Package 'rNOMADS'

June 3, 2015

Type Package

Title An Interface to the NOAA Operational Model Archive and Distribution System

Version 2.1.4

Date 2015-06-03

Depends R (>= 3.1.1), rvest (>= 0.1.0)

Imports scrapeR (>= 0.1.6), stringr (>= 0.6.2), fields (>= 6.7.6), GEOmap (>= 2.1), MBA, RCurl (>= 1.95-4.1), XML (>= 3.98.1.1)

Description An interface to the National Oceanic and Atmospheric Administration's Operational Model Archive and Distribution System (NOMADS) that allows R users to quickly and efficiently download global and regional weather model data for processing. rNOMADS currently supports a variety of models ranging from global weather data to an altitude of 40 km, to high resolution regional weather models, to wave and sea ice models. It can also retrieve archived NOMADS models. rNOMADS can retrieve binary data in grib format as well as import ascii data directly into R by interfacing with the GrADS-DODS system.

License GPL (>= 3)

URL https://r-forge.r-project.org/projects/rnomads/ (subversion repository)

http://www.unc.edu/~haksaeng/rNOMADS/rNOMADS_dods_examples.pdf (GrADS-DODS examples)

http://www.unc.edu/~haksaeng/rNOMADS/rNOMADS_grib_examples.pdf (grib examples)

Maintainer Daniel C. Bowman <daniel.bowman@unc.edu>

MailingList rnomads-user@lists.r-forge.r-project.org

NeedsCompilation no

Author Daniel C. Bowman [aut, cre]

Repository CRAN

Date/Publication 2015-06-03 19:00:30

R topics documented:

`rNOMADS-package` *An interface to the NOAA Operational Model Archive and Distribution System*

Description

Automatically download forecast data from the National Oceanic and Atmospheric Administration's Operational Model Archive and Distribution System (NOMADS) and read it into R. This can be done in two ways: reading ascii data directly from the server using the DODS-GrADS system (all operating systems, just needs an Internet connection) or downloading binary files in GRIB format (Linux and Mac OS only, or until a native R reader for GRIB is available). The grib capability of `rNOMADS` uses an external series of routines called `wgrib2` to read operational model data; get `wgrib2` at `http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/`. The package will also attempt to call another external routine called `wgrib` if the user wishes to read GRIB1 files; get `wgrib` at `http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html`.

Details

Package:	rNOMADS
Type:	Package
Version:	2.0.4
Date:	2014-10-30

License: GPL v3

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

NOMADS website:
http://nomads.ncep.noaa.gov/
wgrib2 download page:
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/
wgrib2 reference:
Ebisuzaki, W, Bokhorst, R., Hyvatti, J., Jovic, D., Nilssen, K, Pfeiffer, K., Romero, P., Schwarb, M., da Silva, A., Sondell, N., and Varlamov, S. (2011). wgrib2: read and write GRIB2 files. *National Weather Service Climate Prediction Center*,
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/
wgrib download page:
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html

Examples

```
#Getting temperature for North Carolina, USA,
#6-12 hours ago depending on when the latest model run was.
#Get values at the ground surface and at the 800 mb level
#Then make a contour plot of the surface temperature.
#We use GrADS-DODS here for compatibility.

#Using the Global Forecast System 0.5x0.5 model
## Not run:

urls.out <- GetDODSDates(abbrev = "gfs_0p50")
model.url <- tail(urls.out$url, 1) #Get most recent model date

#Get most recent model run

model.runs <- GetDODSModelRuns(model.url)
model.run <- tail(model.runs$model.run, 1)


#Get ground temperature for the 6 hour prediction
variable <- "tmp2m" #temp at 2 m
time <- c(2,2) #6 hour prediction
lon.dom <- seq(0, 360, by = 0.5) #domain of longitudes in model
lat.dom <- seq(-90, 90, by = 0.5) #domain of latitudes in model
lon <- which((lon.dom >= 360 - 84) & (lon.dom <= 360 - 74)) - 1 #NOMADS indexes start at 0
lat <- which((lat.dom <= 37) & (lat.dom >= 32)) - 1 #NOMADS indexes start at 0
```

```
model.data.surface <- DODSGrab(model.url, model.run, variable, time, c(min(lon), max(lon)),
   c(min(lat), max(lat)))

lev <- c(8, 8) #800 mb level
variable <- "tmpprs"
model.data.800mb <- DODSGrab(model.url, model.run, variable, time, c(min(lon), max(lon)),
   c(min(lat), max(lat)), level = lev)

#Make results into arrays
model.array.surface <- ModelGrid(model.data.surface, c(0.5, 0.5), "latlon")
model.array.800mb <- ModelGrid(model.data.800mb, c(0.5, 0.5), "latlon")

#Make a contour plot of the temperature around North Carolina, USA:
contour(x = model.array.surface$x - 360, y = model.array.surface$y,
    model.array.surface$z[1,1,,] - 273.15, xlab = "Longitude", ylab = "Latitude",
   main = paste("North Carolina Surface Temperatures for",
   model.array.surface$fcst.date, "GMT in Celsius"))

dev.new()
contour(x = model.array.800mb$x - 360, y = model.array.800mb$y,
    model.array.800mb$z[1,1,,] - 273.15, xlab = "Longitude", ylab = "Latitude",
   main = paste("North Carolina Temperatures at 800 mb for",
   model.array.surface$fcst.date, "GMT in Celsius"))

## End(Not run)
```

`ArchiveGribGrab` *Download archived model data from the NOMADS server.*

Description

This function interfaces with the programming API at http://nomads.ncdc.noaa.gov/ to download archived NOMADS model data. The available models can be viewed by calling `NOMADSArchiveList` without arguments. The data arrives in grib (gridded binary) format that can be read with `ReadGrib`. Some of these files are in GRIB format, others are in GRIB2 format; select the appropriate file type when calling `ReadGrib`.

