

Package ‘RghcnV3’

May 12, 2012

Type Package

Title Global Historical Climate Network Version 3

Version 2.8

Date 2012-05-10

Author Steven Mosher

Maintainer Steven Mosher <moshersteven@gmail.com>

Depends R (>= 2.14.0),R.utils,R.oo,R.methodsS3, zoo, raster,sp,ncdf,corpcor,bigmemory,biganalytics

Description The Rghcn package provides the core functions required to download and format the GHCN V3 data and process it into temperature series and temperature anomaly series. The data is reformatted so that integration with objects in the system is straightforward. In addition, there are core functions required to download and create land masks and functions to download and import Sea Surface Temperature (SST) data.

License GPL (>= 2)

URL <http://stevemosher.wordpress.com/>

LazyLoad yes

LazyData FALSE

Repository CRAN

Date/Publication 2012-05-12 15:35:30

R topics documented:

RghcnV3-package	3
allNA	4
annualize	5
anomalize	7

asArray	8
asMts	9
asZoo	10
averageByCell	11
averageStations	12
camMask	14
checkCriteria	15
createAnomaly	15
cropInv	17
CRU.STATIONS.URL	18
CRUTEMP3.MAPS.URL	19
CRUTEMP3.RESULTS.URL	20
downloadCRU	20
downloadHadMaps	21
downloadMask	22
downloadSST	24
downloadV3	25
FILE.PARAMETERS	26
findStartEnd	27
getDemoFiles	28
getLonLat	29
getStations	30
GHCN.V3.DATA	31
GLOBE5	31
HADCRUT3.MAPS.URL	33
HADCRUT3.RESULTS.URL	34
HADSST2.RESULTS.URL	35
HADSST2.URL	35
intersectInvData	36
inverseDensity	37
isArray	39
isBigMatrix	40
isInventory	40
isMts	41
keepIf	42
OCEAN.MASK.URL	43
passesCam	44
rasterizeCells	45
rasterizeZoo	46
readChcn	47
readCruStations	48
readHadMaps	49
readHadResults	50
readInventory	51
readMask	53
readMaskDeg1	54
readQC	55
readSST	56

readV3Data	57
referenceStation	58
removeNaStations	60
solveTempAnnual	60
solveTemperature	61
solveTempMonthly	62
stationCount	63
tempStats	64
trimHeadTail	64
trimNA	65
V3.MEAN.ADJ.URL	66
V3.MEAN.RAW.URL	67
V3.TMAX.ADJ.URL	67
V3.TMAX.RAW.URL	68
V3.TMIN.ADJ.URL	69
V3.TMIN.RAW.URL	69
validData	70
validMonthCount	71
WATER.MASK.URL	71
windowArray	72

Index**74**

RghcnV3-package	<i>A package for downloading, importing and analyzing GHCN version 3</i>
-----------------	--

Description

The RghcnV3 package provides some basic tools for downloading formatting and analyzing Global Historical Climate Network temperature data. It is targeted at the version 3 release. The core functions allow for the easy creation of temperature anomalies for land surface stations. In addition, land masks are provided. Together with the raster package these functions can be used to create a global temperature index using the Common Anomaly Method or CAM. The functions provided will create a data structure that can be directly used by raster functions to create spatially weighted maps of temperatures. Other methods for 'averaging' stations have also been added where regression approaches are taken.

Details

Package:	RghcnV3
Type:	Package
Version:	2.8
Date:	2012-05-10
License:	GPL (>=2)
LazyLoad:	yes
LazyData:	FALSE

Author(s)

Steven Mosher with contributions from Tamino, RomanM and Nick Stokes

Maintainer: Steven Mosher <moshersteven@gmail.com>

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
## Not run:

# download the files
files    <- downloadV3(url = V3.MEAN.ADJ.URL)
# pass the file names to the read functions
v3Mean   <- readV3Data(filename = files$DataFilename,output = "Zoo")
inventory <- readInventory(filename = files$InventoryFile)
Anomaly  <- createAnomaly(v3Mean)
Data     <- intersectInvData(inventory,Anomaly)

## End(Not run)
```

allNA

allNA Detects Months which have no data for all stations

Description

Least squares methods require that every month have at least one station reporting, otherwise a singularity results. the solveTemp routines and other related methods will fail if you have month where no station was reporting. Typiclally this happens at the begining of time series or at the end. If you take subsets of whole collect, say a rural sample, you must ensure that there is no month where all the vales for every station are NA. This function returns a time series of Boolean indicating whether all the stations are NA (TRUE) or not. If at least one station has a report, then FALSE is returned for that month.

Usage

```
allNA(Data)
```

Arguments

Data Accepts an Array, a Mts obkect or a Zoo object

Details

This function wraps a call to apply for the various data formats and returns a simple time series indicating whether a month has any station reporting (FALSE) or all stations missing (TRUE)

Value

Retruns a times series of boolean. FALSE if at least one station reports for that month. TRUE if all stations have missing data for that month

Author(s)

Steven Mosher

annualize	<i>a function to create an annual mean from monthly data</i>
-----------	--

Description

The data used in RGhcnV3 has a monthly frequency. Creating yearly means from this is straightforward. However, the package supports three types of data structures, an Array, a zoo object and an mts object. This function takes any one of those objects and creates an annula series from it

Usage

```
annualize(Data, user.fun = NULL, na.rm = TRUE, ...)
```

Arguments

Data	A monthly series of data. It can be temperatures or anomalies. A check for the frequency of data is made. The data object can be any type of mts, zoo or Array. Internally, the function manages the change of object types.
user.fun	The user.fun defaults to NULL. This variable is provided if the user wants to create their own "NA" handlers. A "NA" handler is a function that describes how to calculate a yearly mean in the presence of "NAs". It can, for example, give you the granular control to accept only a specified number of NAs or NAs in certain seasons. A sample of a user function can be seen in keepIf . If a user defined function is used, then na.rm must be set to NULL. Parameters for a user function are passed through the , . . . mechanism.
na.rm	na.rm is set to TRUE by default. This means that when annual means are calculated, all NAs will be removed. The annual mean will be calculated if ANY of the 12 months are present. If na.rm is set to FALSE, then annual averages will only be calculated if ALL 12 months are present. If a user function is provided, na.rm must be set to NULL.
. . .	for passing parameters to the user defined function.

Details

The `annualize` function works by using the `zoo` `aggregate` function. That function is very straightforward. Like the `aggregate` function in the base package, a data structure, like a `data.frame`, is subdivided by an index function. In this case we map every month of a given year to a unique index, 1, 2, etc. Then, the `mean` function is applied using that index. So, one gets the mean by year. For most purposes setting `na.rm` to either `TRUE` or `FALSE` will suffice. However, in some cases the user may want finer control over how means are calculated in the presence of `NA`. For example, one may want the mean if and only if at least 10 months of data are present. Or, if all 4 four seasons have at least 2 months of data in each season. That can be accomplished by writing a user defined function and passing its handle to `user.fun`. The user defined function describes how to calculate a mean and can take parameters which are passed through the `...`.

Value

Returns a `zoo` series of annual averages either by taking a simple mean or by using the user defined function for taking a mean. If a time series or `mts` object is passed in, one will be passed out

Note

Thanks to Peter O'Neill for the suggestion

Author(s)

Steven Mosher

See Also

[keepIf](#)

Examples

```
## Not run:
annual <- annualize(Anomalies)
# get the averages but only if all 12 months have data
annual2 <- annualize(Anomalies, na.rm = FALSE)
# use a function called keepIf, set na.rm =NULL, pass cnt =10
# to the user.fun. That function calculates a mean if there
# are 10 or months of data in a year
annual3 <- annualize(Anomalies,user.fun = keepIf, na.rm =NULL, cnt =10)

## End(Not run)
```

`anomalize`*A function to create anomalies from temperatures*

Description

Anomalies are created by selecting a base period, usually a 30 year period. The monthly values for a station series are averaged and the result is a "normal" or mean value for that station, that month. This mean is then subtracted from every other value in the time series, thus normalizing the entire series to the average of the base period. The mean "January" temperature for the base period (1961-1990) would thus be subtracted from every January measure. This is done on a per station basis.

Usage

```
anomalize(Data, period = list(Start = 1961, End = 1990))
```

Arguments

<code>Data</code>	The function <code>anomalize</code> must be supplied with a "zoo" series, "mts" series or "Array" of actual temperatures. The series contains the entire time series of data for a station including NAs for the missing data.
<code>period</code>	<code>period</code> is a list of two elements <code>Start</code> and <code>End</code> . These are defaulted to 1961 and 1990 respectively. To specify a start and an end, whole years are required. If <code>period\$End</code> is set to 1990, then the function will ensure that ALL months in 1990 are used. It would be an error to specify 1990.917. Warnings will be thrown and the input values will be trimmed to the integer part. This behavior is different from the behavior of the <code>window</code> function.

Details

There are two ways to create anomalies. The more direct path relies on the `createAnomaly()` function. That function does take input directly in the format delivered from `readV3Data()`. That function takes care of all the processing to turn the data into anomalies and to return only those stations which satisfy the CAM criteria, for example, 15 Years of data in the CAM base period. That function effectively wraps several other function, namely: `anomalize()`; `passesCam()`. The second method of creating anomalies is to first `readV3Data()` then Then that Temperature series would be fed to `anomalize()`. Anomalies could then be plotted for each station. The final step would be to take the result of `anomalize()` and the apply the CAM (Common Anomaly Method) criteria to select the subset of stations which satisfy the criteria for inclusion in data analysis. This is performed by the function `passesCam`. Or, One can execute `passesCam` first and then run `anomalize`

Value

The function returns a series of station anomalies. For zoo and mts data The data structure puts stations in columns and anomaly data in rows. The GHCN station Ids are used as column names. Every row of data is a month of anomaly data. You can think of the structure as a collection of time series in columns. Every column is a time series of station data. For the Array data struture, the

first dimension is stations, the second dimension is months and the 3rd dimension is years. Empty stations, or stations with all NA values are removed. This happens if the anomaly period is empty.

Author(s)

Steven Mosher

See Also

[passesCam](#), [readV3Data](#), [createAnomaly](#)

Examples

```
## Not run: df <- readV3Data(filename = "myfile.dat",output = "Zoo")

Anomalies <- anomalize(TempSeries)
Anomalies <- passesCam(Anomalies)

## End(Not run)
```

asArray

Convert an mts or zoo to Array

Description

This function takes an mts or zoo series and converts it into a 3D array of stations, months, and years. The dimensions are named appropriately so that the Array works with other functions.

Usage

```
asArray(Data)
```

Arguments

Data A zoo object or a mts object with stations in columns.

Details

The function takes a 2D zoo object or mts object and creates a 3D object with stations in the first dimension, months in the second and years in the third. dimnames are set to be consistent with other objects and functions

Value

A 3D array. dimension one accesses stations and station Ids are used as dimnames. Dimension 2 is months and month abbreviations are used as dimnames. Dimension 3 is years and the dimnames is years

Note

Hat tip to Nick Stoke for his work with this data structure in his code

Author(s)

Steven Mosher

See Also

[asZoo](#); [asMts](#)

Examples

```
## Not run:  
data <- asArray(data)  
  
## End(Not run)
```

asMts

Convert a zoo series or Array to an Mts

Description

The function converts a zoo series or Array (3D) into a 3D mts object with stations in columns.

Usage

