Package ‘wux’

October 12, 2022

Type Package

Title Wegener Center Climate Uncertainty Explorer

Version 2.2-1

Date 2016-12-14

Author Thomas Mendlik [aut, cre],
Georg Heinrich [aut],
Armin Leuprecht [aut],
Andreas Gobiet [ths]

Maintainer Thomas Mendlik <thomas.mendlik@uni-graz.at>

Depends R (>= 2.10), sp, ncdf4, reshape, abind, fields

Imports rgdal, rgeos, class, stringr, Hmisc, gdata, corpcor,
rworldmap, methods

Suggests lattice

Description Methods to calculate and interpret climate change signals and time series from climate multi-model ensembles. Climate model output in binary 'NetCDF' format is read in and aggregated over a specified region to a data.frame for statistical analysis. Global Circulation Models, as the 'CMIP5' simulations, can be read in the same way as Regional Climate Models, as e.g. the 'CORDEX' or 'ENSEMBLES' simulations. The package has been developed at the 'Wegener Center for Climate and Global Change' at the University of Graz, Austria.

License GPL (>= 2)

LazyLoad yes

LazyData yes

NeedsCompilation no

Repository CRAN

Date/Publication 2016-12-14 17:23:42

R topics documented:

`wux-package` *Wegener Center Climate Uncertainty Explorer*

Description

The WUX package is a toolbox to analyze climate change uncertainties projected by numerical model simulations.

The package provides methods to calculate and interpret climate change signals and time series from entire multi-model ensembles. Climate model output in binary NetCDF format is read in and aggregated to a `data.frame` for statistical analysis with tools provided by the R environment. The NetCDF format is not restricted to any specific type of climate model. Global circulation models (GCMs), as the CMIP5 or CMIP3 simulations, can be read in the same way as Regional Climate Models (RCMs), as e.g. the CORDEX or ENSEMBLES simulations.

Details

This package can currently perform following actions:

- Reading output of climate model simulations from NetCDF files, processing it, and writing it to a `data.frame` (the so-called WUX data frame).
- Various plotting options and summarizing utilities for a descriptive analysis of the projected climate change signals.

- Performing an Analysis of Variance (ANOVA) in order to estimate variance components.
- Performing a simple two-way linear data reconstruction, in order to fill the missing values of a simulation matrix as e.g. the GCM-RCM simulation matrix of ENSEMBLES.

I. Reading, processing, and writing of climate model ouput

Functions:	
`models2wux`	Reads NetCDF climate model output, processes it, and writes the results to a `data.fra`
`CMIP5fromESGF`	Automated downloading of the CMIP5 multi-model climate ensemble
`read.wux.table`	Reads in wux csv file obtained from `models2wux` from harddisk and creates a data fram
`AverageWuxDataFrame`	WUX data frame averaging function
Datasets:	
`userinput_CMIP5_changesignal`,	
`userinput_CMIP5_timeseries`,	
`modelinput_test`	Example config files for `models2wux`
`ensembles`, `ensembles_gcms`,	
`cmip3_2050`, `cmip3_2100`,	
`cmip5_2050`, `cmip5_2100`,	
`CMIP5_example_changesignal`,	
`CMIP5_example_timeseries`,	
`alpinesummer`	Example data frames calculatated by `models2wux`

II. Descriptive analysis of climate change signals

Descriptive analysis of multiple climate model simulations.

`summary.wux.df`	Summary statistics of the WUX data frame (`wux.df` class)
`plot.wux.df`	Scatter Plot
`hist.wux.df`	Density Plot
`plotAnnualCycle`	Annual Cycle Plot

III. Analysis of variance components

Extracts variance components of multiple climate model simulations using an ANOVA.

`aovWux`	ANOVA for WUX `data.frame`
`plot.wux.aov`	Barplot for `aovWux` output

IV. Reconstruction tools

Tools for filling missing values of an unbalanced climate model simulation matrix (e.g. missing RCM-GCM combinations of ENSEMBLES) in order to avoid biased ensemble estimates. Currently, the underlying linear reconstrtuction technique is based on solving the linear equation system

(LES) of the ANOVA design matrix (`method = "LES"`), or iterative linear reconstruction based on an ANOVA (`method = "Iterative"`) or Leave-one-out cross-calculation (`method = "IterativeCC"`).