Usage

```
ArchiveGribGrab(abbrev, model.date, model.run, pred,
    local.dir = ".", file.name = "fcst.grb", tidy = FALSE,
    verbose = TRUE, download.method = NULL, file.type = "grib2")
```

Arguments

`abbrev`	Model abbreviation per `NOMADSArchiveList`.
`model.date`	The year, month, and day of the model run, in YYYYMMDD format
`model.run`	Which hour the model was run (i.e. 00, 06, 12, 18 for GFS)

pred	Which prediction to get (analysis is 00)
local.dir	Where to save the grib file, defaults to the current directory.
file.name	What to name the grib file, defaults to "fcst.grb".
tidy	If TRUE, remove all files with the suffix ".grb" from local.dir prior to downloading a new grib file.
verbose	If TRUE, give information on connection status. Default TRUE
download.method	Allows the user to set the download method used by download.file: "internal", "wget" "curl", "lynx". If NULL (the default), let R decide.
file.type	Determine whether to get GRIB1 ("grib1") or GRIB2 ("grib2") file formats. Sometimes both are available, sometimes only one.

Value

grib.info$local.dir	The absolute path to the grib file that was downloaded.
grib.info$file.name	The name of the grib file that was downloaded.
grib.info$url	The URL that the grib file was downloaded from

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

http://nomads.ncdc.noaa.gov/

See Also

CheckNOMADSArchive, NOMADSArchiveList, ReadGrib

Examples

```
#An example for the Global Forecast System
#Get data for January 1 2014
#Temperature at 2 m above ground
#3 hour prediction
# using GRIB

abbrev <- "gfsanl"
model.date <- 20140101
model.run <- 06
pred <- 3

## Not run:
model.info <- ArchiveGribGrab(abbrev, model.date,
    model.run, pred, file.type = "grib2")
model.data <- ReadGrib(model.info$file.name, c("2 m above ground"), c("TMP"))
```

```
#Transform to grid
gridded.data <- ModelGrid(model.data, c(0.5, 0.5))

#Get surface temperature in Chapel Hill, NC
lat <- 35.907605
lon <- -79.052147

profile.data <- BuildProfile(gridded.data, lon, lat, TRUE)
print(paste0("The temperature prediction in Chapel Hill was ",
    sprintf("%.0f", profile.data[1,1] - 272.15), " degrees Celsius."))

## End(Not run)
```

`BuildProfile` *Get model data at a specific point.*

Description

Takes the output of `ModelGrid` and extracts data at a specific point, performing interpolation if required.

Usage

```
BuildProfile(gridded.data, lon, lat, spatial.average)
```

Arguments

`gridded.data`	Data structure returned by `ModelGrid`.
`lon`	Longitude of point of interest.
`lat`	Latitude of point of interest.
`spatial.average`	Whether to interpolate data using b-splines to obtain value at the requested point (`spatial.average = TRUE`) or use the nearest model node (`spatial.average = FALSE`).

Details

It is much more efficient to download a large chunk of data and extract profile points from that as opposed to downloading individual small model chunks in the vicinity of each point of interest.

Value

`profile.data`	A levels x variables matrix with data for a given point.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

ModelGrid, BuildProfile

Examples

```
#Get temperature profile in Chapel Hill, NC.

#First, define each location
lon <- -79.052094
lat <- 35.907562

#Get latest GFS 0.5 model, use analysis forecast
## Not run:
model.url <- CrawlModels(abbrev = "gfs_0p50", depth = 1)[1]
pred <- ParseModelPage(model.url)$pred[1]

## End(Not run)

#Get levels
pressure <- c(1, 2, 3, 5, 7,
10, 20, 30, 50, 70,
seq(100, 1000, by = 25))
levels <- paste(pressure, " mb", sep = "")

#Variables - temperature and height only
variables <- c("TMP", "HGT")

## Not run:
grib.info <- GribGrab(model.url, pred, levels, variables,
   model.domain = c(-85, -75, 37, 32))
grib.data <- ReadGrib(grib.info$file.name, levels, variables)
gridded.data <- ModelGrid(grib.data, c(0.5, 0.5))

profile <- BuildProfile(gridded.data, lon, lat, TRUE)
plot(profile[,2] - 273.15, profile[,1], xlab = "Temperature (C)",
   ylab = "Height (m)", main = "Temperature Profile above Chapel Hill, NC")

## End(Not run)
```

CheckNOMADSArchive *Check to see if archived data exists.*

Description

This function checks to see if data exists for a given date and model. It checks for both GRIB1 or GRIB2 files.

Usage

```
CheckNOMADSArchive(abbrev, model.date = NULL)
```

Arguments

abbrev	Model abbreviation per NOMADSArchiveList.
model.date	The year, month, and day to check for data, in YYYYMMDD format. If NULL, check all available dates in NOMADS archive.

Value

`available.models$date`
: What date the file is for, in YYYYMMDD format.

`available.models$model.run`
: At what hour (GMT) the model was run.

`available.models$pred`
: What predictions are available

`available.models$file.name`
: List of file names for available model dates, runs, and predictions

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

http://nomads.ncdc.noaa.gov/

See Also

NOMADSArchiveList, ArchiveGribGrab

Examples

```
#See what is available for January 1 2014

abbrev <- "gfs4"
model.date <- 20140101
## Not run: gfs.available.models <- CheckNOMADSArchive(abbrev, model.date)

#Determine all available North American Mesoscale models in the archive
#This will take some time
## Not run: nam.available.models <- CheckNOMADSArchive("namanl")
```

CrawlModels *Get Available Model Runs*

Description

This function determine which instances of a given model are available for download.

Usage

```
CrawlModels(abbrev = NULL, url = NULL, depth = NULL, verbose = TRUE)
```

Arguments

abbrev	The model abbreviation, see NOMADSRealTimeList. Defaults to NULL.
url	A URL to use instead of using the abbreviations in NOMADSRealTimeList. Defaults to NULL.
depth	How many model instances to return. This avoids having to download the entire model list (sometimes several hundred) if only the first few instances are required. Defaults to NULL, which returns everything.
verbose	Print out each link as it is discovered. Defaults to TRUE.

Details

This function calls WebCrawler, a recursive algorithm that discovers each link available in the URL provided. It then searches each link in turn, and follows those links until it reaches a dead end. At that point, it returns the URL. For the model pages on the NOMADS web site, each dead end is a model instance that can be examined using ParseModelPage or have data retrieved from it using GribGrab.

Value

urls.out	A list of web page addresses, each of which corresponds to a model instance.