```
asMts(x)
```

Arguments

x A 2D zoo series with stations in columns or an Array with stations in the first dimension, months in the second and years in the third.

Details

The function takes a zoo series of stations or a 3D array of stations and returns an Mts like object which is required by some functions

Value

Outputs an mts object which the correct dimnames

Author(s)

Steven Mosher

See Also

[asArray](#); [asZoo](#)

Examples

```
## Not run:  
data <- asMts(data)  
  
## End(Not run)
```

asZoo

Converts an Array or Mts to Zoo

Description

Given inputs of Array or "mts" type objects a zoo series is returned. Station data is in columns.

Usage

```
asZoo(x)
```

Arguments

x An "mts" type object with station data in columns or a 3D Array with stations in the first dimension, months in the second and years in the third

Details

The function will return a properly defined zoo object when given an mts object or an Array object.

Value

returns a properly defined Zoo object

Author(s)

Steven Mosher

See Also

[asArray](#)

Examples

```
## Not run:  
data <- asZoo(data)  
  
## End(Not run)
```

 averageByCell

Average time series in a cell using regression

Description

The offset method uses all available information in the collection of temperature series to estimate the average temperature of all the stations. Unlike the Common anomaly method which relies on stations have periods where they overlap in temporal coverage, this method uses all series however long or short to determine an optimal estimate for the collection that minimizes the stations offsets from each other. The function takes a collection of time series and divides them into various cells in a raster and then performs a regression on each cell to determine the optimal estimate for that cell

Usage

```
averageByCell( inventory, Mts,tol,weights = NULL,r = GLOBE5)
```

Arguments

r	A raster object that will be used as a template to build a brick The default is the built in 5 degree earth. This can be changed to other resolutions. The raster is used to generate the proper cells for allocating stations to cells on the globe
inventory	The inventory of stations
Mts	A "mts" class object that has station data in columns.
tol	tolerance for calculating the psuedoinverse. The function at its core relies on a psuedoinverse which relies on a singular value decomposition. That function must round very small values to zero to operate effectively. This variable may require some "tweaking" in cases where their are very sparse data or very large data. <code>.Machine\$double.eps</code> is a plce to start, but in some cases values towards <code>.0001</code> or larger are required. Errors will be noticeable upon simple inspection.
weights	Set to NULL. Setting to NULL sets the weights of all data to be equal within the cell.

Details

The function operates by taking in a raster, an inventory, and a "mts" object. The function then allocates stations to raster cells and then calls the function [averageStations](#) for every cell in the grid. The result is an object that contains a time series for every cell.

Value

The function outputs a 2D zoo series. The series has a column for every grid cell. Column names identify the cell by the numbering system used by raster. This object as an attribute added to it to identify the resolution of the raster that was used to generate the cell numbers. The name of the attribute is "Res".

Note

This function is called prior to putting the cell and all its layers into a brick

Author(s)

Steven Mosher

References

<http://statpad.wordpress.com/2010/03/08/combining-stations-plan-c/>

See Also

[averageStations](#), [rasterizeCells](#)

Examples

```
## Not run:
TEXAS.DAT <- system.file("external/Texas.dat", package = "RghcnV3")
TEXAS.INV <- system.file("external/Texas.inv", package = "RghcnV3")
texas <- readV3Data(TEXAS.DAT,output = "Mts")
inv <- readInventory(TEXAS.INV)
texas <- window(texas,start =1900, end = 2010 +11/12)
DATA <- intersectInvData(inv,texas)
globe3 <- GLOBE5
res(globe3) <- 3
ZooCell <- averageByCell(r = globe3, inventory = DATA$Inventory, Mts = DATA$Mts)

## End(Not run)
```

averageStations

A function to optimally average stations

Description

The offset method uses all available information in the collection of temperature series to estimate the average temperature of all the stations. Unlike the Common anomaly method which relies on stations have periods where they overlap in temporal coverage, this method uses all series however long or short to determine an optimal estimate for the collection that minimizes the stations offsets from each other. The function takes a collection of time series and then performs a regression on to determine the optimal estimate for that collection of stations

Usage

```
averageStations(Mtsdata, tol, weights = NULL, all = TRUE)
```

Arguments

Mtsdata	A "mts" class object with multiple time series in columns
tol	tolerance for calculating the psuedoinverse. The function at its core relies on a psuedoinverse which relies on a singular value decomposition. That function must round very small values to zero to operate effectively. This variable may require some "tweaking" in cases where their are very sparse data or very large data. <code>.Machine\$double.eps</code> is a plce to start, but in some cases values towards <code>.0001</code> or larger are required. Errors will be noticeable upon simple inspection.
weights	Optional weights
all	output option for outputing all the data

Details

The function performs RomanM's offset method for anomaly regression. internally it calls the two functions `.calcOffset` and the function `.psx()`. Those files can be viewed in the package source and are not exported as functions.

Value

a list of returned	
Average	The estimated average of all time series in zoo format
Prediction	The predicted value of every time series, missing data is predicted
Residual	The residual
Offsets	The offsets for every station. The function minimizes offsets

Author(s)

RomanM, JeffId and clean up by Steven Mosher

References

<http://statpad.wordpress.com/2010/03/08/combining-stations-plan-c/>

See Also

[referenceStation](#)

Examples

```
## Not run:
TEXAS.DAT <- system.file("external/Texas.dat", package = "RghcnV3")
TEXAS.INV <- system.file("external/Texas.inv", package = "RghcnV3")
texas <- readV3Data(TEXAS.DAT,output="Mts")
inv <- readInventory(TEXAS.INV)
texas <- window(texas,start =1900, end = 2010 +11/12)

TexasAve <- averageStations(Mtsdata = texas)
```

```
## End(Not run)
```

```
camMask          create a mask temperature data
```

Description

The function returns a true/false mask for a dataset of station temperatures. If station records "pass" the cam test specified TRUE is returned otherwise false.

Usage

```
camMask(Data, criteria = list(Start = 1961, End = 1990, Years = 15, Threshold = 12))
```

Arguments

Data	A data structure of stations and their temperatures. This can be an Array object, Zoo object or MTS
criteria	A list of criteria for passing a CAM test.

Details

A data structure such as a MTS object will have station data in columns with every row holding a month of data. In certain situations the user may wish to select those stations that have "complete" records. The "camMask" is one approach to selecting stations. For example, you might select stations that had at least 10 complete years in the 1961 to 1990 period. When a data structure is passed to camMask, the function will return a mask that can then be applied to the data structure to select only those stations that pass the test.

Value

Returns a true/false mask as long as the data structure

Author(s)

Steven Mosher

Examples

```
## Not run:  
  
mask <- camMask(Data)  
Data <- Data[mask,]  
  
## End(Not run)
```


Arguments

Data	data must be an "Array", "Zoo", or "Mts" object as output by readV3Data. If the data is not in that format, the function will fail.
criteria	criteria is a list with 4 elements named Start, End, Years, and Threshold. Start defines the first year of the base period. End is the last year. Years, defines how many years of data are required in the period. Threshold defines the number of months required to "count" as a full year. The default is 15 full years in the period 1961-1990, where full year is defined as a year with all 12 months reporting.

Details

The function takes the temperature data and calculates the average temperature for each station for each month in the base period. This base period monthly mean is then subtracted from the entire time series. The net effect is the entire database is transformed into a collection of time series where the monthly mean temperature from 1961 to 1990 has been subtracted from every measure. Thus, the anomaly for every month is the departure from the norm for that station, where the norm is defined as the average during the base period. To be included in the analysis a station's record must pass the screening test. The screening test examines how many full years of data a station has in the base period (1961-1990). Full years can be specified by the Threshold variable, typically 9-12 months. That is, one could screen stations based on a criteria of havin 20 "full" years in the base period, where 10 months of data constituted a full year. see the demo "StationCounts" to understand the effect changing this parameter has.

Value

The resultant series is a collection anomalies for every station during its entire period. outpute is organized by stations id in the columns and time in the rows. So that column one contains the time series for station 1, and so forth. Missing years and months are already infilled with NA so that the result is suitable for plotting. Array type objects have their data dimensions preserved and stations are reurned in the first dimension.

Author(s)

Steven Mosher

Examples

```
## Not run:  
V3Anomaly <- createAnomaly(V3mean)  
  
## End(Not run)
```

`cropInv`*A function to select stations that fall in a Bounding Box*

Description

Every station in the inventory has an associated Latitude (y value) and Longitude (x value). Selecting subsets of an inventory can be handled by standard indexing methods on the data.frame. Since selecting stations by region is a oft performed function a simple interface is provided to do that. That interface depends upon specifying an extent from the raster package. An extent is merely a bounding box in Longitude and Latitude. The extent is used to crop the inventory of stations. It can also be used to [crop](#) a Raster* defined object.

Usage

```
cropInv(inventory, bBox)
```

Arguments

<code>inventory</code>	An inventory data.frame of stations. The data.frame must conform to certain specifications. Column 1 must be a column named Id, column 2 is a column named Lat and column 3 is a column named Lon. Latitudes and Longitudes should be in a WGS84 coordinate system. -180 to 180 for Longitude and -90 to 90 for latitude.
<code>bBox</code>	bBox must be an extent object as defined by the raster package. extent is created by passing the 4 corners of the bounding box to the function. The order must be lonMin, lonMax, latMin, LatMax. This is also refered to as xmin,xmax,ymin,ymax.

Details

This function works by operating on the underlying dataframe to return only those stations that fall within the bounding box.

Value

The function returns a data.frame of stations in the standard inventory format

Author(s)

Steven Mosher

See Also

[extent](#)

Examples

```
## Not run:
cuba <- extent(-88, -65, 16, 30)
CubaInv <- cropInv(Inventory, cuba)
Cuba <- crop(Land, cuba)

## End(Not run)
```

CRU.STATIONS.URL *A url for the CRU station data*

Description

url of the CRU station data

Usage

CRU.STATIONS.URL

Format

The format is: chr "http://www.cru.uea.ac.uk/cru/data/temperature/station-data/station-data.zip"

Details

Provides a link to the CRU station data for download

Source

<http://www.cru.uea.ac.uk/cru/data/temperature/station-data>

References

Brohan, P., Kennedy, J., Harris, I., Tett, S.F.B. and Jones, P.D., 2006: Uncertainty estimates in regional and global observed temperature changes: a new dataset from 1850. *J. Geophys. Res.* 111, D12106

Examples

```
print(CRU.STATIONS.URL)
```

CRUTEMP3.MAPS.URL	<i>Url to temperature data from CRU</i>
-------------------	---

Description

crutem3 is the latest gridded temperature product from CRU (climate research unit). The dataset linked to in this url starts in 1850 and goes up to the present day. It contains temperature anomalies for the entire land surface. The anomalies have been gridded in a 5 deg by 5 degree bin structure. The file is a netcdf type file where the varname is "temp". This data can be read directly into a raster brick object. The time variable, however, has to be set by hand into a format acceptable by raster.

Usage

```
CRUTEMP3.MAPS.URL
```

Format

The format is: `chr "http://hadobs.metoffice.com/crutem3/data/CRUTEM3.nc"`

Details

The crutem3 dataset can be used a gross validation check of work performed with RghcnV3. The source datasets for creating the final grids are slightly different so an exact match is not expected. There are also other datasets one can use such as the variance adjusted dataset.

Source

The dataset is created by CRU. The publication is http://www.metoffice.gov.uk/hadobs/crutem3/HadCRUT3_accepted.pdf Brohan et al, 2006

References

the data is located at <http://www.metoffice.gov.uk/hadobs/crutem3/>

Examples