`reconstruct`	Linear reconstruction of missing RCM-GCM combinations

Author(s)

Thomas Mendlik <`thomas.mendlik@uni-graz.at`>, Georg Heinrich <`g.heinrich@uni-graz.at`>, Andreas Gobiet <`andreas.gobiet@uni-graz.at`> and Armin Leuprecht <`armin.leuprecht@uni-graz.at`>

Maintainer: Thomas Mendlik <`thomas.mendlik@uni-graz.at`>

`alpinesummer` *Timeseries example of CMIP5 data over Greater Alpine Region in summer*

Description

An example temperature timerseries for one subregion Greater Alpine Region (GAR) `models2wux` in summer (JJA) of the CMIP5 ensemble for RCP 4.5. This is what a timeseries result from `models2wux` would look like.

Usage

```
data(alpinesummer)
```

See Also

`models2wux`

Examples

```
## thats what alpinesummer looks like
data("alpinesummer")
head(alpinesummer)
## it really is a timeseries! hooraay!

## get an idea what the data look like
require(lattice)
## Not run: xyplot(air_temperature ~ year|gcm,
       groups = acronym,
       data = alpinesummer,
       type = c("l", "g"),
       main = "JJA temperature of CMIP5 over Greater Alpine Region\nRCP 4.5 forcing")
## End(Not run)
## have fun playing around with the data :)
```

aovWux *Missing value reconstruction based on ANOVA*

Description

Calculates an analysis of variance (ANOVA) based on the specified model.

Usage

```
aovWux(model.formula = formula(model.formula), datain.df)
```

Arguments

model.formula	Model formula used for aov.
datain.df	WUX dataframe obtained from models2wux.

Value

Returns a object of class wux.aov, a list containing the ANOVA results for each subregion and season. The names of the list entries are "subreg = xx;season = yy".

Author(s)

Georg Heinrich <g.heinrich@uni-graz.at>

Examples

```
## read WUX test data
library("wux")
data(ensembles)

wuxtest.df <- subset(ensembles, subreg == "GAR")

## data reconstruction to obtain a balanced design
reconstruct.df <- reconstruct(wuxtest.df,
  factor1.name = "acronym", factor2.name = "gcm", data.name =
  "perc.delta.precipitation_amount")

## calculate ANOVA
anova.list <- aovWux(perc.delta.precipitation_amount ~ acronym +
  gcm, reconstruct.df)
```

AverageWuxDataFrame *WUX data frame averaging function*

Description

Collapses WUX data frame by averaging over specified factor (column name). The chosen data frame column will disappear after aggregation.

This function is primarily used to average over model runs (see example).

Usage

```
AverageWuxDataFrame(x, INDEX, fun = "mean")
```

Arguments

x	wux data.frame (returned by models2wux or read.wux.table)
INDEX	character column names from wux data.frame over which aggregation should take place. Those columns will dissapear after aggregation
fun	keyword for aggregation function. Default is mean

Author(s)

Thomas Mendlik <thomas.mendlik@uni-graz.at>

Examples

```
## load WUX and read WUX test data
require(wux)
data(cmip3_2050)

## average over runs
cmip3.avg.runs <- AverageWuxDataFrame(cmip3_2050, "gcm.run")
## average over seasons, runs and subregions
cmip3.avg.all <- AverageWuxDataFrame(cmip3_2050, INDEX = c("gcm.run", "subreg", "season"))
```

cmip3_2050 *Climate Change signals for CMIP3 ensemble*

Description

This dataset contains air temperature and precipitation climate change signals of all climate simulations from the CMIP3 project from 1961-1990 to 2021-2050. Subregions are defined according to the CORDEX project. Subregion EU.ENS contains the European region defined in the ENSEMBLES project and World contains the entire earth. http://wcrp.ipsl.jussieu.fr/SF_RCD_CORDEX.html.

