite, we can obtain multi-slice `ncdfData` objects, but these objects will have inconsistencies (compared to typical NetCDF files) that must be resolved.

The inconsistencies in merged files include the following:

1. The time attributes may be incorrectly formatted. The time unit attribute in converted files is spread over multiple parameters such as a day, hour, minute, second rather than a single attribute. In a typical NetCDF file, the units for the single time attribute may be something such as "days since 2006-01-01 00:00:00."
2. The time dimension may be absent. After the individual parameters of the converted file are manipulated to create a single time attribute, this attribute may not be correctly identified as the time dimension. In a NetCDF file, the data dimension (sequence of slices) must have a corresponding time dimension that provides a time attribute for each slice. The time dimension is required because slices may be spaced irregularly in time.

3. Latitude and longitude coordinate sequences may be absent. The new NetCDF files do not contain proper X and Y sequence components. For more information on these sequences, see Fig. 3.5. Without these components, it is impossible to create an `ncdfData` object because points of data are lacking locations in geographic space.
4. The missing value attribute cannot be found. On import, the ncdf4 library automatically detects the missing value argument and converts each missing value to `NA`. With the new NetCDF file, the missing value argument cannot be found by the library.

```
$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 3.5: The attributes of a sample `ncdfData` object within R.

With the additional software from NCO (`http://nco.sourceforge.net/#Executables`), most of the issues above can be fixed. After addressing the issues, newly created NetCDF files are fully functional with PBSsatellite.

3.3.3 Conversion process: HDF to NetCDF to `ncdfData`

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.

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

At the start of the process, assume that three HDF files exist in a directory with the following filenames: `20170102.hdf`, `20170105.hdf`, and `20170109.hdf`.[2] These three related HDF files hold data from three separate dates, with the first file having the date January 2, 2017. It

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

[2]These files were downloaded from `https://data.nodc.noaa.gov/crw/tsps50km/sst/2017/`. The original filenames, e.g., `sst.night.field.50km.n19.20170102.hdf`, have been abbreviated for clarity.

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. 3.6.

```
ncl_filedump 20170102.hdf
 Copyright (C) 1995-2017 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 6.4.0
 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:        20170102
path:        20170102.hdf
   file global attributes:
      crwhdf_version : 1.0
      cwhdf_version : 3.4
      ...
      start_time :    0
      start_time_unit : seconds since 00:00:00 UTC
      begin_date : 2016-12-29
      begin_time : 00:00:00 UTC
      stop_date : 2017-01-02
      stop_time : 00:00:00 UTC
      ...
```

Figure 3.6: 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 line

```
for f in *.hdf; do
    ncl_convert2nc $f -c 'Comment: Converted 3 HDF files to NetCDF'
done
```

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.[3]

[3]Using an earlier version of `ncl_convert2nc`, the following command line would perform the conversion:

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

According to `ncl_convert2nc --help` for the latest version of the software, this command line continues to be correct. Unfortunately, the latest of version of the software fails when the command line specifies multiple HDF files.

The `-c` argument will create a comment within the new NetCDF files. In our example, the preceding command will convert `20170102.hdf`, `20170105.hdf`, and `20170109.hdf` to produce `20170102.nc`, `20170105.nc`, and `20170109.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 line uses `ncecat` (provided by NCO) to merge all of the files with the file extension `nc` within the current directory:

```
ncecat 20170102.nc 20170105.nc 20170109.nc out/20170102-20170109.nc
```

It produces the output file `20170102-20170109.nc` within a directory named `out`. In the preceding command, the input filenames were listed explicitly to ensure that they appear in the correct order. Depending on the filenames, you may be able to use a shortcut. If the following command

```
echo *.nc
20170102.nc 20170105.nc 20170109.nc
```

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

```
ncecat *.nc out/20170102-20170109.nc
```

Part 3: Add missing attributes

At this stage, we have a single NetCDF file (`20170102-20170109.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 ensure that it contains several essential values:

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

The command `ncdump` shows the content of a NetCDF file, and its output can help us determine whether we need to create any new dimensions. For example, the initial lines from the output of the following command line