```
print(CRUTEMP3.MAPS.URL)
```

CRUTEMP3.RESULTS.URL *A url to Crutemp3 temperature series*

Description

This is the url to the crutemp3 land temperature record. The data consists of monthly anomalies relative to a 1961-1990 baseline. The data is starts in 1850 and is updated monthly.

Usage

```
CRUTEMP3.RESULTS.URL
```

Format

The format is: `chr "http://www.cru.uea.ac.uk/cru/data/temperature/crutem3gl.txt"`

Details

The dataset contains monthly figures for anomalies and annual figures as well. In addition, a metric for the percentage of land sampled is included

Source

the data is described here <http://www.cru.uea.ac.uk/cru/data/temperature/>

References

documentation for the data is contained the following paper http://www.cru.uea.ac.uk/cru/data/temperature/HadCRUT3_accepted.pdf

Examples

```
print(CRUTEMP3.RESULTS.URL)
```

downloadCRU *A function to download CRU station data*

Description

Downloads Station data from Climate Research Unit

Usage

```
downloadCRU(url = CRU.STATIONS.URL, directory = getwd())
```

Arguments

url The URL for the CRU version of CRU stations
 directory The download directory

Details

There are two version of the CRU station data. One version on MET website and the other on CRU. This function get the data from the CRU website and unpacks it

Value

returns a handle to the unzipped CRU station data

Note

Not that this data differs in format from the data on the MET site

Author(s)

Steven Mosher

References

["http://www.cru.uea.ac.uk/cru/data/temperature/station-data/"](http://www.cru.uea.ac.uk/cru/data/temperature/station-data/)

Examples

```
## Not run:

  crufilename <- downloadCRU()

## End(Not run)
```

downloadHadMaps *A function to download temperature grids*

Description

The results of CRU temperature analysis is stored in two netcdf files. One file contains crutem3 data which is land only. The other dataset is hadcrut3 which contains a combined land/ocean data set. The grids are 5 deg by 5 degree grids running from -180 to +180 longitude and -90 to +90 latitude. Each cell contains the monthly anomaly for the grid, relative to a 1961-1990 baseline

Usage

```
downloadHadMaps(url = c(CRUTEMP3.MAPS.URL, HADCRUT3.MAPS.URL),
  directory = getwd())
```

Arguments

url	two urls are given. Only one url should be used in loading the data. Loading both datasets will require calling the function twice with the different urls. A warning is thrown if you list more than one url, and the first url in the list is used by default.
directory	the directory where you want the data downloaded to.

Details

The data is in a .nc file. That file is downloaded to the directory you select. There are times when the CRU server is not reliable. Downloading by hand is always an option. The data is up to date. It starts at 1850. The varname (see the ncdf package) for the data is "temp"

Value

The function returns a handle to the local file. This can be used to read in the data.

Author(s)

Steven Mosher

References

All data is found at this url. <http://www.metoffice.gov.uk/hadobs/>

See Also

[readHadMaps](#)

Examples

```
## Not run:
cru <- downloadHadMaps(url = CRUTEMP3.MAPS.URL, directory = getwd())
hadcrut <- downloadHadMaps(url = HADCRUT3.MAPS.URL , directory = getwd())

## End(Not run)
```

downloadMask

A function to download a land mask

Description

The function download a land mask data set from the provided URL and returns the local filename. The package provides two land mask URLs. WATER.MASK.URL and OCEAN.MASK.URL. These URLs should be loaded as the package is loaded. Otherwise the data function will load them. The masks are land percent masks where the value of the data runs from 0 to 100. See details below

Usage

```
downloadMask(url = OCEAN.MASK.URL, directory = getwd())
```

Arguments

url	The "ocean" land mask is the default. This url points to a mask that has land and all ocean bodies. The other url provided is an water/land mask. That mask is more appropriate for checking whether stations are located in an inland body of water. That is, for error checking locations. For processing a global temperature, the ocean mask should be used. It is the default.
directory	by default the file is downloaded to the working directory If you provide a directory name that does not exist it will be created.

Details

The package provides URLs to two land masks. One that looks at all bodies of water and one that only considers oceans. The value in the cells is the percent of land in the cell. 100 represents 100 percent land in the cell. 0 represents 0 percent land. The cell size is 1/4 of a degree. The entire globe is provided

Value

The function returns the full path to the local file. If you provide a different directory that will be reflected in the returned value.

Note

"All of the files in the ISLSCP Initiative II data collection are in the ASCII, or text format. The file format consists of numerical fields of varying length, which are delimited by a single space and arranged in columns and rows. All values in the land/water/sea fraction files are written as integers from 0 to 100.

The files at different spatial resolutions each contain the following numbers of columns and rows:
One degree (1d): 360 columns by 180 rows 1/2 degree (hd): 720 columns by 360 rows 1/4 degree (qd): 1440 columns by 720 rows

All files are gridded to a common equal-angle lat/long grid, where the coordinates of the upper left corner of the files are located at 180 degrees W, 90 degrees N and the lower right corner coordinates are located at 180 degrees E, 90 degrees S. Data in the files are ordered from North to South and from West to East beginning at 180 degrees West and 90 degrees North." (From the files README)

Author(s)

Steven Mosher

References

Please acknowledge the EOS DEM Science Working Group and the EOS ASTER Instrument Project at JPL for provision of the 30 arcsecond EOS AM1 DEM and the ISLSCP2 data collection when these data are used. ISLSCP2 Land/Water and Land/Sea Masks and other Ancillary Data at 0.25, 0.5 and 1 degree Spatial Resolutions

Examples

```
## Not run: water <- downloadmask(url=OCEAN.MASK.URL)
```

downloadSST

A function to download Hadley SST

Description

The function locates the HADSST2 sea surface temperature data and downloads and unpacks it. This data set currently has a start date of 1850 and data is included for every month to the present. The data is downloaded to the working directory unless directory is explicitly set otherwise. If directory does not exist, it will be created.

Usage

```
downloadSST(url = HADSST2.URL, directory = getwd(),  
            overwrite = FALSE, remove = FALSE)
```

Arguments

url	The url is predefined according to its published location at the time of testing the package.
directory	The directory where the files will be downloaded to. This defaults as the working directory. If directory does not exist, it will be created.
overwrite	If overwrite is false and the file already exists an error will be thrown. To overwrite the file, set overwrite to TRUE. The download takes a while, so overwrite is set to false by default. If you want to re download the data, then set the flag to TRUE.
remove	Remove the compressed file after decompression

Details

The file is compressed at a .gz file. That file is gunzipped and a ncdf file is unpacked.

Value

The function returns the path name to the unzipped file. This is a complete path.

Author(s)

Steven Mosher

Examples

```
## Not run: sstFile <- downloadSST()
```

downloadV3

A function to download the version 3 Temperature data

Description

The function downloads the datasets for the Global Historical Climate Network version 3, available on a public ftp. The function downloads the tar.gz data sets and decompresses the contents to a local directory.

Usage

```
downloadV3(url = V3.MEAN.ADJ.URL, directory = getwd(),
           overwrite = TRUE, remove = FALSE)
```

Arguments

url	The url is defaulted to download the "adjusted" V3 'Tave' or Tmean data. Tmean data is the data one normally works with. There are 6 different datasets that are updated on a near daily basis. Each has a different URL. This package provides the url to each in stored data. V3.MEAN.ADJ.URL, V3.MEAN.RAW.URL, V3.MAX.ADJ.URL, V3.MAX.RAW.URL, V3.MIN.ADJ.URL, V3.MIN.RAW.URL. These urls are loaded when the package loads. If not, data() will load them. That does not mean the data behind the url is loaded, but rather the url string is defined.
directory	The default directory is the working directory. After downloading the files are unpacked and stored under a dated directory. This directory will be written directly under the working directory or the directory provided by the user. If directory does not exist it will be created.
overwrite	Overwrites the compressed files (tar or gz) This is defaulted to TRUE. Since fresh files are posted every day the default is to overwrite existing copies.
remove	Removes the compressed files after decompression

Details

Using the URL provided the specified compressed files are downloaded to the directory provided. They are gunzipped using the R.utils package and then untar. This process creates a subdirectory that is dated. The date is required as these files are updated on a daily basis. Within that subdirectory there will be two files: a data file and an inventory file. They must be used together. After download and decompression a typical set up would have a subdirectory named. ghcnm.v3.0.0.20110612 with two files in it. ghcnm.tavg.v3.0.0.20110612.qca.dat ghcnm.tavg.v3.0.0.20110612.qca. Those files are the data files and the inventory files on June 12th. Downloading on a different day would collect a different dataset.

Value

The function returns a list with three values. DataFilename and InventoryFile and Date. These variables can be used to read in the local data readV3Data().

DataFilename a full path to the datafile just downloaded
 InventoryFile a full path to the inventory file just downloaded
 Date a Posix date taken from the filenames

Author(s)

Steven Mosher

Examples

```
## Not run:
files <- downloadV3(url = V3.MEAN.RAW.URL)
v3Mean <- readV3Data(filename=files$DataFilename)
Inv <- readInventory(filename=files$InventoryFile)
Anomaly <- createAnomaly(V3Data=v3Mean)

## End(Not run)
```

FILE.PARAMETERS

A dataset containing various constants required to read data

Description

The data files and inventory files of GHCN V3 differ from previous versions. Each dataset now has different columns and different column descriptions. In order to read in these datafiles certian variables such as column widths and column names must be set. Those parameters are kept in a list that is loaded when the package loads.

Usage

FILE.PARAMETERS

Format

The format is: List of 9

```
$ InvWidths : num [1:16] 11 9 10 7 31 5 1 5 2 2 ...
$ InvNames : chr [1:16] "Id" "Lat" "Lon" "Elevation" ...
$ DataWidths : num [1:51] 11 4 4 5 1 1 1 5 1 1 ...
$ DataOnly : num [1:51] 11 4 -4 5 -1 -1 -1 5 -1 -1 ...
$ QcOnly : num [1:51] 11 4 -4 -5 1 1 1 -5 1 1 ...
$ DataNames : chr [1:51] "Id" "Year" "Element" "Jan" ...
$ DataColumnns : chr [1:14] "Id" "Year" "Jan" "Feb" ...
$ QCColumns : chr [1:38] "Id" "Year" "DMF1" "DMF2" ...
$ DMFlags : chr [1:14] "Id" "Year" "DMF1" "DMF2" ...
$ QCFlags : chr [1:14] "Id" "Year" "QCF1" "QCF2" ...
$ DSFlags : chr [1:14] "Id" "Year" "DSF1" "DSF2" ...
```

Details

the various elements of the list are used when reading in the files. "widths" and "names" are provided for reading in the temperature data and the inventory data. In addition, with this release of GHCN NCDC has supplied QC flags and source flags for the temperature elements. Data columns can be extracted by using the `$DataColumns` vector, and QC flags can be extracted by using vectors as well.

Source

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README> This readme contains all the information you need to understand the files The format of the files and the description of the various flags is contained there. The names for Inventory columns has been changed slightly to be more expressive.

Examples

```
data(FILE.PARAMETERS)
str(FILE.PARAMETERS)
```

findStartEnd

Detects the first January and the last december in a time series

Description

Clipping or windowing time series data often requires detecting which year is the first full year in a dataset and which year is the last full year. Some datasources "pad" out their data to full years, while others do not. This function quickly returns the first full year (with no missing data) and the last year with no missing data

Usage