Details

This dataset is an exemplary output of `models2wux`.

Source

CMIP3 project: http://www-pcmdi.llnl.gov

References

Meehl, G. A., C. Covey, T. Delworth, M. Latif, B. McAvaney, J. F. B. Mitchell, R. J. Stouffer, and K. E. Taylor, 2007: The WCRP CMIP3 multi-model dataset: A new era in climate change research, Bulletin of the American Meteorological Society, 88, 1383-1394.

Examples

```
require(wux)
data(cmip3_2050)

str(cmip3_2050)
summary(cmip3_2050)

## Not run: plot(cmip3_2050,  "perc.delta.precipitation_amount",
              "delta.air_temperature",  subreg.subset = "CORDEX.Africa",
              boxplots = TRUE, xlim = c(-10,10), label.only.these.models = "",
              ylim = c(0, 3), xlab = "Precipitation Amount [%]",
              ylab = "2-m Air Temperature [K]", draw.legend = FALSE,
              draw.median.lines = FALSE,
              main = "CMIP3 2-m Air Temp. and Precip. Amount")

## End(Not run)
```

`cmip3_2100` *Climate Change signals for CMIP3 ensemble*

Description

This dataset contains air temperature and precipitation climate change signals of all climate simulations from the CMIP3 project from 1961-1990 to 2071-2100. Subregions are defined according to the CORDEX project. Subregion `EU.ENS` contains the European region defined in the ENSEMBLES project and `World` contains the entire earth. http://wcrp.ipsl.jussieu.fr/SF_RCD_CORDEX.html.

Details

This dataset is an exemplary output of `models2wux`.

Source

CMIP3 project: http://www-pcmdi.llnl.gov

References

Meehl, G. A., C. Covey, T. Delworth, M. Latif, B. McAvaney, J. F. B. Mitchell, R. J. Stouffer, and K. E. Taylor, 2007: The WCRP CMIP3 multi-model dataset: A new era in climate change research, Bulletin of the American Meteorological Society, 88, 1383-1394.

Examples

```
require(wux)
data(cmip3_2100)

str(cmip3_2100)
summary(cmip3_2100)

## Not run: plot(cmip3_2100,  "perc.delta.precipitation_amount",
              "delta.air_temperature",  subreg.subset = "CORDEX.Africa",
              boxplots = TRUE, xlim = c(-20,20), label.only.these.models = "",
              ylim = c(0,5), xlab = "Precipitation Amount [%]",
              ylab = "2-m Air Temperature [K]", draw.legend = FALSE,
              draw.median.lines = FALSE,
              main = "CMIP3 2-m Air Temp. and Precip. Amount")

## End(Not run)
```

`CMIP5fromESGF` *Downloads CMIP5 climate simulations from the ESGF data portal*

Description

Downloads available monthly CMIP5 simulations from ESGF data portal. You need an account at any ESGF node (see `http://cmip-pcmdi.llnl.gov/cmip5/data_getting_started.html`). This function creates subdirectories for each climate simulation in the specified folder, automatically recieves the bash scripts needed for the partiular simulation-variable-experiment combination and then executes the bash scripts one by one. An external PYTHON script is called for this task. If either data or bash-scripts exist, the download will be skipped. Use this function with care. You need a working internet connection for this function to work.

Usage

```
CMIP5fromESGF(save.to = NULL, variables = NULL,
              experiments = NULL, models = NULL)
```

Arguments

`save.to`	Directory location for downloading CMIP5 data and bash scripts. ATTENTION: subdirectories for each model-experiment combination will be created!

`variables`	Short variable names for meteorological parameters of interest (e.g. "tas" for 2m air temperature or "pr" for precipitation amount). See e.g. the IPCC Standard Output from GCMs (`http://www-pcmdi.llnl.gov/ipcc/standard_output.html`).
`experiments`	Experiment of the climate simulation (e.g. c("historical", "rcp45"), see Taylor (2012) for a detailed description.
`models`	Climate simulations to be downloaded. If no models are provided (default), all available simulations will be retrieved. See the "Model" column at `http://cmip-pcmdi.llnl.gov/cmip5/availability.html` for available simulations. ATTENTION: This is a considerable amount of data, so watch out for your diskspace!