Note

It is a good idea to set depth to a small number rather than leave it at the default value. Some models (such as the Global Forecast System) have a large number of instances, and crawling each one can take a lot of time. I recommend depth = 2, since the first URL may not have an active model on it yet if the model is still being uploaded to the server. In that case,the first URL will contain no data, and the second URL can be used instead.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

WebCrawler, ParseModelPage, NOMADSRealTimeList, GribGrab

Examples

```
#Get the latest 5 instances
#for the Global Forecast System 0.5 degree model

## Not run: urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 5)
```

DODSGrab *Download model data from the NOMADS server using the GrADS-DODS system.*

Description

This function interfaces with the NOMADS server to download weather, ocean, and sea ice data. The available models can be viewed by calling NOMADSRealTimeList and NOMADSArchiveList. The data arrives in ascii format, so this function can be used to retrieve data on any operating system.

Usage

```
DODSGrab(model.url, model.run, variables, time, lon, lat,
    levels = NULL, display.url = TRUE, verbose = FALSE, request.sleep = 1)
```

Arguments

model.url	A model URL for a specific date, probably from GetDODSDates.
model.run	A specific model run to get, probably from GetDODSModelRuns.
variables	A list of the data types to get.
time	An two component vector denoting which time indices to get.
lon	An two component vector denoting which longitude indices to get.
lat	An two component vector denoting which latitude indices to get.
levels	An two component vector denoting which levels to get, if applicable.
display.url	If TRUE, print out the URL for the data request.
verbose	If TRUE, give a very detailed description of the download. Default FALSE.
request.sleep	If multiple requests are to be sent to the server, pause by this many seconds between them. This is courteous and also helps prevent timeouts.

Value

model.data	A structure with a series of elements containing data extracted from GrADS-DODS system.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

http://nomads.ncep.noaa.gov/

See Also

GetDODSDates, GetDODSModelRuns, GetDODSModelRunInfo

Examples

```
#An example for the Global Forecast System 0.5 degree model
#Make a world temperature map for the latest model run

## Not run:
#Figure out which model is most recent
model.urls <- GetDODSDates("gfs_0p50")
latest.model <- tail(model.urls$url, 1)
model.runs <- GetDODSModelRuns(latest.model)
latest.model.run <- tail(model.runs$model.run, 1)

#Download worldwide temperature data at 2 m
variable <- "tmp2m"
time <- c(0, 0) #Analysis run, index starts at 0
lon <- c(0, 719) #All 720 longitude points
lat <- c(0, 360) #All 361 latitude points
model.data <- DODSGrab(latest.model, latest.model.run,
   variable, time, lon, lat)

#Make it into a nice array and plot it
model.grid <- ModelGrid(model.data, c(0.5, 0.5), "latlon")
image(model.grid$z[1,1,,])

## End(Not run)
```

GetClosestForecasts *Get the forecast time closest to a given date for a given model*

Description

This function returns which forecast precedes the date and which forecast follows the date for a given model product. Thus a user can average the two forecasts together to provide a precise forecast for a given date.

Usage

```
GetClosestForecasts(abbrev, forecast.date, model.date = "latest",
   depth = NULL, verbose = TRUE)
```

Arguments

abbrev	The requested model product
forecast.date	What date you want a forecast for, as a date/time object. It must be in the GMT time zone.
model.date	Which model run to use, in YYYYMMDDHH, where HH is 00, 06, 12, 18. Defaults to "latest", which gets the most recent model uploaded to the server.
depth	How many model instances to return. This avoids having to download the entire model list (sometimes several hundred) if only the first few instances are required. Defaults to NULL, which returns everything. This input only makes sense when model.date != "latest".
verbose	Gives a detailed account of progress. Defaults to TRUE.

Value

forecasts$model.url
: URL to send to GribGrab for downloading data.

forecasts$model.run.date
: When the model was run.

forecasts$back.forecast
: Nearest forecast behind requested date.

forecasts$fore.forecast
: Nearest forecast after requested date.

forecasts$back.hr
: How many hours the back forecast is behind the requested date.

forecasts$fore.hr
: How many hours the fore forecast is in front of the requested date.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

RTModelProfile, BuildProfile, GribGrab

Examples

```
#Get the exact temperature profile of Chapel Hill, NC
#by performing a weighted average of GFS model forecasts.

#Figure out which forecasts to use
forecast.date <- as.POSIXlt(Sys.time(), tz = "GMT")

## Not run:
forecasts <- GetClosestForecasts(abbrev = "gfs_0p50", forecast.date)

## End(Not run)
```

```
#Get levels
pressure <- c(1, 2, 3, 5, 7,
10, 20, 30, 50, 70,
seq(100, 1000, by = 25))
levels <- paste(pressure, " mb", sep = "")

#Variables - temperature and height only
variables <- c("TMP", "HGT")

#Location
lon <- c(-79.052083)
lat <- c(35.907492)

#Get the data for each
resolution <- c(0.5, 0.5)
grid.type <- "latlon"

## Not run:
back.profile <- RTModelProfile(forecasts$model.url, forecasts$back.forecast,
   levels, variables, lon, lat, resolution = resolution, grid.type = grid.type)

fore.profile <- RTModelProfile(forecasts$model.url, forecasts$fore.forecast,
   levels, variables, lon, lat, resolution = resolution, grid.type = grid.type)

temps <- cbind(back.profile$profile[[1]][,2], fore.profile$profile[[1]][,2]) - 273.15
heights <- cbind(back.profile$profile[[1]][,1], fore.profile$profile[[1]][,1])
time.gap <- forecasts$fore.hr - forecasts$back.hr
exact.temp <- (temps[,1] * abs(forecasts$fore.hr) + temps[,2] * abs(forecasts$back.hr))/time.gap
exact.hgt <- (heights[,1] * abs(forecasts$fore.hr) + heights[,2] * abs(forecasts$back.hr))/time.gap

#Plot results
plot(c(min(temps), max(temps)), c(min(heights), max(heights)), type = "n",
    xlab = "Temperature (C)", ylab = "Height (m)")
points(temps[,1], heights[,1], pch = 1, col = 1)
points(temps[,2], heights[,2], pch = 2, col = 2)
lines(exact.temp, exact.hgt, col = 3, lty = 2)
legend("topleft", pch = c(1, 2, NA), lty = c(NA, NA, 2), col = c(1, 2, 3),
  legend = c(forecasts$back.forecast, forecasts$fore.forecast, as.character(Sys.time())))

## End(Not run)
```

GetDODSDates *Find available model run dates for data on the GrADS - DODS system.*

Description

This function checks the GrADS data server to see what dates and model subsets are available for model specified by ABBREV

Usage

```
GetDODSDates(abbrev, archive=FALSE, request.sleep=1)
```

Arguments

abbrev	A model abbreviation as specified in NOMADSRealTimeList or NOMADSArchiveList.
archive	Whether the model is on the NCEP real time server (FALSE) or on the NCDC model archive server (TRUE).
request.sleep	Seconds to pause between HTTP requests when scanning model pages - this prevents timeouts. Default 1.