```
findStartEnd(Data)
```

Arguments

Data An Array object, Mts object or Zoo object with multiple time series

Details

The function uses `allNA` to determine the first year in the dataset where every month has at least one station report. In addition, the last year in which every month has at least one station report is returned.

Value

Returns a list of Start and End

Start the first full year where every month has at least one station
End the last full year where every month has at least one station

Note

If by change some month in the middle of a collection has a missing month, or a month with no stations, this function will not detect that.

Author(s)

Steven Mosher

getDemoFiles	<i>A utility used by demo programs</i>
--------------	--

Description

A simple function to return downloaded file names required by various demo programs

Usage

```
getDemoFiles(directory = GHCN.V3.DATA)
```

Arguments

directory The data directory

Details

Used by the demo programs to get the file names or to download the files if they are absent.

Value

Returns a list of filenames that can be used in the read* functions. Gets, data and inventories, masks and SST

Author(s)

Steven Mosher

getLonLat	<i>A convenience function to create a 2 column matrix of Longitude and Latitude</i>
-----------	---

Description

The inventory data.frame has station Latitude and Longitude as well as other metadata. When using the raster package several functions require that points be supplied in a 2 column matrix or 2 column dataframe of Longitude and Latitude. This function returns a data structure that can be used as a parameter to raster calls such as rasterize

Usage

```
getLonLat(inventory)
```

Arguments

inventory	The function is passed an inventory data.frame That data.frame should have columns named Lat and Lon. Inventories created with this package do have those names.
-----------	--

Details

This is a convenience function for creating a simple two column matrix of coordinates of stations with Longitude in column 1 and Latitude in Column 2. That order (x,y) is the order expected by functions in the raster package.

Value

The function returns a matrix of two columns. Column 1 is Longitude (-180 to 180) and column 2 is Latitude (-90 to 90). The number of rows is equal to the number of stations.

Note

For future versions the Id variable will be used as a rowname, provided that is compatible with raster calls.

Author(s)

Steven Mosher

Examples

```
## Not run: stationCoordinates <- getLonLat(inventory)
```

getStations	<i>A function to return the stations from a data object</i>
-------------	---

Description

This convenience function returns a vector of station Ids from the dataset that is passed to it. It can be passed a inventory Array, a zoo object or a mts object.

Usage

```
getStations(data)
```

Arguments

data the function takes in an inventory, a zoo object, a mts object or an Array.

Details

If the data fed to the function is a data.frame with a column named "Id" then that column is returned as a vector. If the data is a "zoo" object, then the column names are returned as a vector of Ids. The same logic is applied to mts objects and the Arrays. When these objects are created their column names are set as station Ids. This function returns those ids as numerics

Value

the function returns a vector of station Ids.

Author(s)

Steven Mosher

Examples

```
## Not run: StationsId <- getStations(anomalies)
          StationId <- getStations(Inventory)
          identical(getStations(Inventory),Inventory$Id)
## End(Not run)
```

`GHCN.V3.DATA`*Directory name for data directory*

Description

A demo program is included that allows the user to download all of the data files used by this package. That demo `demo(DemoSetup)` will create a directory and download the various datasets used in constructing a global temperature index.

Usage`GHCN.V3.DATA`**Format**

The format is: `chr "GhcnV3Data"`

Details

The data directory can be used to store various files that are downloaded from the internet for this package. The directory will be written below the current working directory when the `demo(DemoSetup)` command is issued at the console

Examples

```
print(GHCN.V3.DATA)
```

`GLOBE5`*A blank raster object of the globe*

Description

This is a predefined global raster with 5 degree by 5 degree cells. Its used as a default parameter for the function `rasterizeAnomaly`. It can be simply scaled to any resolution by using the raster `res` command. For example a 1 degree grid is created by setting `res` to 1. Like so, `res(GLOBE5)<-1`

Usage`GLOBE5`

Format

```

The format is: Formal class 'RasterLayer' [package "raster"] with 15 slots ..@ file :Formal class
'.RasterFile' [package "raster"] with 9 slots .. ..@ name : chr ""
.. ..@ datanotation: chr "FLT4S"
.. ..@ byteorder : chr "little"
.. ..@ nodatavalue : num -Inf
.. ..@ nbands : int 1
.. ..@ bandorder : chr "BIL"
.. ..@ offset : int 0
.. ..@ toptobottom : logi TRUE
.. ..@ driver : chr ""
..@ data :Formal class '.SingleLayerData' [package "raster"] with 11 slots .. ..@ values : logi(0)
.. ..@ offset : num 0
.. ..@ gain : num 1
.. ..@ inmemory : logi FALSE
.. ..@ fromdisk : logi FALSE
.. ..@ isfactor : logi FALSE
.. ..@ attributes: list()
.. ..@ haveminmax: logi FALSE
.. ..@ min : num Inf
.. ..@ max : num -Inf
.. ..@ band : int 1
..@ legend :Formal class '.RasterLegend' [package "raster"] with 5 slots .. ..@ type : chr(0)
.. ..@ values : NULL
.. ..@ color : NULL
.. ..@ names : NULL
.. ..@ colortable: NULL
..@ history : chr(0)
..@ title : chr(0)
..@ extent :Formal class 'Extent' [package "raster"] with 4 slots .. ..@ xmin: num -180
.. ..@ xmax: num 180
.. ..@ ymin: num -90
.. ..@ ymax: num 90
..@ rotated : logi FALSE
..@ rotation :Formal class '.Rotation' [package "raster"] with 2 slots .. ..@ geotrans: num(0)
.. ..@ transfun:function ()
..@ ncols : int 72
..@ nrows : int 36
..@ crs :Formal class 'CRS' [package "sp"] with 1 slots .. ..@ projargs: chr "+proj=longlat +da-
tum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
..@ layernames: chr ""
..@ zname : chr ""
..@ zvalue : chr ""
..@ unit : chr ""

```

Details

This raster object is loaded when the package loads. it can be used as a 'blank' form for creating new rasters.

Examples

```
Globe1deg <- disaggregate(GLOBE5, fact = 5)
```

HADCRUT3.MAPS.URL	<i>Url to temperature data from CRU</i>
-------------------	---

Description

Hadcrut3 is the latest gridded temperature product from CRU (climate research unit). The dataset linked to in this url starts in 1850 and goes up to the present day. It contains temperature anomalies for the entire land/ocean surface. The anomalies have been gridded in a 5 deg by 5 degree bin structure. The file is a ncdf type file where the varname is "temp". This data can be read directly into a raster brick object. The time variable, however, has to be set by hand into a format acceptable by raster.

Usage

```
HADCRUT3.MAPS.URL
```

Format

The format is: `chr "http://hadobs.metoffice.com/hadcrut3/data/HadCRUT3.nc"`

Details

The hadcrut3 dataset can be used a gross validation check of work performed with RghcnV3. The source datasets for creating the final grids are slightly different so an exact match is not expected. There are also other datasets one can use such as the variance adjusted dataset.

Source

The dataset is created by CRU. The publication is http://www.metoffice.gov.uk/hadobs/crutm3/HadCRUT3_accepted.pdf Brohan et al, 2006

Examples

```
print(HADCRUT3.MAPS.URL)
```

HADCRUT3.RESULTS.URL *A url to Hadcrut3 temperature series*

Description

This is the url to the hadcrut3 land temperature record. The data consists of monthly anomalies relative to a 1961-1990 baseline. The data starts in 1850 and is updated monthly. This is a combined land/ocean dataset as opposed to crutem3 which is land only

Usage

HADCRUT3.RESULTS.URL

Format

The format is: `chr "http://www.cru.uea.ac.uk/cru/data/temperature/hadcrut3gl.txt"`

Details

The dataset contains monthly figures for anomalies and annual figures as well. In addition, a metric for the percentage of land sampled is included. It is up to the date. Care should be taken because of the way future months are included. For example, in June, there will be data elements for July through Dec. They will typically be zero. Future releases will use the system clock to trim the data to only read in data that is before clock time

Source

the data is described here <http://www.cru.uea.ac.uk/cru/data/temperature/>

References

documentation for the data is contained the following paper http://www.cru.uea.ac.uk/cru/data/temperature/HadCRUT3_accepted.pdf

Examples

```
print(HADCRUT3.RESULTS.URL)
```

HADSST2.RESULTS.URL *A url to Hadley SST temperature series*

Description

This is the url to the Hadley SST temperature record. The data consists of monthly anomalies relative to a 1961-1990 baseline. The data is starts in 1850 and is updated monthly.

Usage

```
HADSST2.RESULTS.URL
```

Format

The format is: `chr "http://www.cru.uea.ac.uk/cru/data/temperature/hadsst2gl.txt"`

Details

The dataset contains monthly figures for anomalies and annual figures as well.

Source

the data is described here <http://www.cru.uea.ac.uk/cru/data/temperature/>

Examples

```
print(HADSST2.RESULTS.URL)
```

HADSST2.URL *A Url to the HADSST2 dataset*

Description

A url that points to the dataset. This Url is used to download the data

Usage

```
HADSST2.URL
```

Format

The format is: `chr "http://hadobs.metoffice.com/hadsst2/data/HadSST2_1850on.nc.gz"`

Details

The dataset is gz compressed. It is a 5 degree by 5 degree dataset, from -180 to +180, -90 to 90. The data is temperature anomalies. The base period is 1961-1990. The first date in the file is Jan 1850.

Source

From the website description: "The SST data are taken from the International Comprehensive Ocean-Atmosphere Data Set, ICOADS, from 1850 to 1997 and from the NCEP-GTS from 1998 to the present. HadSST2 is produced by taking in-situ measurements of SST from ships and buoys, rejecting measurements which fail quality checks, converting the measurements to anomalies by subtracting climatological values from the measurements, and calculating a robust average of the resulting anomalies on a 5 deg by 5 deg degree monthly grid. After gridding the anomalies, bias corrections are applied to remove spurious trends caused by changes in SST measuring practices before 1942. The uncertainties due to under-sampling have been calculated for the gridded monthly data as have the uncertainties on the bias corrections following the procedures described in the paper."

References

[urlhttp://www.metoffice.gov.uk/hadobs/hadsst2/rayner_etal_2005.pdf](http://www.metoffice.gov.uk/hadobs/hadsst2/rayner_etal_2005.pdf) The above paper describes the dataset

Examples

```
print(HADSST2.URL)
```

intersectInvData *A function to reconcile a collection of data and an Inventory*

Description

The Inventory and temperature data are tied together by the station Ids. If you edit the inventory to remove or select certain stations then the temperature data must contain the same stations. Typically, one would select a subset of the inventory, for example all rural stations, and then find those stations in the temperature data. This function accepts the three basic dataset as an input (Array, Zoo and Mts) along with a inventory. The input Inventory and the input data are both reconciled and on output both are adjusted to have matching stations. Stations are also sorted so that the Ids are in a common order.

Usage

```
intersectInvData(inventory, Data)
```

Arguments

inventory	An inventory data.frame containing station Ids
Data	The temperature or anomaly data

Details

It's often quicker to prune the data prior to creating anomalies. So, for example one can "window" the data or you can select a subset of the stations before you turn the temperature data into an anomaly series. For example, you could select only the "rural" stations from the inventory. Then call this function `intersectInvData()`. The advantage in this is that fewer stations will have to be run through the anomalize process.

Value

the function returns a list with two elements. An inventory element and a data element

Inventory	a inventory with stations matching the temperature series
Array	an Array object if an Array is input
Zoo	a Zoo object if a Zoo is input
Mts	a Mts object if a Mts is input

Author(s)

Steven Mosher

Examples