Details

Firstly you need an ESGF account. If you do not have any, start here: `http://cmip-pcmdi.llnl.gov/cmip5/data_getting_started.html`

This function calls an external python script which

1. looks for the latest CMIP5 models at ESGF `http://pcmdi9.llnl.gov`
2. generates a local directory structure where the data will be stored
3. receives the corresponding bash files from ESGF
4. executes the bash files.

You can find the location of the script with `system.file("exec", "CMIP5_downloader.py", package="wux")`

This function is an alternative to downloading the corresponding models by point-and-click on the ESGF node (as `http://pcmdi9.llnl.gov` or `http://esgf-data.dkrz.de`), as it takes advantage of the ESGF search API mechanism for automated data screening and wget-script generation. For thei nterested user more information on downloading strategies is available at `https://github.com/ESGF/esgf.github.io/wiki/ESGF_Data_Download_Strategies`.

Warning

This function automatically creates new directories on your system, downloads wget scripts, flags them execueable and runs them, which can download quite some data.

Note

Use with care, your harddisk might get stuffed. This tools works on unix platforms only.

Author(s)

Thomas Mendlik `<thomas.mendlik@uni-graz.at>`

References

Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl, 2012: An Overview of CMIP5 and the Experiment Design. Bull. Amer. Meteor. Soc., 93, 485-498. doi: http://dx.doi.org/10.1175/BAMS-D-11-00094.1

Examples

```
## Not run:
## download temperature fields of two example GCMs (NorESM1-M and
## CanESM2) with the  RCP 8.5 and the historical run
## into your temporary directory. This command will create a folder
## "CMIP5" in "~/tmp" with two subfolders for each model again with
## two subfolders for each experiment.
CMIP5fromESGF(save.to = "~/tmp/CMIP5/",
              models = c("NorESM1-M", "CanESM2"),
              variables = c("tas"),
              experiments= c("historical", "rcp85"))

## End(Not run)
```

`CMIP5toModelinput` *Creates a "modelinput" input file based on CMIP5 data on your disk to be further processed by models2wux.*

Description

This function creates a "modelinput" list (written into a file). That file can then be used to have all the relevant CMIP5 model information available for "models2wux". The input for this function is the directory where `CMIP5fromESGF` saves all the data. It is crucial for this funcion that the data is saved in the same directory strucure as created by "CMIP5fromESGF" (GCMname/RCP) . This function works for CMIP5 data on monthly basis only and can currently process only the parameters "air_temperature" and "precipitation_amount".

Usage

```
CMIP5toModelinput(filedir = NULL, save.to = NULL, verbose = FALSE)
```

Arguments

	filedir: Direcotry where the CMIP5 data are being stored. It is crucial for this funcion that the data is saved in the same directory strucure as created by "CMIP5fromESGF". save.to: character. Filename to safe the modelinupt. verbose: boolean. For additional information printed on the screen.
	Direcotry where the CMIP5 data are being stored. It is crucial for this funcion that the data is saved in the same directory strucure as created by `CMIP5fromESGF`.
`filedir` `save.to`	character. Filename to safe the modelinupt.
`verbose`	boolean. For additional information printed on the screen.

Details

This function is based on the data obtained by `CMIP5fromESGF`. However, it is also possible to download the data manually and run this function. The directory structure must look like this: "GCMname/RCP" (e.g. `/tmp/CMIP5/NorESM1-M/rcp85`).

In the background a PYTHON script is executed. You can find the file by typing the command `system.file("exec", "cmip5_to_wux_modeldict.py", package="wux")`

Note

This tools works on unix platforms only.