Details

This function determines which dates are available for download for a particular model through the GrADS - DODS system. Once the user determines which dates are available, the output of this function can be passed to GetDODSModelRuns to determine which model runs can be downloaded.

Value

model	The model that was requested.
date	A list of model run dates available for download.
url	A list of URLs corresponding to the model run dates.

Note

Sometimes, sending lots of HTTP requests in rapid succession can cause errors. If messages resembling "Error: failed to load HTTP resource" appear, try increasing request.sleep. The code will take longer to execute but it will be more likely to finish successfully.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

GetDODSModelRuns, DODSGrab

Examples

```
#An example for the Global Forecast System 0.5 degree model

#Get the latest model url and date
abbrev <- "gfs_0p50"
## Not run:
urls.out <- GetDODSDates(abbrev)
print(paste("Most recent model run:",tail(urls.out$date, 1)))

#Get model dates from the GFS archive
```

```
abbrev <- "gfs-avn-hi"
urls.out <- GetDODSDates(abbrev, archive = TRUE, request.sleep = 1)

## End(Not run)
```

`GetDODSModelRunInfo` *Get model coverage and data information for models on GrADS-DODS system.*

Description

Given a URL from `GetDODSDates` and a model run from `GetDODSModelRuns`, get information on the model domain, levels, and variables.

Usage

```
GetDODSModelRunInfo(model.url, model.run)
```

Arguments

`model.url`	A URL for a model on the GrADS - DODS system, probably returned by `GetDODSDates`.
`model.run`	A specific model run, probably returned by `GetDODSModelRuns`

Details

This routine grabs information about the latitude, longitude, and time coverage of a specific model instance. It also finds data about levels (if present) and lists all the available variables (though they may not have data in them). The user can refer to this information to construct calls to the DODS system via `DODSGrab`.

Value

`model.info`	Information provided by the GrADS - DODS system about the given model instance.

Note

This function is very helpful in figuring out what's inside poorly documented models.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

`GetDODSDates`, `GetDODSModelRuns`, `DODSGrab`

Examples

```
#An example for the Global Forecast System 0.5 degree model

#Get some information about the latest model url and date, real time server
abbrev <- "gfs_0p50"
## Not run:
urls.out <- GetDODSDates(abbrev)
model.url <- tail(urls.out$url, 1)
model.runs <- GetDODSModelRuns(model.url)
model.info <- GetDODSModelRunInfo(model.url, tail(model.runs$model.run, 1))
print(model.info)
## End(Not run)
```

`GetDODSModelRuns` *Find available model runs on the GrADS - DODS system.*

Description

Given a URL from `GetDODSDates`, find which model runs are available for download on the GrADS - DODS system.

Usage

```
GetDODSModelRuns(model.url)
```

Arguments

`model.url`	A URL for a model on the GrADS - DODS system, probably returned by `GetDODSDates`.

Details

This function determines which dates are available for download for a particular model through the GrADS - DODS system. Once the user determines which dates are available, the output of this function can be passed to `GetDODSModelRuns` to determine which model runs can be downloaded.

Value

`model.run`	A list of model runs available for the requested date.
`model.run.info`	Information provided by the GrADS - DODS system about each model run.

Note

To get model run information for archived analysis models, pass URLs directly from `NOMADSArchiveList` directly to `GetDODSModelRuns`.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

GetDODSDates, DODSGrab, GetDODSModelRunInfo

Examples

```
#An example for the Global Forecast System 0.5 degree model

#Get the latest model url and date, real time server
abbrev <- "gfs_0p50"
## Not run:
urls.out <- GetDODSDates(abbrev)
model.url <- tail(urls.out$url, 1)
model.runs <- GetDODSModelRuns(model.url)
print(paste("Latest model run", tail(model.runs$model.run.info, 1)))

## End(Not run)
#Get model dates from the GFS analysis archive
abbrev <- "gfsanl"
model.url <- NOMADSArchiveList("dods", abbrev = abbrev)$url
## Not run:
model.runs <- GetDODSModelRuns(model.url)
print(model.runs$model.run.info)

## End(Not run)
```

GribGrab *Download grib file from the NOMADS server.*

Description

This function interfaces with the programming API at http://nomads.ncep.noaa.gov/ to download NOMADS model data. The available models can be viewed by calling NOMADSRealTimeList. The data arrives in grib (gridded binary) format that can be read with ReadGrib.

Usage

```
GribGrab(model.url, pred, levels, variables,
    local.dir = ".", file.name = "fcst.grb",
    model.domain = NULL, tidy = FALSE, verbose = TRUE,
    check.url = TRUE, download.method = NULL)
```

Arguments

model.url	The address of a model download page, probably from CrawlModels.
pred	The list of predictions (or model times) determined by the specific model from model.url
levels	A list of model levels to download.
variables	A list of model variables to download.
local.dir	Where to save the grib file, defaults to the current directory.
file.name	What to name the grib file, defaults to "fcst.grb".
model.domain	A vector of latitudes and longitudes that specify the area to return a forecast for. This is a rectangle with elements: west longitude, east longitude, north latitude, south latitude.
tidy	If TRUE, remove all files with the suffix ".grb" from local.dir prior to downloading a new grib file.
verbose	If TRUE, give information on connection status. Default TRUE
check.url	If TRUE, verify that the model URL is real and contains data. Default TRUE
download.method	Allows the user to set the download method used by download.file: "internal", "wget" "curl", "lynx". If NULL (the default), let R decide.