```
ncdump out/20170102-20170109.nc
netcdf \20170102-20170109 {
dimensions:
        record = UNLIMITED ; // (3 currently)
        latitude = 331 ;
        longitude = 720 ;
```

list the dimensions in the NetCDF file. In this case, the output shows that both the `latitude` and `longitude` dimensions exist and the `time` dimension is missing.

The integer `time` dimension provides 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 20170102.nc
      ...
      pass_date_unit : days since 1 January 1970
      pass_date : 17164
      ...
```

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 need it in Part 4. If the earlier command **`echo *.nc`** listed files in the correct order, use the following command to extract all of the integer timestamps

```
TIMES=($(ls *.nc | xargs -n1 ncl_filedump | grep "pass_date :" | \
  egrep -o '[0-9]+'))
```

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

```
echo ${TIMES[@]}
```

In our example, the command produces the output `17164 17168 17171`.

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/20170102-20170109.nc out/20170102-20170109.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 named `TIMES`. The fixed size should also correspond to the number of files that were merged together. The argument `out/20170102-20170109.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/20170102-20170109.nc` with a new file containing the time dimension.

Although the NetCDF file already contains dimensions for both longitude and latitude, the following text demonstrates the creation of both dimensions. To manually create the dimensions, we recommend that you use `ncl_filedump` to manually determine the distance between points in the newly created NetCDF file `out/20170102-20170109.nc`.

```
ncl_filedump out/20170102-20170109.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_unit : degrees
            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/easternmost_longitude`, the `spatial_description`, and `longitude` contradict each other. If 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 (`longitude`). 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=($(printf "%sf " $(seq -180 0.5 179.5)))
echo ${#LON[@]}
720
LAT=($(printf "%sf " $(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/20170102-20170109.nc out/20170102-20170109.nc
ncap2 -Oh -s "defdim(\"lat\", ${#LAT[@]}); \
              lat[lat]={$(IFS=,; echo "${LAT[*]}")};" \
  out/20170102-20170109.nc out/20170102-20170109.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/20170102-20170109.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/20170102-20170109.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 the ncdfData object

The preceding parts have produced a NetCDF file, e.g., `out/20170102-20170109.nc` that can be imported into R. For this part, use the PBSsatellite package within R.

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

```
ncdfData <- read.ncdfData("out/20170102-20170109.nc",
                          Ux="degrees", Uy="degrees",
                          Utime="days since 1970-01-01",
                          dataUnits="x100 Celsius",
                          dataType="SST", x="lon",
                          y="lat", time="time")
```

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 the `ncdfData` object 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` *PBS Satellite: Plot and Analyze Satellite Data*

Description

This software stores NetCDF data within a clearly defined structure that we call `ncdfData`. Given the consistent representation in an `ncdfData` object, these objects can be used in a variety of PBSsatellite functions regardless of variations in length, attributes, data type, and version. Using the functions available within PBSsatellite, users can manipulate `ncdfData` objects in a variety of ways: they can be scaled, clipped, and summarized. PBSsatellite also allows the subsetting of data based on dates and date ranges. Users have the ability to run summary functions on `ncdfData` that allow for statistical analysis of specific regions of data over time. Users also have the ability to create their own summary functions for more specific statistical analysis.

Author(s)

Nicholas Lefebvre and Nicholas Boers

See Also

`PBSmapping`

`assessMissingData`

Assess Missing Data in an ncdfData Object

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.
`tlim`	start date and end date of assessment in a vector of length 2; if `tlim == NULL`, then assess all slices.
`xlim`	range of X-coordinates.
`ylim`	range of Y-coordinates.
`polygons`	polygons that describe the complex region to assess; if specified, the value must be a PBSmapping PolySet.
`include.lowest`	see `clipRegion`.

Details

It is common with satellite data for data sets to have missing values. This function indicates to the user the completeness of a given data set. It is also common for satellites to have bad readings for a duration, e.g., a week; thus, the function can be applied to every slice. With the output of this function, the user can determine which slices are complete enough to use.

Value

A numeric vector containing the percentage (i.e., a value from 0 to 100) describing the missing data in each slice of the `ncdfData` object.

Author(s)

Nicholas Lefebvre

See Also

`extractSlices`, `clipRegion`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## load ncdfData object
  data(sst)
  ## get missing data assessment
  md <- assessMissingData(sst)
  print(md)

  ## use 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 all slices of an `ncdfData` object to the specified region.

Usage

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

Arguments

`ncdfData`	`ncdfData` from which to clip region.
`xlim`	range of X-coordinates included in the result.
`ylim`	range of Y-coordinates included in the result.
`polygons`	data points that occur inside a polygon are included and points outside all polygons are excluded; if specified, the value must be a PBSmapping PolySet.
`include.lowest`	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, including 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 his or her individual needs. For example, a user can select a geographical area such as the Northeast Pacific by specifying `xlim` and/or `ylim` arguments. For more complex selection, a user can select a geographical area such as the Georgia Strait using `polygons`.

Clipping is applied to every `ncdfData` slice.

In situations where both `xlim` and/or `ylim` and polygons are provided, `xlim` and `ylim` clipping occurs first and then `polygons` clipping occurs on the intermediate result.

Value

A new `ncdfData` object containing modified slices from an existing `ncdfData`.

Author(s)

Nicholas Lefebvre

See Also

`extractSlices`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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")
})
```