```
## Not run:
# we select only certain stations and time periods
# then we reconcile the two data sources.
vdata <- readV3Data(filename="yourfile", output = "Array")
ruralInv <- inv[which(inv$Rural == "R"),]
vdata <- windowArray(vdata, start = 1920, end =1996)
DATA <- intersectInvData(ruralInv,vdata)
anomalies <- anomalize(DATA$Array)
```

```
## End(Not run)
```

inverseDensity *calculates the inverse density of station reports*

Description

Stations are allocated to grid cells on the planet. then for every month in the entire time series a count is generated for the number of valid reports in that cell for that month. This value then scales the area of the grid cell. The output is a series of weights for every gridcell on the globe for every month in the series. The area is effectively divided by the number of reports so that series are weighted inversely to their density on an area basis. many stations in a small area would receive little weight. One station in a large area would get a large weight.

Usage

```
inverseDensity(inv, Data, r = GLOBE5)
```

Arguments

inv	the station inventory which is used for Lat/lon
Data	the temperature data in Array format. If you need to pass in a Zoo type object covert using asZoo
r	a raster layer. The resolution of the raster determines the gridding of the data. It also determines the Area used for weighting. The raster cells should have values between 0 and 1. These values should represent the FRACTION of each cell that should be used in the area calculation. This allows you to use a land mask to mask off portions of a cell that are Ocean or water or outside an area of interest. A value of .5 indicates that half of a cells area should not be used. A cell that is 50 percent water thus would have its area adjusted accordingly. This also lets you use irregular areas or polygons inside the square raster

Details

The inventory and data are checked to ensure that stations match. Then an array is created to collect the "counts" of temperature reports for every gridcell for every month. The area of the grid cell is then divided by the count data to produce an inverse density

Value

Return a 3D array of weights for use in other functions

Author(s)

Nick Stokes, rewritten for integration by Steven Mosher. Land masking added by Steven Mosher

Examples

```
## Not run:  
weights <-inverseDensity(Inventory,Data)  
  
## End(Not run)
```

isArray	<i>tests if an object is an Array type</i>
---------	--

Description

the Array object in RghcnV3 has a particular layout. Stations are in dimension 1, months in dimension 2 and years in dimension 3. This function checks that the array has 3 dimensions and that the second dimension is named properly. checks for valid station Ids and years will be added in future versions

Usage

```
isArray(x)
```

Arguments

x an R object

Details

The function checks for the correct number of dimensions and that the second dimension is named properly.

Value

Boolean

Author(s)

Steven Mosher

Examples

```
## Not run:  
check <- isArray(data)  
  
## End(Not run)
```

`isBigMatrix`*isBigMatrix tests the class of an object*

Description

Simply tests whether a matrix is a Big memory file backed big matrix

Usage

```
isBigMatrix(x)
```

Arguments

`x` A matrix to be tested for its class

Details

Functons such as `removeNAStations` test the type of data they are working with

Value

Returns true false

Author(s)

Steven Mosher

`isInventory`*A test function to tell if a dataframe is an inventory*

Description

Since functions depend on an inventory having certain data This function checks inventories to ensure that they have a station Id, a latitude in the second column and Longitide in the thrid column

Usage

```
isInventory(inv)
```

Arguments

`inv` The function is pass a data structure and it is tested to ensure that it has the right inventory information

Details

Because the inventory dataframe can be altered by programmers there is no way to ensure that the require information stays in the object. Until this package is ported to an OOP approach the code will make simple checks to determine if objects comply with certain specifications. An inventory must be a dataframe with Id in column 1, Lat in column 2 and Lon in column 3.

Value

Returns true or false and throws a warning if false

Author(s)

Steven Mosher

Examples

```
## Not run:  
x <- isInventory(inv)  
  
## End(Not run)
```

isMts

A function to test for class type

Description

Checks whether an object is an mts or multiple time series class.

Usage

```
isMts(data)
```

Arguments

data a data object to be tested

Details

This function simply makes code more readable as opposed to `class(data)[1]`

Value

return true or false and throws a warning if false

Author(s)

Steven Mosher

`keepIf`*a sample function for handling NA when calculating means*

Description

This function is used in conjunction with the function [annualize](#). The `annualize` function takes a monthly zoo series of data (temperatures or anomalies) and returns an annual average. The `mean` function, however, only has two ways of calculating a mean: With NA removed or with no NA removed. In order to provide great granular control over this the user can define a function to pass to `annualize`. `keepIf` is an example of such a function.

Usage

```
keepIf(yearData, cnt)
```

Arguments

<code>yearData</code>	<code>yearData</code> is a 12 month section of a zoo monthly series. When <code>annualize</code> is called it passes year sized chunks to <code>mean</code> or to the user defined function. The user defined function must work on a years worth of data. It should return a single number
<code>cnt</code>	In this function the variable <code>cnt</code> represents the minimum number of months required to calculate a mean. So, if <code>cnt</code> is set to 10, then years with 10 or more months will have a annual mean calculated, otherwise NA will be returned

Details

The code for the function takes in a years worth of data. The data is then converted to true/false flags by using the `is.na` function. The NAs are summed and if the year has enough monthly temperatures then a mean is returned to the caller. `keepIf` will work to set a threshold for the number of NAs one allows before rejecting a year. One could also write functions that demanded two months of data in every season.

Value

The function returns a mean for the year or NA if the criteria are not met. For the function `keepIf` one could specify the variable `cnt` to equal 6 and years with 6 or more months of data would have the mean calculated.

Note

Thanks to Peter Oneill for the idea.

Author(s)

Steven Mosher

See Also[annualize](#)

`OCEAN.MASK.URL`*The Url of the land ocean mask*

Description

This url points to the land ocean mask. The dataset consists of percentage of land area in a grid square. Ocean values are represented by 0, while land values can range up to 100 percent. Lakes and rivers and inland water is not taken into account. This is, therefore, a land-ocean mask. The resolution of the map is .25 degrees. Global coverage is -180 to 180 longitude by -90 to 90 latitude. A Land Water mask is also available. This file, as described below, is a "qd" file or quarter degree land fraction file.

Usage`OCEAN.MASK.URL`**Format**

The format is: `chr "http://www.drivehq.com/file/df.aspx/publish/ stevenmosher/PublicFolder/land_percent2_qd.asc"`

Details

"All of the files in the ISLSCP Initiative II data collection are in the ASCII, or text format. The file format consists of numerical fields of varying length, which are delimited by a single space and arranged in columns and rows. The values for the binary land/water or land/sea masks are written as the integers 0 and 1. All values in the land/water/sea fraction files are written as integers from 0 to 100.

The files at different spatial resolutions each contain the following numbers of columns and rows:

One degree (1d): 360 columns by 180 rows
1/2 degree (hd): 720 columns by 360 rows
1/4 degree (qd): 1440 columns by 720 rows

All files are gridded to a common equal-angle lat/long grid, where the coordinates of the upper left corner of the files are located at 180 degrees W, 90 degrees N and the lower right corner coordinates are located at 180 degrees E, 90 degrees S. Data in the files are ordered from North to South and from West to East beginning at 180 degrees West and 90 degrees North." (From the files README)

Source

ISLSCP2 Land/Water and Land/Sea Masks and other Ancillary Data at 0.25, 0.5 and 1 degree Spatial Resolutions

References

Please acknowledge the EOS DEM Science Working Group and the EOS ASTER Instrument Project at JPL for provision of the 30 arcsecond EOS AM1 DEM and the ISLSCP2 data collection when these data are used.

Examples

```
data(OCEAN.MASK.URL)
```

passesCam

A function for picking stations that meet CAM criteria

Description

The Common Anomaly Method (CAM) relies on stations having a minimum number of full years of data in the common anomaly base period. That base period is typically taken as 1961-1990. That period has the maximum amount of stations reporting. With more data in the period the estimate of the mean for that period is improved. The criteria is specified by two numbers: Year and Threshold. If the period is 30 years long (1961-1990) then you want to insure that a station has a good number of full years reporting temperatures in that period. The variable Year defines that criteria. In addition, one has to define what constitutes a full year. Typically 10 to 12 months is used. This means a year with 10 months could qualify as a full year if Threshold is set to 10. The function returns a object with the stations and their data

Usage

```
passesCam(Data, criteria = list(Start = 1961, End = 1990,
                               Years = 15, Threshold = 12))
```

Arguments

Data	Data is an Array, Mts object or Zoo object
criteria	A list of criteria to meet the CAM requirements

Details

The function operates by first checking the criteria that are supplied for obvious errors such as Start being greater than End. Next the function transforms the data into 0 and 1 depending on whether an NA is present or not. The number of valid temperatures or anomalies is computed for each year. A year with 12 months of data will have a count of 12. Then the number of years with months of data greater than or equal to the Threshold is calculated. Stations that pass the criteria are returned.

Value

A data structure matching the call is returned. You feed the function an Array, for example, and you supply a CAM criteria and the data is culled and only those stations that meet the criteria are returned. The format returned for the data matches the structure of the calling data. If you pass in a zoo, you get a zoo out.

Author(s)

Steven Mosher

Examples

```
## Not run:
v3data <- readV3Data(filename="yourfile",output ="Array")
v3data <- windowArray(v3data,start = 1900, end = 2010)
v3data <- passesCam(v3data)

## End(Not run)
```

rasterizeCells	<i>A function to place cell series into a brick</i>
----------------	---

Description

The function `averageByCell` creates a time series that represents the average temperature for a grid cells in a raster. This function places those time series in a raster brick structure

Usage

```
rasterizeCells(ZooCells, r = GLOBE5)
```

Arguments

ZooCells	A 2D zoo object of time series for the cells in a raster.
r	a base raster layer which is used as a format for the output raster brick It's resolution must match the input dataset which was created by <code>averageByCell</code>

Details

This function takes a zoo object of time series for various cells and copies it to a brick. The layer names are set and the Z variable is set. If the resolution of ZooCells (it has an attribute) doesnt match the raster, an error is thrown

Value

Returns a raster brick

Author(s)

Steven Mosher

See Also[averageByCell](#)

Examples

```
## Not run:
TEXAS.DAT <- system.file("external/Texas.dat", package = "RghcnV3")
TEXAS.INV <- system.file("external/Texas.inv", package = "RghcnV3")
texas <- readV3Data(TEXAS.DAT,output = "Mts")
inv <- readInventory(TEXAS.INV)
texas <- window(texas,start =1900, end = 2010 +11/12)

DATA <- intersectInvData(inv,texas)
globe3 <- GLOBE5
res(globe3) <- 3
CellAve <- averageByCell(r = globe3, inventory =DATA$Inventory, Mts = DATA$Mts)
Texas <- rasterizeCells(ZooCells = CellAve, r = globe3)

## End(Not run)
```

rasterizeZoo

A function to rasterize 2D Zoo

Description

The first step in bringing the series of time data into the raster system is the process of rasterization. Every station has a Lat/Lon. In `rasterizeZoo` stations that lie in the same grid cell are averaged together. This function, then outputs a gridded average of time series. It relies on the raster function `rasterize`. At this stage the time series are not area weighted. They are simply averaged with each other per grid cell.

Usage