Value

grib.info$local.dir	The absolute path to the grib file that was downloaded.
grib.info$file.name	The name of the grib file that was downloaded.
grib.info$url	The URL that the grib file was downloaded from

Note

This requires the external programs wgrib2 and\or wgrib to be installed (depending on whether the files are in GRIB2 or GRIB format). Currently, these routines are only available on Unix/Linux and MacOS systems.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

http://nomads.ncep.noaa.gov/

See Also

CrawlModels, ParseModelPage, ReadGrib

Examples

```
#An example for the Global Forecast System 0.5 degree model

#Get the latest model url
## Not run:
urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 1)

#Get a list of forecasts, variables and levels
model.parameters <- ParseModelPage(urls.out[1])

#Figure out which one is the 6 hour forecast
#provided by the latest model run
#(will be the forecast from 6-12 hours from the current date)

my.pred <- model.parameters$pred[grep("06$", model.parameters$pred)]

#What region of the atmosphere to get data for
levels <- c("2 m above ground", "800 mb")

#What data to return
variables <- c("TMP", "RH") #Temperature and relative humidity

#Get the data
grib.info <- GribGrab(urls.out[1], my.pred, levels, variables)

#Extract the data
model.data <- ReadGrib(grib.info$file.name, levels, variables)

#Reformat it
model.grid <- ModelGrid(model.data, c(0.5, 0.5))

#Show an image of world temperature at ground level
image(model.grid$z[2, 1,,])

## End(Not run)
```

GribInfo *Get grib file inventory.*

Description

Find out what model, date, levels, and variables are contained in a grib file.

Usage

```
GribInfo(grib.file, file.type = "grib2")
```

Arguments

grib.file	Full path to a grib file.
file.type	Whether the file is in grib2 format ("grib2") or grib format ("grib").

Details

This function allows you to find out what is inside an unknown grib file. It does this by performing a system call to wgrib2 or wgrib.

Value

grib.info	Inventory of the grib file. If the input is in grib2 format, you also get the grid definition.

Note

In order to use this function, you need to have installed wgrib2 (for grib2 files) or wgrib (for grib files). You can find these here: http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/ and http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

ReadGrib, GetDODSModelRunInfo, GribGrab

Examples

```
## Not run:
#An example for the Global Forecast System 0.5 degree model

#Get the second latest model url, for stability
urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 2)

#Get a list of forecasts, variables and levels
model.parameters <- ParseModelPage(urls.out[2])

#Figure out which one is the 6 hour forecast
#provided by the latest model run
#(will be the forecast from 6-12 hours from the current date)

my.pred <- model.parameters$pred[grep("06$", model.parameters$pred)]

#What region of the atmosphere to get data for
levels <- c("2 m above ground", "800 mb")

#What data to return
variables <- c("TMP", "RH") #Temperature and relative humidity
```

```
#Get the data
grib.info <- GribGrab(urls.out[2], my.pred, levels, variables)

#Print out the inventory - it should match the requested data
grib.inv <- GribInfo(grib.info$file.name, "grib2")

## End(Not run)
```

MagnitudeAzimuth *Convert zonal-meridional wind speeds to magnitude/azimuth.*

Description

Given zonal (East-West) and meridional (North-South) wind speeds, calculate magnitude of wind vector and azimuth from north, in degrees.

Usage

```
MagnitudeAzimuth(zonal.wind, meridional.wind)
```

Arguments

zonal.wind	A vector of zonal (East-West) winds, west negative.
meridional.wind	A vector of meridional (North-South) winds, south negative.

Value

winds$magnitude	Magnitude of wind vector.
winds$azimuth	Azimuth of wind vector in degrees from North

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

Examples

```
zonal.wind <- c(35.5, -2)
meridional.wind <- c(-5, 15)
winds <- MagnitudeAzimuth(zonal.wind, meridional.wind)
print(winds$magnitude)
print(winds$azimuth)
```

ModelGrid *Transform model data into an array*

Description

This function takes output from ReadGrib or DODSGrab and produces an array with dimensions: levels x variables x longitudes x latitudes. This greatly reduces the size of the data set as well as makes it easier to manipulate. The data must be either in a regular latitude/longitude grid (like the GFS model, for example) or projected to a regular Cartesian grid.

Usage

```
ModelGrid(model.data, resolution, grid.type = "latlon",
    levels = NULL, variables = NULL,
    model.domain = NULL, cartesian.nodes=NULL)
```

Arguments

model.data	Output from ReadGrib.
resolution	Resolution of grid, in degrees if grid.type = "latlon", in kilometers if grid.type = "cartesian", as a 2 element vector c(East-West, North-South).
grid.type	Whether the grid is in lat/lon or cartesian. Options "latlon" or "cartesian".
levels	The model levels to include in the grid, if NULL, include all of them.
variables	The model variables to include in grid, if NULL, include all of them.
model.domain	A vector c(LEFT LON, RIGHT LON, TOP LAT, BOTTOM LAT) of the region to include in output. If NULL, include everything.
cartesian.nodes	If grid.type = "cartesian", this is a list of the x and y values that define the grid of the model, in kilometers. Define it as a list with elements x and y, e. g. list(x = my.x.values, y = my.y.values).

Details

If you set the spacing of lon.grid and/or lat.grid coarser than the downloaded model grid, you can reduce the resolution of your model, possibly making it easier to handle.

Value

z	An array of dimensions levels x variables x lon x lat; each level x variable contains the model grid of data from that variable and level
x	Vector of longitudes
y	Vector of latitudes
variables	The variables contained in the grid
levels	The levels contained in the grid
model.run.date	When the forecast model was run
fcst.date	The date of the forecast

Note

Only use this function when the model grid is regular. For example, the GFS high resolution model is 0.5 x 0.5 degree across its domain. Some models may have a regular Cartesian grid, in which case you must first project the data (see `GLOB.XY` in the `GEOmap` package), then define the Cartesian spacing using `cartesian.nodes`. I have provided this function as a convenience since I only use it for manipulating GFS model data. I am not sure how well it works for other models. Consider yourself warned!