`convert.ncdfData` *Convert ASCII Data to a NetCDF File*

Description

Convert satellite ASCII data to a NetCDF file, specifically an ncdf4 binary file.

Usage

```
convert.ncdfData(filename, zfld, nc.filename = "converted.nc",
    summary.func = sum, offset = c(0, 0), mv = NA,
    dataType = "Chlorophyll", dataUnits = "mg/m3")
```

Arguments

`filename`	name of the ASCII source file (comma-delimited, CSV-like) containing satellite gridded data.
`zfld`	string vector of fields in the source file that contain satellite measurements (e.g., ”Chl”).
`nc.filename`	name of the NetCDF binary file that user wants to create.
`summary.func`	summary function (e.g., `sum`) to aggregate multiple measurements at unique combinations of (lon, lat, time).
`offset`	coordinate (x, y) offset to adjust ASCII (X, Y) grid coordinates in case the latter is defined by some vertex other than the top left one.
`mv`	missing value indicator, usually NA in PBSsatellite.
`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”).

Details

Users sometimes prefer storing satellite data in cumbersome ASCII files. This function attempts to convert such files to a more efficient ncdf4 binary format. The function imports the ASCII file and locates the appropriate three dimensions (lon, lat, date) to create 3D arrays of z-value data. These data are then passed to the PBSsatellite function `create.ncdfData`, which creates an ncdf4 binary file.

Value

No object is returned to the user's working environment. An ncdf4 file (e.g., "converted_chla.nc") is created in the user's working directory.

Note

This function uses the `fread` function from the R package data.table to facilitate loading very large ASCII files.

Author(s)

Rowan Haigh, Research Biologist,
Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

See Also

`create.ncdfData`, `read.ncdfData`, `plot.ncdfData`

`create.ncdfData` *Create an ncdf4 File*

Description

Create a NetCDF (ncdf4) file from data.

Usage

```
create.ncdfData(filename, xvals, yvals, tvals,
    tmess = "days since 1900-01-01", zlist, mv = NA,
    dataType = "Chlorophyll", dataUnits = "mg/m3",
    longname = filename)
```

Arguments

`filename`	name of ncdf4 file to create.
`xvals`	discrete values of X (longitude) that define the spatial grid.
`yvals`	discrete values of Y (latitude) that define the spatial grid.
`tvals`	discrete values of time (dates ”YYYY-MM-DD”) to define temporal slices.
`tmess`	time message or descriptor; the most versatile is ”days since YYYY-MM-DD”, where specified date is the first in the series.
`zlist`	list of z-value 3D arrays, where each array dimension is defined by (lon, lat, date).
`mv`	missing value indicator, usually NA in PBSsatellite.
`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”).
`longname`	long name descriptor, e.g., ”Some chlorophyll data I downloaded from NOAA.”

Details

This function creates NetCDF files in the ncdf4 format. Such files load into R incredibly quickly.