```
rasterizeZoo(inventory, Zoo, land = GLOBE5)
```

Arguments

inventory	An inventory data.frame. If the stations in the inventory do not match those in the zoo series, an error is thrown.
Zoo	a zoo object of data. If the stations in the zoo object do not match those in the inventory an error is thrown. Inputs are not limited to anomalies. Temperature zoo objects can also be fed to the function. Stations must match in the two data structures.
land	land is defaulted to an empty raster with 5 degree cells. The empty raster will be filled with time series according to the Lat/Lon in the inventory.

Details

In the Common Anomaly Method stations that lie in the same grid cell are averaged. This average is not area weighted at this stage of processing. For every month in the zoo series a new layer of the raster object is created. This results in a brick with a number of layers that is equal to months in the data. The time index of the zoo object is used as the layerNames of the brick. It will also be set in the z slot which can be accessed via getZ in the raster package. For more details please see the documentation on the function rasterize. Effectively, the "points" from the Inventory and the zoo series of data that is associated with those points is fed to the rasterize function.

Value

The function returns a brick. That brick will have layerNames that represent time. Every grid cell will have the average anomaly of all the stations in that grid cell. This data can be plotted, summarized, or further calculations can be performed. Typically, the next step is to area weight the anomalies using a land percentage map and the raster area function.

Author(s)

Steven Mosher

Examples

```
## Not run:
Data <- intersectInvData(inventory,anomalies)
Land <- rasterizeZoo(Data$Inventory,Data$Zoo)
Amonth <-raster(Land,layer=1000)
plot(Amonth)

## End(Not run)
```

readChcn

Reads a file created by package CHCN

Description

The CHCN package output 14 column datasets that are similar to the GHCN version 2 format. Column 1 is a station Id, column 2 is a year, and columns 3:14 are data for the 12 months of the year.

Usage

```
readChcn(filename, output = c("Array", "Mts", "Zoo"))
```

Arguments

filename	Filename to read in
output	Select an output format one of "Array", "Zoo" or "Mts"

Details

The 14 column data is read in using `read.table` then the data is collated into an Array type structure or Zoo or Mts. Stations with no temperature data are redacted as are stations with less than one year of data.

Value

returns an Array type object, Zoo or Mts

Author(s)

Steven Mosher

Examples

```
## Not run:
data <- readChcn(filename = "TaveCHCN.dat", output = "Array")

## End(Not run)
```

readCruStations *A function to read Cru Station data*

Description

reads CRU station data and returns a user selected data format: Zoo, mts or Array. The function supports both CRU3 and CRU4

Usage

```
readCruStations(inFile , output = c("Array", "Zoo", "Mts"))
```

Arguments

<code>inFile</code>	The filename of CRU station data
<code>output</code>	Select One of these three data output options

Details

CRU data and inventory is contained in one monolithic file. The routine parses the data into two objects, an Inventory and a data set. The dataset returned is defined by the setting of the "output" parameter

Value

The function returns a list with two objects and inventory objects and a dataset

Inventory	a dataframe is returned
Zoo	if zoo is selected as output a list element named Zoo is returned
Mts	if Mts is selected as output a list element named Mts is returned
Array	if Array is selected as output a list element named Array is returned

Author(s)

Steven Mosher

Examples

```
## Not run:

Cru <- readCruStations()

## End(Not run)
```

readHadMaps

A function to read in Had/Cru gridded data

Description

Hadley and CRU publish gridded datasets of land temperatures and combined land/ocean data. This data is contained in 5 degree by 5 degree gridded data structures. The function takes a url and reads the ncdf file in and returns a 'raster' brick to the caller. Data starts at 1850. data for every month since 1850 is supplied.

Usage

```
readHadMaps(filename, start = 1850)
```

Arguments

filename	The filename is the local filename of the dataset. Typically, one would call downloadHadMaps and feed the value of that function to the read function.
start	Start should always be set to 1850 unless Hadley or CRU change the date. The dates provided in the files themselves are not self documenting and they appear to be a number of days from Jan 1850.

Details

The hadley maps, either crutem3 or hadcrut3 can both be read in with readHadMaps. This data can be used to validate your work or compare your answers with hadcrut or crutem.

Value

the function returns a raster brick with the time data properly formatted in the layerNames variable. In future releases the time variable will be migrated to the new zSlot in the raster brick structure.

Author(s)

Steven Mosher

References

data can be found at <http://www.metoffice.gov.uk/hadobs/hadcrut3/data/download.html>

See Also

[CRUTEMP3.MAPS.URL](#) [HADCRUT3.MAPS.URL](#) [downloadHadMaps](#)

Examples

```
## Not run:
cruFile <- downloadHadMaps(url = CRUTEMP3.MAPS.URL)
cruBrick <- readHadMaps(cruFile)

## End(Not run)
```

readHadResults

A function to read Monthly Hadley Temperature

Description

Every Month HadCru posts monthly updates of its temperature series. There are three monthly files all in the same format. This function reads and retruns that data directly from the supplied URL.

Usage

```
readHadResults(url = c(HADSST2.RESULTS.URL,
                      HADCRUT3.RESULTS.URL, CRUTEMP3.RESULTS.URL))
```

Arguments

url Three options for the url are shown. Each corresponds to A different dataset. The SST series, The land/ocean series and the land series. In calling this function ONE series should be read. Reading all three series requires three calls to the function with a different URL sent each time. A local copy is not kept. If more than one series is included in the variable url, then a warning is thrown and the first in the list is read from the internet.

Details

Monthly data is updated every month for the three datasets Only monthly data is read in and the annual averages are recalculated. care should be taken in looking at the most recent months. The last line of data in the file will contain data for all the months in that year, even if that month hasn't happened. These are recorded as zeros, rather than NA. The output of this function thus must trim the time series to ensure that these zeros are dropped.

Value

The function returns a monthly zoo series of data from the 1850 to the last month of the current Year.

Author(s)

Steven Mosher

Examples

```
## Not run:

cruMonthly <- readHadResults(CRUTEMP3.RESULTS.URL)
hadMonthly <- readHadResults(HADCRUT3.RESULTS.URL)
sstMonthly <- readHadResults(HADSST2.RESULTS.URL)

## End(Not run)
```

readInventory	<i>A function to read Version 3 Inventory</i>
---------------	---

Description

GHCN temperature datasets contain a station Id which is matched in the inventory file. Between version 2 and version 3 the format of the inventory file has changed. This function reads in an inventory file and outputs a data.frame of stations and their metadata. During this process the function also "cleans" the names in the file. This removes various punctuation marks and country names. Country names are already encoded in the station Id as documented in the readme.

Usage

```
readInventory(filename, Constants = FILE.PARAMETERS)
```

Arguments

filename	A filename on the local machine of the inventory data this file typically ends in .inv
Constants	Defaults to the preloaded constants for reading a file column widths and column names are set there. The column names are changed from the raw data to be more expressive.

Details

columns are renamed
 column one here has the names from the README. column 2 has the names they have been changed into.

GhcnName Newname Type

ID Id Integer
 LATITUDE Lat Real
 LONGITUDE Lon Real
 STNELEV Elevation Real
 NAME Name Character
 GRELEV GridE1 Integer
 POPCLS Rural Character
 POPSIZ Population Integer
 TOPO Topography Character
 STVEG Vegetation Character
 STLOC Coastal Character
 OCNDIS DistanceToCoast Integer
 AIRSTN Airport Boolean
 TOWNDIS DistanceToTown Integer
 GRVEG NDVI Character
 POPCSS Light_Code Character

Value

The function returns a data.frame with columns defined by the constants set by FILE.PARAMETERS. Some variables, such as airports are transformed to boolean. GHCN has a varied method of representing missing data based on the field, this is corrected and NAs are used instead.

Author(s)

Steven Mosher

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README> contains description of all the columns for the inventory.

Examples

```
## Not run: Inv <- readInventory(filename="ghcnm.tavg.v3.0.0.20110612.qca.inv")
```

readMask	<i>A function to read the mask data into a Raster</i>
----------	---

Description

After a land mask is download using `downloadMask()` the next task is to read it into a Raster object. Raster objects are used to represent geographical dataset. This function "wraps" some simple raster commands for ease of use

Usage

```
readMask(filename)
```

Arguments

filename	the filename of a land mask provided with this package. The mask is downloaded using <code>downloadMask()</code>
----------	--

Details

The masks provided through download are global masks the format of the data is asc, ascii data. The function reads in that data using the following raster calls:

```
land <- read.table(filename, sep = " ")  
world <- raster(as.matrix(land), xmn = -180, xmx = 180, ymn = -90, ymx = 90, crs = "+proj=longlat +d  
return(world/100)
```

This means the raster will have a value that runs between 0 and 1 (one). That represents the fraction of land in the cell and is used for land area calculations.

Value

A raster is returned. If the 1/4 degree maps are used each cell will be 1/4 of a degree. That raster can be "aggregated" into a 1 degree map (or any factor) by applying the raster function `aggregate`. Typically a 5 degree cell map will be used.

Author(s)

Steven Mosher

References

see documentation on the raster package

Examples

```
## Not run:  
mask <- getMask(filename="land_water_mask_qd.asc")  
mask5x5 <- aggregate(mask, fact=20)  
  
## End(Not run)
```

readMaskDeg1

Reads a 1 deg land ocean mask from package sources

Description

A local copy of a 1 degree land mask is included with the package. This is derived from the 1/4 degree land ocean mask that the function `downloadMask` gets from ftp. The file has been saved in "raster" format. Downloading and building this yourself is always an option or you can just use the 1 degree version that comes with the package

Usage

```
readMaskDeg1()
```

Details

See documentation on the land masks. This function reads a copy of the land ocean mask that has been included in the package directories. It is constructed from the 1/4 degree land ocean mask using the raster command `aggregate`.

Value

The function returns a raster layer.

Note

At times the ftp server which hosts the 1/4 degree land mask is not available. This file is included in the package should the server be unavailable.

Author(s)

Steven Mosher

Examples

```
LandMask <- readMaskDeg1()
```

`readQC`*A function to read in only the Quality Control Flags*

Description

the GHCN V3 dataset consists of temperature data interspersed with QC data. Every monthly temperature has 3 quality flags. To ease processing and improve the data structures the temperatures are read in independently of the QC flags. This routine reads in the QC data

Usage

```
readQC(filename, Parameters = FILE.PARAMETERS)
```

Arguments

filename	a local filename to the GHCN V3 .dat file. This is the same file that temperatures are read out of.
Parameters	defaulted to the file read parameters.

Details

The dataframes returned are all in 14 column format. The first column is the station Id, second column is the Year. Columns 3:14 are months of the year. This is the same format for temperatures. each Flag type is copied into its own dataframe DM : data measurement flag, nine possible values: QC : quality control flag DS : data source

Value

The function returns a list of dataframes.

QC	a dataframe of QC flags
DM	a dataframe of DM Flags
DS	a dataframe of DS or source flags

Author(s)

Steven Mosher

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

`readSST`*A function to read SST data*

Description

Hadley Sea surface temperature data SST is read from disk into a raster [brick](#). The data extends from 1850 to the present. The data is contain in 5 degree grid cells. each cell has an anomaly from the base period 1961-1990. If you choose different period for your land analysis you should rebaseline the SST data.

Usage

```
readSST(filename)
```

Arguments

`filename` The file path to the "ncdf" file. Upon download this file is named "HadSST2_1850on.nc"

Details

From the website description:

Value

The function returns a raster object. The [layerNames](#) have been replaced with time data where numeric dates are used. There is a layer per month

Author(s)

Steven Mosher

References

[urlhttp://www.metoffice.gov.uk/hadobs/hadsst2/rayner_etal_2005.pdf](http://www.metoffice.gov.uk/hadobs/hadsst2/rayner_etal_2005.pdf) The above paper describes the dataset

See Also

[brick](#)

Examples