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

`ReadGrib`, `BuildProfile`, `RTModelProfile`

Examples

```
## Not run:
#Get some example data
urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 1)
model.parameters <- ParseModelPage(urls.out[1])
levels <- c("2 m above ground", "100 mb")
variables <- c("TMP", "RH") #Temperature and relative humidity
grib.info <- GribGrab(urls.out[1], model.parameters$pred[1], levels, variables)
#Extract the data
model.data <- ReadGrib(grib.info$file.name, levels, variables)

#Make it into an array
gfs.array <- ModelGrid(model.data, c(0.5, 0.5))

#What variables and levels we have
print(gfs.array$levels)
print(gfs.array$variables)

#Find minimum temperature at the ground surface, and where it is
min.temp <- min(gfs.array$z[2, 1,,] - 273.15)
sprintf("%.1f", min.temp) #in Celsius

ti <- which(gfs.array$z[2, 1,,] == min.temp + 273.15, arr.ind = TRUE)

lat <- gfs.array$y[ti[1,2]] #Lat of minimum temp
lon <- gfs.array$x[ti[1,1]] #Lon of minimum temp

#Find maximum temperature at 100 mb atmospheric pressure
max.temp <- max(gfs.array$z[1, 1,,]) - 273.15
sprintf("%.1f", max.temp) #Brrr!

## End(Not run)
```

NOMADSArchiveList *Archived models available for download through rNOMADS*

Description

A list of abbreviations, names and URLs for the NOMADS models archived on the NCDC web site.. Users can refer to this list to find out more information about the available models, and rNOMADS uses the abbreviations to determine how to access the archives.

Usage

```
NOMADSArchiveList(url.type, abbrev = NULL)
```

Arguments

url.type	Determine whether to return a URL for extracting GRIB files ("grib") or for getting ascii format data directly from the server ("dods").
abbrev	Return information about the model that this abbreviation refers to. Defaults to NULL, in which case information about all the models available through rNOMADS.

Value

abbrevs	An abbreviation for each model
names	A full name for each model
urls	The web address of the download page for each model

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

NOMADSRealTimeList

Examples

```
#The archived model list in rNOMADS

archived.model.list <- NOMADSArchiveList("grib")
```

NOMADSRealTimeList *Models available for download through rNOMADS*

Description

Scans the NOMADS Real Time web site to generate a list of available model products. Users can refer to this list to find out more information about the available models, and rNOMADS uses the abbreviations to determine which URLs to scan and download.

Usage

```
NOMADSRealTimeList(url.type, abbrev = NULL)
```

Arguments

url.type	Determine whether to return a URL for extracting GRIB files ("grib") or for getting ascii format data directly from the server ("dods").
abbrev	Return information about the model that this abbreviation refers to. Defaults to NULL, in which case information about all the models available through rNOMADS.

Value

abbrevs	An abbreviation for each model
names	A full name for each model
urls	The web address of the download page for each model

Note

A big thanks to user hrbrmstr at Stack Overflow for helping with the NCEP web site parsing code.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

WebCrawler, ParseModelPage, NOMADSArchiveList, GribGrab, DODSGrab

Examples

```
## Not run:
#Grib filter
model.list <- NOMADSRealTimeList("grib")

#DODS interface
model.list <- NOMADSRealTimeList("dods")

## End(Not run)
```

ParseModelPage *Extract predictions, levels, and variables*

Description

This function parses the model download pages on NOMADS, and extracts information on predictions, levels, and variables available for each.

Usage

```
ParseModelPage(model.url)
```

Arguments

model.url	The URL of the model to extract information from, probably returned by NOMADSRealTimeList.

Details

This function scrapes the web page for a given model and determines which predictions, levels, and variables are present for each. Predictions are instances returned by each model (for example, the GFS model produces 3 hour predictions up to 192 hours from the model run). Levels are regions of the atmosphere, surface of the Earth, or subsurface that the model produces output for (for example the GFS model has a "2 m above ground" level that has data for temperature, etc, at that height across the Earth). Variables are types of data (temperature, for example).

Value

pred	Model predictions
levels	Locations of data points
variables	Data types

Note

Many of the names for predictions, levels, and variables are somewhat cryptic. Future versions of rNOMADS may have a reference function similar to NOMADSRealTimeList to help users with this issue.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

WebCrawler, ParseModelPage, GribGrab

Examples

```
#An example for the Global Forecast System 0.5 degree model

#Get the latest model url
## Not run:
urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 1)

#Get a list of forecasts, variables and levels
model.parameters <- ParseModelPage(urls.out[1])


## End(Not run)
```

`PlotWindProfile` *Plot wind speed and direction*

Description

Creates a polar plot showing the azimuth, elevation, and magnitude of winds.

Usage

```
PlotWindProfile(zonal.wind, meridional.wind, height, magnitude = NULL,
   magnitude.range = c(0, 50), height.range = c(0, 50000), points = TRUE, lines = FALSE,
   radial.axis = TRUE, elev.circles = NULL, elev.labels = NULL, radial.lines = NULL,
   colorbar = TRUE, colorbar.label = NULL, north.label = TRUE, invert = FALSE, ...)
```

Arguments

`zonal.wind`	A vector or list of vectors of zonal (East-West) winds, west negative.
`meridional.wind`	A vector or list of vectors of meridional (North-South) winds, south negative.
`height`	A vector or list of vectors of height at which each wind measurement was taken.
`magnitude`	A vector or list of vectors of magnitudes to plot at each wind azimuth, instead of using the wind magnitudes. This allows plotting of other data (such as the speed of sound) along the wind vectors. Defaults to `NULL`.
`magnitude.range`	Ranges of wind speed to plot. Values greater or lesser than these will be saturated. Defaults to `c(0, 50)`.
`height.range`	Ranges of heights to plot, values outside of this will not appear. Defaults to `c(0, 50000)`.
`points`	Whether to plot measurements as points. Defaults to `TRUE`.
`lines`	Whether to connect measurements together with lines. Defaults to `FALSE`.