Value

No object is returned to the user’s working environment. An ncdf4 file (e.g., ”some_chla.nc”) is created in the user’s working directory.

Note

See posting by user3710546 (Mar. 11, 2015 at 10:48) at:
`http://stackoverflow.com/questions/28949971`

Author(s)

Rowan Haigh, Research Biologist,
Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

See Also

`read.ncdfData`, `print.ncdfData`, `plot.ncdfData` `extractSlices` *Extract ncdfData Slice(s)*

Description

Extract slices from an `ncdfData` object.

Usage

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

Arguments

`ncdfData`	`ncdfData` from which to extract slices.
`slices`	numeric vector containing the slices to extract from `ncdfData`.
`dates`	vector of date strings containing specific dates of slices used to extract slices from `ncdfData`.
`tlim`	vector of date strings (length 2) used to extract slices from `ncdfData` based on a time range.

Details

The 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.

Value

An `ncdfData` object containing the specified slices.

Author(s)

Nicholas Lefebvre

See Also

`assessMissingData`, `clipRegion`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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 from `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.
`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 and will be labelled with the PID. If specified, the value must be a PBSmapping PolySet.
`functions`	vector of strings where each string names a function; each function must accept a numeric vector and produce a single numeric value.
`na.rm`	Boolean or Boolean vector that indicates whether NAs should be omitted; if a vector, should match one-to-one with elements of `functions`.
`tlim`	start date and end date of when to begin and end a time series.
`combine`	integer number of slices to be combined in 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 the value for each time is derived from 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 `xlim/ylim`, the resulting time series data does not include a subregion identifier. In the case of `polygons` (one or more), the resulting data set contains a subregion identifier equal to the corresponding PID from `polygons`.

For each slice in the data set, the function will determine which points fall within the region(s) of interest. It 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)` (the number of slices), slices will be removed from the tail of the `ncdfData` object in order to accommodate the `combine` value.

Value

A list of data frames, where each list element is named according to its corresponding timestamp. The data frames contain a PID for each `polygon` that exists inside of `xlim` and `ylim`. If `xlim == NULL`, `ylim == NULL`, and `polygons == NULL`, `functions` will summarize the entire region specified in `ncdfData`.

If `polygons == NULL` there will be only one PID for the entire `xlim/ylim` of `ncdfData`. If `polygons != NULL`, each polygon will have a corresponding row in the data frame. Each specified function in `functions` will have a corresponding column in the data frame.

Author(s)

Nicholas Lefebvre

See Also

`clipRegion`, `EventData`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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 list returned from `extractTimeSeries`, into a single data frame object.

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 of data frames to a single data frame. It performs the conversion by adding a new column to each individual data frame, where the column's name comes from the argument `newColumnName` and the column's value is the name of the list element. It then concatenates all of these data frames to produce a single data frame.

Value

A data frame where the list's `names` attribute has been incorporated into the data frame as a new column named according to the value of `newColumnName`.

Author(s)

Nicholas Boers

See Also

`data.frame`, `extractTimeSeries`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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

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

Details

An `ncdfData` object is, at the highest level, a list. Each element of this list is named with a timestamp, and its value is referred to as a slice. Each slice is actually a list, too, and its elements are matrices that describe data at the slice's timestamp. Each of these list elements is referred to as a layer, and the only required layer has the name ”data”.

In addition to the `names` attributes required for the lists, `ncdfData` objects contain attributes for data type (title of data), vectors of x and y coordinates, 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. Additional layers are created with functions such as `scaleRegion`, which gives the user the option to include both missing and error layers when scaling down.

An `ncdfData` object clipped with a polygon creates a complex region. In order to use a matrix to store such complex regions, layers use NaN as a place holder for data clipped from the original `ncdfData` object. Data that is missing from the original data set and has not been clipped is represented as NA.

Value

An `ncdfData` object.

Author(s)

Nicholas Lefebvre and Nicholas Boers

See Also

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

Description

Plot an ncdfData time 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.
`slice`	time slice to plot; if `NULL`, then the first slice is selected.
`layer`	layer name to plot.
`xlim`	range of X-coordinates to plot.
`ylim`	range of Y-coordinates to plot.
`style`	method for plotting the Z-value – either `"image"` or `"contour"`.
`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 labelled. 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 labelled. 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 an ncdfData layer. If no slice is given assumes the first slice. If no layer is given assumes the "data" layer of ncdfData object.