```
## Not run: SST <- readSST(filename = "HadSST2_1850on.nc")
```

readV3Data

A Function to read Temperature data in GHCN V3 format

Description

The function performs a fixed width read of the local copy of the GHCN version 3 temperature data file. That file contains more than temperature data, unlike past versions. The read function allows the programmer to read in V3 data and output a V2 format which is much easier for processing. The QA Flags present in V3 format are not passed out to the calling program. They can be accessed by the function readQC(). Missing values are recoded from -9999 to NA.

Usage

```
readV3Data(filename, output = c("Array", "Mts", "Zoo"), Parameters = FILE.PARAMETERS)
```

Arguments

filename	The filename is the local filename of the GHCN V3 data. This file ends with the extension .dat
output	output selects the one type of data you want returned. If you specify more than one type, it will default to the first type. output = "Zoo" will return a zoo object and likewise for the other settings. Default is Array
Parameters	Parameters defaults to the variables set in the list FILE.PARAMETERS This list describes the column widths and column names for the read.fwf call.

Details

In Version 3 of GHCN several Quality control flags have been added. There are three types of flags, DMflags, QCflags, and DSflags. In readV3Data() the quality control flags are not output. They can be read in separately with a call to readQC(). In addition, V3 has a column of data that represents the type of measure being reported, mean, max or min. This is removed as well. The measurand type is evident in the name of the file, so it's redundant

Value

The function will return one of three types of data structures. In versions of RghcnV3 prior to 2.0 a 14 column format was returned That has been replaced with the Array format the format of the source data file is documented at <ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>. Essentially, every month column is followed by 3 columns of QA and source flags. After processing by readV3Data() the QC Flags are removed. UNITS ARE EXPRESSED IN FULL DEGREES C

Author(s)

Steven Mosher; modified for speed by Nick Stokes

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
## Not run:
TEXAS.DAT <- system.file("external/Texas.dat", package = "RghcnV3")
texas <- readV3Data(TEXAS.DAT, output = "Zoo")
## End(Not run)
```

referenceStation	<i>Determines a reference station for local area</i>
------------------	--

Description

This function is based on the Berkeley Method of calculating an average temperature from a collection of stations. This particular variant was developed by the Blogger/Time series specialist Tamino. An improvement on the reference station method for combining stations in a close proximity, this method uses all the information provided by the stations to calculate an average for the region. See the post referenced in the notes below for a detailed mathematical theory

Usage

```
referenceStation(Data, offres = 0.001)
```

Arguments

Data	Data must be a zoo series as created by the function <code>v3ToZoo</code> . Support for 'mts' class objects has also been added. These can be created with <code>v3ToMts</code>
offres	A threshold for the offset variable that determines when the iterative solution is completed.

Details

The function should be called with a zoo series of temperature data or mts of temperature data. The function finds a set of "offsets" that are used to estimate a reference temperature for the region.

Value

The function has been modified somewhat from tamino's public release. The number of input parameters has been cut. Since the time data in a zoo series is already matched, there is no rounding required for time. The rounding variable for data has been removed, users can round the output if required. The flag that allowed for full output of data, including the temperature input dataset and the station Ids has been removed. The full dataset is the calling parameter and is not changed by the program. The output consists of a list: The names of that list will depend upon the class of the object being worked on. If the function is fed a 'zoo' object it returns an item named "Zoo". Mts, returns "Mts"

Zoo	A zoo object of data, stations, se and stdev
Mts	a "mts" object of data, station count, se,stdev
offsets	a vector of the offsets

Note

http://www.berkeleyearth.org/Resources/Berkeley_Earth_Summary.pdf, <http://tamino.files.wordpress.com/2011/07/align.pdf>

Author(s)

Tamino; adapted by Steven Mosher

References

[urlhttp://tamino.wordpress.com/2011/07/06/aligning-station-records/](http://tamino.wordpress.com/2011/07/06/aligning-station-records/)

Examples

```
## Not run:
# get the data
# get the inventory
# crop the inventory to a region
# window the data to a time
# makeZoo series
# intersect the two
# call regional
# plotthe answers
TEXAS.DAT <- system.file("external/Texas.dat", package = "RghcnV3")
TEXAS.INV <- system.file("external/Texas.inv", package = "RghcnV3")
v3Mean <- readV3Data(TEXAS.DAT,output ="Mts")
v3Inv <- readInventory(TEXAS.INV)
e <- extent(-100,-90,30,40)
inv <- cropInv(v3Inv,e)
v3late <- window(v3Mean,start = 1950, end = 2010 +11/12)

Data <- intersectInvData(inv,v3late)
REGION <- referenceStation(Data=Data$Zoo)
plot(REGION$Zoo$data)

## End(Not run)
```

removeNaStations	<i>A function to remove stations that have no temperatures</i>
------------------	--

Description

During the course of processing zoo objects, Mts objects and Arrays it is possible to create stations that have only NAs in their data fields. This can cause problems for code that take a mean of the data. a mean of NAs is not NA, but rather is Nan. This function then is used internally by other functions to ensure that such stations are removed so that they do not cause errors downstream. The window function from zoo does not make such a check so care must be taken. For safety always call this function after windowing zoo data. Especially if you intend to perform other functions on the data. NOTE: na.trim() does not remove a single zoo series from a zoo object.

Usage

```
removeNaStations(Data)
```

Arguments

Data	A zoo series object or mts object or Array
------	--

Details

The collection of zoo series or ts() series are examined and if a station (column) is all NA, then that column is removed. Similar processing occurs for Array type objects.

Value

returns a zoo series, mts series or Array with the empty stations removed

Author(s)

Steven Mosher

solveTempAnnual	<i>Solve temperature, annual version</i>
-----------------	--

Description

This version of the temperature solver is the same as solveTemperature except that it returns residuals and local offsets

Usage

```
solveTempAnnual(Data, Weights)
```

Arguments

Data	A 3D array of temperatures
Weights	Weights created by inverseDensity

Details

This function is identical to solveTemperature except it returns residuals and offsets

Value

Solution	Returns the solution
Resid	Returns residuals
LocalOffset	Local mean offsets

Author(s)

Nick Stokes

Examples

```
## Not run:
###
## End(Not run)
```

solveTemperature *A function to solve a least squares*

Description

In Nick Stoke's Method a collection of temperature series is solved used a weighted least squares approach. There is no common anomaly period and no combining of stations into long records. Every bit of station data is used to estimate the best fit to all the data

Usage

```
solveTemperature(Data, Weights)
```

Arguments

Data	A 3D Array of temperatures
Weights	Weights as caluclated by inverseDensity for example

Details

Get nick to write something

Value

returns a normalize series of temperature for the input series

Author(s)

Nick Stokes, rewritten by Steven Mosher

References

link for nicks blog posts

Examples

```
## Not run:
# write and test a demo

## End(Not run)
```

solveTempMonthly

Solve temperature, Monthly version

Description

This version of the temperature solver is the same as solveTemperature except that it returns residuals and local offsets

Usage

```
solveTempMonthly(Data, Weights)
```

Arguments

Data	A 3D array of temperatures
Weights	Weights created by inverseDensity

Details

This function is identical to solveTemperature except it returns residuals and offsets

Value

Solution	Returns the solution
Resid	Returns residuals
LocalOffset	Local mean offsets

Author(s)

Nick Stokes

See Also

solveTemperature

Examples

```
## Not run:  
#  
  
## End(Not run)
```

`stationCount`*A function to count the stations versus time*

Description

The function counts the number of stations reporting temperatures for every month in the series

Usage`stationCount(Data)`**Arguments**

`Data` An Array, Zoo series or Mts object

Details

simply counts the number of valid temperatures per month which is the number of stations reporting for that month

Value

returns a zoo series of stations

Author(s)

Steven Moshier

Examples

```
## Not run:  
plot(stationCount(Data))  
  
## End(Not run)
```

tempStats	<i>calculates statistics for all stations in an object</i>
-----------	--

Description

returns a dataframe of stats for every station. mean, max, min, sd, first year, full years, total months

Usage

```
tempStats(data)
```

Arguments

data a Array, Zoo object or Mts object

Details

Statistics are calculated for every station in an object

Value

returns a data frame

Author(s)

Steven Mosher

Examples

```
## Not run:  
stat <- tempStats(data)  
  
## End(Not run)
```

trimHeadTail	<i>Trims the head and tail of a time series object</i>
--------------	--

Description

The functions in this package assume that time series start with january and end with december. This function will trim any object (Array,Mts, Zoo) to remove all months before the first january and after the last december

Usage

```
trimHeadTail(Data)
```

Arguments

Data Any Array object, Mts Object or Zoo object

Details

Different sources such as CRU or Berkeley supply data in formats that may include incomplete years at the head and tail. This function quickly trims the head and tail.

Value

Returns the same object only trimmed such that the first time is a january and the last time is a december. Every year will have 12 months.

Author(s)

Steven Mosher

```
trimNA
```

```
TrimNA
```

Description

This function trims the leading NA values from an Array object, Mts object or Zoo object

Usage

```
trimNA(Data)
```

Arguments

Data A 3D Array of stations temps or a Mts object. Zoo is acceptable but is coerced to Mts on return

Details

The functions solveTemp and solveMonthly temp rely on having inputs where leading NA have been removed. For example, if a collection of stations starts in 1880 with the first six months missing or NA, the solveMonthly algorithm will fail. Leading NA must be removed. In general the start year should be a year where at least one station has a january present

Value

returns a trimmed dataobject. You can use windowArray or window to trim to an exact year

Author(s)

Steven Mosher

See Also

solveTempMonthly, solveTempAnnual

V3.MEAN.ADJ.URL

Url for Adjusted GHCN V3 data

Description

The url to the Adjusted GHCN V3 data. This dataset is an Adjusted dataset of Tave or Average temperature.

Usage

V3.MEAN.ADJ.URL

Format

The format is: chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tavg.latest.qca.tar.gz"

Source

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/>

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
data(V3.MEAN.ADJ.URL)
print(V3.MEAN.ADJ.URL)
```

V3.MEAN.RAW.URL *Url for UnAdjusted GHCN V3 data*

Description

The url to the UnAdjusted GHCN V3 data. This dataset is an Adjusted dataset of Tave or Average temperature.

Usage

V3.MEAN.RAW.URL

Format

The format is: chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tavg.latest.qcu.tar.gz"

Source

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/>

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
data(V3.MEAN.RAW.URL)
print(V3.MEAN.RAW.URL)
```

V3.TMAX.ADJ.URL *Url for Adjusted GHCN V3 TMAX data*

Description

A url to the GHCN V3 TMAX dataset. This data is adjusted. see the raw version of the URL for unadjusted data

Usage

V3.TMAX.ADJ.URL

Format

The format is: chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tmax.latest.qca.tar.gz"

Source

```
ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/
```

References

```
ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README
```

Examples

```
data(V3.TMAX.ADJ.URL)  
print(V3.TMAX.ADJ.URL)
```

V3.TMAX.RAW.URL

A url for the GHCN V3 Tmax data, unadjusted

Description

This url points to the unadjusted Tmax (maximum) dataset for GHCN V3

Usage

```
V3.TMAX.RAW.URL
```

Format

The format is: `chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tmax.latest.qcu.tar.gz"`

Source

```
ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/
```

References

```
ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README
```

Examples

```
data(V3.TMAX.RAW.URL)  
print(V3.TMAX.RAW.URL)
```

V3.TMIN.ADJ.URL *A url to the GHCN V3 adjusted Tmin data*

Description

A url to the adjusted Tmin (minimum) Temperature dataset for GHCN version 3.