radial.axis	Whether to plot an axis at the outer edge of the diagram. Defaults to TRUE.
elev.circles	Plot a dashed circles as elevation scales. Defaults to NULL, which plots nothing.
elev.labels	What labels to put on the elevation scales. Defaults to NULL, which means no labels.
radial.lines	Plot lines radiating from the center of the plot showing azimuth directions. Defaults to NULL, which plots nothing.
colorbar	If TRUE, plot a color bar. Defaults to TRUE.
colorbar.label	What label to put on the colorbar. Defaults to NULL, which means no label.
north.label	If TRUE, put a label denoting the north direction. Defaults to TRUE.
invert	Reverses the edge and the center of the plot, making elevations decrease towards the center. Defaults to FALSE.
...	This function supports some optional parameters as well: • r.axis - radius of plot axis • tick.len - length of azimuth ticks • r.axis.ticks - Whether or not to put tick marks on the outer axis • max.az - If plotting lines and the difference between two segments is greater than this value, interpolate between them to make things smooth • color.map - A list of colors to use, defaults to `rainbow(n.cols, start=0, end=5/6)` • n.cols - Number of color bins in color map • sub.col - Color of internal (elevation and azimuth) axes as a vector of length 2 • sub.lty - Type of internal axes, as a vector of length 2 • sub.lwd - Width of internal axes, as a vector of length 2 • elev.labels.az - Which azimuth to plot elevation labels on • point.cex - size of points, if plotted • pch - Plot character of points, if plotted • lty - Line style, if lines are selected • lwd - Line thickness, if lines are selected • colorbar.tick - Where to put labels on colorbar

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

Examples

```
## Not run:
download.file("http://www.unc.edu/~haksaeng/rNOMADS/myTA.RDATA",
   destfile = "myTA.RDATA")
load("myTA.RDATA")
#Find the latest Global Forecast System model run
model.urls <- GetDODSDates("gfs_0p50")
latest.model <- tail(model.urls$url, 1)
model.runs <- GetDODSModelRuns(latest.model)
```

```
latest.model.run <- tail(model.runs$model.run, 1)

#Get model nodes

lons <- seq(0, 359.5, by = 0.5)
lats <- seq(-90, 90, by = 0.5)
lon.ind <- which(lons <= max(myTA$lon + 360) & lons >= min(myTA$lon + 360)) - 1
lat.ind <- which(lats <= max(myTA$lat) & lats >= min(myTA$lat)) - 1
levels <- c(0, 46)
time <- c(0, 0)

#Get data
hgt.data <- DODSGrab(latest.model, latest.model.run, "hgtprs",
   time, c(min(lon.ind), max(lon.ind)), c(min(lat.ind), max(lat.ind)), levels)
ugrd.data <- DODSGrab(latest.model, latest.model.run, "ugrdprs",
   time, c(min(lon.ind), max(lon.ind)), c(min(lat.ind), max(lat.ind)), levels)
vgrd.data <- DODSGrab(latest.model, latest.model.run, "vgrdprs",
   time, c(min(lon.ind), max(lon.ind)), c(min(lat.ind), max(lat.ind)), levels)

#Reformat the data
hgt.grid <- ModelGrid(hgt.data, c(0.5, 0.5))
ugrd.grid <- ModelGrid(ugrd.data, c(0.5, 0.5))
vgrd.grid <- ModelGrid(vgrd.data, c(0.5, 0.5))

#Build profiles
zonal.wind <- c()
meridional.wind <- c()
height <- c()
for(k in 1:length(myTA$lon)) {
   hgt.profile <- BuildProfile(hgt.grid,
       myTA$lon[k], myTA$lat[k], spatial.average = TRUE)
   ugrd.profile <- BuildProfile(ugrd.grid,
      myTA$lon[k], myTA$lat[k], spatial.average = TRUE)
   vgrd.profile <- BuildProfile(vgrd.grid,
      myTA$lon[k], myTA$lat[k], spatial.average = TRUE)
   synth.hgt <- seq(min(hgt.profile),
       max(hgt.profile), length.out = 1000)
   ugrd.spline <- splinefun(hgt.profile, ugrd.profile, method = "natural")
   vgrd.spline <- splinefun(hgt.profile, vgrd.profile, method = "natural")
   zonal.wind[[k]] <- ugrd.spline(synth.hgt)
   meridional.wind[[k]] <- vgrd.spline(synth.hgt)
   height[[k]] <- synth.hgt
   print(paste("Finished", k, "of", length(myTA$lon), "profiles!"))
}

#Plot them all
PlotWindProfile(zonal.wind, meridional.wind, height, lines = TRUE,
    points = FALSE, elev.circles = c(0, 25000, 50000), elev.tick.labels = c(0, 25, 50),
    radial.lines = seq(45, 360, by = 45), colorbar = TRUE, invert = FALSE,
    point.cex = 2, pch = 19, lty = 1, lwd = 1,
    height.range = c(0, 50000), colorbar.label = "Wind Speed (m/s)")

## End(Not run)
```

ReadGrib *Extract data from grib file*

Description

This function wraps wgrib2 and wgrib, external grib file readers provided by the National Weather Service Climate Prediction Center (see http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/ and http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html). ReadGrib extracts forecast data into R. It does this by building an argument string, executing a system call to the appropriate external grib file reader, and extracting the result. Note that wgrib2 must be installed for ReadGrib to work for current grib files, and wgrib may need to be installed when looking at archived data.

Usage

```
ReadGrib(file.name, levels, variables, file.type = "grib2", missing.data=NULL)
```

Arguments

file.name	The path and file name of the grib file to read.
levels	The levels to extract.
variables	The variables to extract.
file.type	Whether the file is in GRIB ("grib1") or GRIB2 ("grib2") format. Default is "grib2".
missing.data	Replace missing data in grib archive with this value. If NULL, leave the data out. Only works with wgrib2. Default NULL.

Details

This function constructs system calls to wgrib and wgrib2. Therefore, you must have installed these programs and made it available on the system path. Unless you are interested in accessing archive data that's more than a few years old, you can install wgrib2 only. A description of wgrib2 and installation links are available at http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/ and http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html. Note that Windows users will likely have to use a UNIX emulator like Cygwin to install these programs. Also, rNOMADS is focused towards GRIB2 files; I have included GRIB1 format support as a convenience.