The user 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

```
local(envir = .PBSsatEnv, expr = {
  ## 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 an ncdfData Object*

Description

Print `ncdfData` object.

Usage

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

Arguments

`x`	`ncdfData` object to be printed.
`...`	additional printing arguments.

Details

Prints an `ncdfData` object.

Author(s)

Nicholas Boers

See Also

`read.ncdfData`, `ncdfData`, `print`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## load ncdfData object
  data(sst)

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

`read.ncdfData` *Create a New ncdfData Object*

Description

Create and return an `ncdfData` object. When possible, auto detect 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,
    convertMissingValues = FALSE, 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.
`dataVariable`	location of data variable within the NetCDF file.
`convertMissingValues`	if `TRUE`, convert missing values `NA` back to native form.
`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 an 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

An ncdfData object containing attributes and data slices from a NetCDF file.

Author(s)

Nicholas Lefebvre

See Also

clipRegion, extractSlices.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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 an ncdfData Object

Description

Remove specified anomalies from every `ncdfData` slice.

Usage

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

Arguments

`ncdfData`	`ncdfData` from which to remove anomalies.
`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 and replaced with the missing value that's being used for the `ncdfData` object. The value `NA` can be used to omit part of the range specified 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

An `ncdfData` object containing slices with removed anomalous values.

Author(s)

Nicholas Lefebvre

See Also

`assessMissingData`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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`.
`scaleFactor`	positive or negative integer describing the scale factor. A positive integer will scale up an `ncdfData` object; a negative integer will scale down an `ncdfData` object. All integers must be a power of two. A positive integer increases the number of data points to 1*`scaleFactor` in each axis for a total increase of 1*`scaleFactor^2`. A negative integer reduces the number of data points to 1/`scaleFactor` in each axis for a total reduction of 1/`scaleFactor^2`.
`fun`	string naming the function to used to scale down: `"mean"`, `"min"`, `"max"`, `"drop"`. When scaling up, `"repeat"` is always used.
`placement`	string indicating placement for the computed data point: `"topleft"` or `"centre"`.
`includeErrorMatrix`	logical indicates whether an error matrix should paired with each data matrix in the resulting `ncdfData` object.
`includeMissMatrix`	logical indicates whether a missing matrix should be paired with each data matrix in the resulting `ncdfData` object.
`remainder`	string specifying how to handle extra rows and/or columns at the extremeties of a matrix. If `"crop"`: if `len(x)` of `ncdfData` and/or `len(y)` of `ncdfData` is not a factor of `scaleFactor`, 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 indicates if vector values should omit NA values before call to `fun`.

Details

It is common for satellite data to be in different resolutions, e.g., a sea surface temperature data set may use 1/4 degree grid spacing while a chlorophyll data set may use 1/8 degree grid spacing. 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 to a new resolution. For this initial version, the new resolution must be an user-specified integer `scaleFactor` of the original 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 (scaling down only).

A negative `scaleFactor` argument can take a function named by `fun` to perform the scaling operation. When scaling down, the user can specify `fun="drop"` to drop points that do not fall on the points of the new scaled down region.

A positive `scaleFactor` argument will only use the `"repeat"` method. When scaling up, `fun="repeat"` will repeat a point's data 1 * `scaleFactor^2` times in order to properly increase the scale of `ncdfData` slices.

Value

An `ncdfData` object containing slices with a newly scaled region. 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 `miss`). These additional layers will have a percent error and a percentage of missing values with every corresponding point in a given slice, and they have the same resolution as the `data` layer.

Author(s)

Nicholas Lefebvre

See Also

`read.ncdfData`.

Examples

```
local(envir = .PBSsatEnv, expr = {
  ## 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 an `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 to make the example code easier to understand.

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 an ncdfData Slice to EventData*

Description

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

Usage

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

Arguments

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

Details

Converts an `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

```
local(envir = .PBSsatEnv, expr={
  ## load ncdfData object
  data(sst)

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

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.