Usage

V3.TMIN.ADJ.URL

Format

The format is: chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tmin.latest.qca.tar.gz"

Source

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/>

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
data(V3.TMIN.ADJ.URL)
print(V3.TMIN.ADJ.URL)
```

V3.TMIN.RAW.URL *A url to the GHCN V3 undjusted Tmin data*

Description

A url to the unadjusted Tmin (minimum) Temperature dataset for GHCN version 3.

Usage

V3.TMIN.RAW.URL

Format

The format is: chr "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/ghcnm.tmin.latest.qcu.tar.gz"

Source

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/>

References

<ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/v3/README>

Examples

```
data(V3.TMIN.RAW.URL)
print(V3.TMIN.RAW.URL)
```

validData

validData performs sanity checks on temperature data

Description

During the course of creating temperature data objects, such as Arrays various sources can be merged. This can lead to situations where you have duplicate stations or stations with all the data missing or stations that do not start on whole years. validData checks for a variety of issues and returns diagnostics. Internally other functions which depend upon valid data will call validData

Usage

```
validData(Data)
```

Arguments

Data An Array or Mts or Zoo object

Details

The function checks for the following conditions: duplicate station Ids in the temperature data. Objects with a number of months that is not a multiple of 12. Months where no station has a report. The issues are reported but not fixed.

Value

Returns TRUE or returns FALSE with a diagnostic

Author(s)

Steven Mosher

validMonthCount	<i>Checks an object for having all months complete</i>
-----------------	--

Description

By definition a 3D Array has slots for every month of every year. Even if the data is NA. Some sources, however, may drop months if they have missing data. This function checks whether an Mts or Zoo object has all months present and checks that it begins with January and ends with december

Usage

```
validMonthCount(Data)
```

Arguments

Data	An Array, Mts or Zoo object
------	-----------------------------

Details

If you pass an Array it returns true by definition. Otherwise the number of months is checked for divisibility by 12 and the cycle is checked to insure that the structure starts with january and ends with december

Value

returns TRUE or FALSE

Author(s)

Steven Mosher

WATER.MASK.URL	<i>A url to a land water mask</i>
----------------	-----------------------------------

Description

This url points to the land water mask. The dataset consists of percentage of land area in a grid square. Water values are represented by 0, while land values can range up to 100 percent. Lakes and rivers and inland water are taken into account. This is, therefore, a land-water mask. The resolution of the map is .25 degrees. Global coverage is -180 to 180 longitude by -90 to 90 latitude. A Land Ocean mask is also available. This files, as described below, is a "qd" file or quarter degree, land fraction file.

Usage

```
WATER.MASK.URL
```

Format

The format is: chr "http://www.drivehq.com/file/df.aspx/publish/ stevenmosher/PublicFolder/land_percent_qd.asc"

Details

"All of the files in the ISLSCP Initiative II data collection are in the ASCII, or text format. The file format consists of numerical fields of varying length, which are delimited by a single space and arranged in columns and rows. The values for the binary land/water or land/sea masks are written as the integers 0 and 1. All values in the land/water/sea fraction files are written as integers from 0 to 100.

The files at different spatial resolutions each contain the following numbers of columns and rows:

One degree (1d): 360 columns by 180 rows 1/2 degree (hd): 720 columns by 360 rows 1/4 degree (qd): 1440 columns by 720 rows

All files are gridded to a common equal-angle lat/long grid, where the coordinates of the upper left corner of the files are located at 180 degrees W, 90 degrees N and the lower right corner coordinates are located at 180 degrees E, 90 degrees S. Data in the files are ordered from North to South and from West to East beginning at 180 degrees West and 90 degrees North." (From the files README)

Source

ISLSCP2 Land/Water and Land/Sea Masks and other Ancillary Data at 0.25, 0.5 and 1 degree Spatial Resolutions

References

Please acknowledge the EOS DEM Science Working Group and the EOS ASTER Instrument Project at JPL for provision of the 30 arcsecond EOS AM1 DEM and the ISLSCP2 data collection when these data are used.

Examples

```
data(WATER.MASK.URL)
print(WATER.MASK.URL)
```

windowArray

A function to subset Array Data by year

Description

After reading in V3 data using readV3Data the dataset can be reduced in size by selecting a subset or windowArray of data. This operates exactly like the window function in zoo or time series window functions.

Usage

```
windowArray(Data, start, end)
```

Arguments

Data	A 3D data array
start	The start year you want to include. Only integer years are allowed
end	The end year you want to include. Only integer years are allowed

Details

The function only takes an Array. It must have a dimension with year data as dimnames

Value

Returns an Array with Years outside the window removed

Author(s)

Steven Mosher

Examples

```
## Not run:
meanAdj   <- downloadV3(url=V3.MEAN.ADJ.URL)
meanAdata <- readV3Data(filename=meanAdj$DataFilename,output ="Array")
Inventory <- readInventory(filename=meanAdj$InventoryFile)
mean1900  <- windowArray(meanAdata,start=1900,end=2010)

## End(Not run)
```

Index

- *Topic **Contributed**
 - referenceStation, 58
- *Topic **Downloads**
 - downloadCRU, 20
- *Topic **Download**
 - downloadHadMaps, 21
 - downloadMask, 22
 - downloadSST, 24
 - downloadV3, 25
- *Topic **Internals**
 - checkCriteria, 15
 - getDemoFiles, 28
 - isArray, 39
 - isInventory, 40
 - isMts, 41
 - removeNaStations, 60
- *Topic **Regression**
 - averageByCell, 11
 - averageStations, 12
 - rasterizeCells, 45
 - referenceStation, 58
- *Topic **TimeSeries**
 - annualize, 5
 - anomalize, 7
 - asArray, 8
 - asMts, 9
 - asZoo, 10
 - averageByCell, 11
 - averageStations, 12
 - camMask, 14
 - createAnomaly, 15
 - keepIf, 42
 - passesCam, 44
 - rasterizeCells, 45
 - referenceStation, 58
 - removeNaStations, 60
 - solveTempAnnual, 60
 - solveTemperature, 61
 - solveTempMonthly, 62
 - stationCount, 63
 - tempStats, 64
 - windowArray, 72
- *Topic **Timeseries**
 - allNA, 4
 - findStartEnd, 27
 - isBigMatrix, 40
 - trimHeadTail, 64
 - trimNA, 65
 - validData, 70
 - validMonthCount, 71
- *Topic **Zoo**
 - annualize, 5
 - anomalize, 7
- *Topic **datasets**
 - CRU.STATIONS.URL, 18
 - CRUTEMP3.MAPS.URL, 19
 - CRUTEMP3.RESULTS.URL, 20
 - FILE.PARAMETERS, 26
 - GHCN.V3.DATA, 31
 - GLOBE5, 31
 - HADCRUT3.MAPS.URL, 33
 - HADCRUT3.RESULTS.URL, 34
 - HADSST2.RESULTS.URL, 35
 - HADSST2.URL, 35
 - OCEAN.MASK.URL, 43
 - V3.MEAN.ADJ.URL, 66
 - V3.MEAN.RAW.URL, 67
 - V3.TMAX.ADJ.URL, 67
 - V3.TMAX.RAW.URL, 68
 - V3.TMIN.ADJ.URL, 69
 - V3.TMIN.RAW.URL, 69
 - WATER.MASK.URL, 71
- *Topic **files**
 - downloadCRU, 20
 - readChcn, 47
 - readCruStations, 48
- *Topic **file**
 - readHadMaps, 49

- readHadResults, [50](#)
- readInventory, [51](#)
- readMask, [53](#)
- readQC, [55](#)
- readSST, [56](#)
- readV3Data, [57](#)
- *Topic **package**
 - RghcnV3-package, [3](#)
- *Topic **spatial**
 - averageByCell, [11](#)
 - averageStations, [12](#)
 - cropInv, [17](#)
 - getLonLat, [29](#)
 - getStations, [30](#)
 - intersectInvData, [36](#)
 - inverseDensity, [37](#)
 - rasterizeCells, [45](#)
 - rasterizeZoo, [46](#)
 - readMaskDeg1, [54](#)
 - RghcnV3-package, [3](#)
 - solveTemperature, [61](#)
- allNA, [4](#)
- annualize, [5](#), [42](#), [43](#)
- anomalize, [7](#)
- asArray, [8](#), [10](#)
- asMts, [9](#), [9](#)
- asZoo, [9](#), [10](#), [10](#)
- averageByCell, [11](#), [45](#)
- averageStations, [11](#), [12](#), [12](#)
- brick, [56](#)
- camMask, [14](#)
- checkCriteria, [15](#)
- createAnomaly, [8](#), [15](#)
- crop, [17](#)
- cropInv, [17](#)
- CRU.STATIONS.URL, [18](#)
- CRUTEMP3.MAPS.URL, [19](#), [50](#)
- CRUTEMP3.RESULTS.URL, [20](#)
- downloadCRU, [20](#)
- downloadHadMaps, [21](#), [50](#)
- downloadMask, [22](#)
- downloadSST, [24](#)
- downloadV3, [25](#)
- extent, [17](#)
- FILE.PARAMETERS, [26](#)
- findStartEnd, [27](#)
- getDemoFiles, [28](#)
- getLonLat, [29](#)
- getStations, [30](#)
- GHCN.V3.DATA, [31](#)
- GLOBE5, [31](#)
- HADCRUT3.MAPS.URL, [33](#), [50](#)
- HADCRUT3.RESULTS.URL, [34](#)
- HADSST2.RESULTS.URL, [35](#)
- HADSST2.URL, [35](#)
- intersectInvData, [36](#)
- inverseDensity, [37](#)
- isArray, [39](#)
- isBigMatrix, [40](#)
- isInventory, [40](#)
- isMts, [41](#)
- keepIf, [5](#), [6](#), [42](#)
- layerNames, [56](#)
- OCEAN.MASK.URL, [43](#)
- passesCam, [8](#), [44](#)
- rasterize, [46](#)
- rasterizeCells, [12](#), [45](#)
- rasterizeZoo, [46](#)
- readChcn, [47](#)
- readCruStations, [48](#)
- readHadMaps, [22](#), [49](#)
- readHadResults, [50](#)
- readInventory, [51](#)
- readMask, [53](#)
- readMaskDeg1, [54](#)
- readQC, [55](#)
- readSST, [56](#)
- readV3Data, [8](#), [57](#)
- referenceStation, [13](#), [58](#)
- removeNaStations, [60](#)
- RghcnV3 (RghcnV3-package), [3](#)
- RghcnV3-package, [3](#)
- solveTempAnnual, [60](#)
- solveTemperature, [61](#)
- solveTempMonthly, [62](#)

stationCount, [63](#)

tempStats, [64](#)

trimHeadTail, [64](#)

trimNA, [65](#)

V3.MEAN.ADJ.URL, [66](#)

V3.MEAN.RAW.URL, [67](#)

V3.TMAX.ADJ.URL, [67](#)

V3.TMAX.RAW.URL, [68](#)

V3.TMIN.ADJ.URL, [69](#)

V3.TMIN.RAW.URL, [69](#)

validData, [70](#)

validMonthCount, [71](#)

WATER.MASK.URL, [71](#)

windowArray, [72](#)