Value

model.data	A structure with a series of elements containing data extracted from the grib file.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

References

Ebisuzaki, W, Bokhorst, R., Hyvatti, J., Jovic, D., Nilssen, K, Pfeiffer, K., Romero, P., Schwarb, M., da Silva, A., Sondell, N., and Varlamov, S. (2011). wgrib2: read and write GRIB2 files. *National Weather Service Climate Prediction Center*, http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/

See Also

GribGrab, ArchiveGribGrab, ModelGrid

Examples

```
#Operational Forecast Data Extraction
#NCEP output is always in GRIB2 format - this makes things easy for us
#An example for the Global Forecast System 0.5 degree model

#Get the latest model url
## Not run:
urls.out <- CrawlModels(abbrev = "gfs_0p50", depth = 1)

#Get a list of forecasts, variables and levels
model.parameters <- ParseModelPage(urls.out[1])

#Figure out which one is the 6 hour forecast
#provided by the latest model run
#(will be the forecast from 6-12 hours from the current date)

my.pred <- model.parameters$pred[grep("06$", model.parameters$pred)]

#What region of the atmosphere to get data for
levels <- c("2 m above ground", "800 mb")

#What data to return
variables <- c("TMP", "RH") #Temperature and relative humidity

#Get the data
model.info <- GribGrab(urls.out[1], my.pred, levels, variables)

#Extract the data
model.data <- ReadGrib(model.info$file.name, levels, variables)

#Reformat it
model.grid <- ModelGrid(model.data, c(0.5, 0.5))

#Show an image of world temperature at ground level
image(model.grid$z[2, 1,,])


#Archived Data Extraction
#This is sometimes in GRIB1 format
#This example is in GRIB1
```

```
abbrev <- "gfsanl"
model.date <- 20040302 #March 2, 2004
model.run <- 18 #1800 GMT model run
pred <- 0 #Analysis

grib.info <- ArchiveGribGrab(abbrev, model.date, model.run,
    pred, file.type = "grib1")

model.data <- ReadGrib(grib.info$file.name, c("1000 mb"), c("TMP"),
    file.type = "grib1")

## End(Not run)
```

`RTModelProfile` *Get an atmospheric profile for a list of locations.*

Description

This routine simplifies the rapid generation of data for specific points on the Earth's surface. It currently requires `grib` support, but future versions will have GrADS-DODS support.

Usage

```
RTModelProfile(model.url, pred, levels, variables, lon, lat, resolution,
    grid.type, model.domain = NULL, spatial.average = FALSE, verbose = TRUE)
```

Arguments

`model.url`	The address of a model download page, probably from `CrawlModels`.
`pred`	The requested model prediction.
`levels`	A list of model levels to get for the profile.
`variables`	A list of model variables to download.
`lon`	Longitudes of points of interest.
`lat`	Latitudes of points of interest.
`resolution`	Resolution of model, in degrees if Lat/Lon, in kilometers if Cartesian, as a 2 element vector (ZONAL, MERIDIONAL)
`grid.type`	If the model is gridded in Lat/Lon or Cartesian units. Use `"latlon"` if Lat/Lon, `"cartesian"` if cartesian.
`model.domain`	A four element vector of latitudes and longitudes that defines a rectangular area to get data for. If `NULL`, the model domain will be 1 degree past the maximum and minimum defined by `lon` and `lat` arguments.
`spatial.average`	If `TRUE`, perform nearest neighbor interpolation for 4 grid nodes to get average profile at a specific point. If `FALSE`, get data from nearest grid node. Default `FALSE`.
`verbose`	If `TRUE`, provide information on the download process. Default `TRUE`.

Details

It is much more efficient to download a large chunk of data and extract profile points from that as opposed to downloading individual small model chunks in the vicinity of each point of interest. That is why I developed this function.

Value

`profile$profile.data`
: Table of requested values, with rows corresponding to the requested levels.

`profile$spatial.averaging`
: What kind of spatial interpolation was used, if any, for the profile calculations.

`profile$pred`
: The model prediction used for generating the profile.

`profile$model.date`
: When the model was run.

`profile$variables`
: Model variables, in the order presented in `profile$profile.data`

`profile$levels`
: Model levels, in the order presented in `profile$profile.data`

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

GetClosestForecasts, BuildProfile

Examples

```
#Get temperature profiles in Pantego, Chapel Hill, and Asheville, NC

#First, define each location
lon <- c(-76.662819, -79.052094, -82.550011)
lat <- c(35.589446, 35.907562, 35.591994)

#Get latest GFS 0.5 model, use analysis forecast
## Not run:
model.url <- CrawlModels(abbrev = "gfs_0p50", depth = 1)[1]
pred <- ParseModelPage(model.url)$pred[1]

## End(Not run)

#Get levels
pressure <- c(1, 2, 3, 5, 7,
10, 20, 30, 50, 70,
seq(100, 1000, by = 25))
levels <- paste(pressure, " mb", sep = "")

#Variables - temperature and height only
variables <- c("TMP", "HGT")
```

```
#Resolution of GFS is 0.5 x 0.5 degree
resolution <- c(0.5, 0.5)
grid.type <- "latlon"

#Get data
## Not run:
profile <- RTModelProfile(model.url, pred, levels, variables,
    lon, lat, resolution, grid.type, spatial.average = TRUE)

#Plot it
plot(c(-100, 50), c(0, 50000), type = "n", xlab = "Temperature (C)",
   ylab = "Height (m)", main = paste("GFS", profile$model.date,
   "GMT Analysis Forecast"))

for(k in seq_len(3)) {
   points(profile$profile.data[[k]][,2] - 273.15,
       profile$profile.data[[k]][,1], pch = k, col = k)
}
legend("topright", pch = 1:3, col = 1:3, legend = c(
   "Pantego, NC", "Chapel Hill, NC", "Asheville, NC"))

## End(Not run)
```

`WebCrawler` *Get web pages*

Description

Discover all links on a given web page, follow each one, and recursively scan every link found. Return a list of web addresses whose pages contain no links.

Usage

```
WebCrawler(url, depth = NULL, verbose = TRUE)
```

Arguments

`url`	A URL to scan for links.
`depth`	How many links to return. This avoids having to recursively scan hundreds of links. Defaults to `NULL`, which returns everything.
`verbose`	Print out each link as it is discovered. Defaults to `TRUE`.

Details

`CrawlModels` uses this function to get all links present on a model page.

Value

`urls.out` A list of web page addresses, each of which corresponds to a model instance.

Note

While it might be fun to try `WebCrawler` on a large website such as Google, the results will be unpredictable and perhaps disastrous if `depth` is not set. This is because there is no protection against infinite recursion.

Author(s)

Daniel C. Bowman <daniel.bowman@unc.edu>

See Also

`CrawlModels`, `ParseModelPage`

Examples

```
#Find the first 10 model runs for the
#GFS 0.5x0.5 model

## Not run: urls.out <- WebCrawler(
"http://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p50.pl", depth = 10)
## End(Not run)
```

Index