Author(s)

Thomas Mendlik `<thomas.mendlik@uni-graz.at>`

References

Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl, 2012: An Overview of CMIP5 and the Experiment Design. Bull. Amer. Meteor. Soc., 93, 485-498. doi: http://dx.doi.org/10.1175/BAMS-D-11-00094.1

Examples

```
## Not run:
## download temperature fields of two example GCMs (NorESM1-M and
## CanESM2) with the  RCP 8.5 and the historical run
## into your temporary directory. This command will create a folder
## "CMIP5" in "~/tmp" with two subfolders for each model again with
## two subfolders for each experiment.
CMIP5fromESGF(save.to = "~/tmp/CMIP5/",
              models = c("NorESM1-M", "CanESM2"),
              variables = c("tas"),
              experiments= c("historical", "rcp85"))

## create the corresponding modelinput list:
CMIP5toModelinput(filedir = "~/tmp/CMIP5",
                  save.to = "~/tmp/CMIP5_modelinput.R")


## End(Not run)
```

`cmip5_2050` *Climate Change signals for CMIP5 ensemble*

Description

This dataset contains air temperature and precipitation climate change signals of all climate simulations from the CMIP5 project from 1961-1990 to 2021-2050. Subregions are defined according to the CORDEX project. Subregion `EU.ENS` contains the European region defined in the ENSEMBLES project and `World` contains the entire earth. `http://wcrp.ipsl.jussieu.fr/SF_RCD_CORDEX.html`.

Details

This dataset is an exemplary output of `models2wux`.

Source

CMIP5 project: http://www-pcmdi.llnl.gov

References

Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl, 2012: An Overview of CMIP5 and the Experiment Design. Bull. Amer. Meteor. Soc., 93, 485-498. doi: http://dx.doi.org/10.1175/BAMS-D-11-00094.1

Examples

```
require(wux)
data(cmip5_2050)

str(cmip5_2050)
summary(cmip5_2050)

tas.range <- c(0, 2.5)
pr.range <- c(-10, 15)
## Not run: plot(cmip5_2050, "delta.air_temperature",
              "perc.delta.precipitation_amount", boxplots = TRUE,
              ylim = pr.range, xlim = tas.range, ylab = "Precipitation Amount [
              xlab = "2-m Air Temperature [K]", draw.legend = TRUE,
              draw.median.lines = FALSE, subreg.subset = "CORDEX.Africa",
              main = "CMIP5 2-m Air Temp. and Precip. Amount 1961-1990 to 2021-2050",
              label.only.these.models = "", draw.seperate.legend = TRUE,
              copyright = TRUE, horiz.box.col = "coral", vert.box.col = "cyan")

## End(Not run)
```

cmip5_2100 *Climate Change signals for CMIP5 ensemble*

Description

This dataset contains air temperature and precipitation climate change signals of all climate simulations from the CMIP5 project from 1961-1990 to 2071-2100. Subregions are defined according to the CORDEX project. Subregion `EU.ENS` contains the European region defined in the ENSEMBLES project and `World` contains the entire earth. http://wcrp.ipsl.jussieu.fr/SF_RCD_CORDEX.html.

Details

This dataset is an exemplary output of `models2wux`.

Source

CMIP5 project: http://www-pcmdi.llnl.gov

References

Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl, 2012: An Overview of CMIP5 and the Experiment Design. Bull. Amer. Meteor. Soc., 93, 485-498. doi: http://dx.doi.org/10.1175/BAMS-D-11-00094.1

Examples

```
require(wux)
data(cmip5_2100)

str(cmip5_2100)
summary(cmip5_2100)

tas.range <- c(0, 2.5)
pr.range <- c(-10, 15)
## Not run: plot(cmip5_2100, "delta.air_temperature",
              "perc.delta.precipitation_amount", boxplots = TRUE,
              ylim = pr.range, xlim = tas.range, ylab = "Precipitation Amount [
              xlab = "2-m Air Temperature [K]", draw.legend = TRUE,
              draw.median.lines = FALSE, subreg.subset = "CORDEX.Africa",
              main = "CMIP5 2-m Air Temp. and Precip. Amount 1961-1990 to 2071-2100",
              label.only.these.models = "", draw.seperate.legend = TRUE,
              copyright = TRUE, horiz.box.col = "coral", vert.box.col = "cyan")

## End(Not run)
```

CMIP5_example_changesignal

Climate change signals of example userinput for models2wux

Description

This example of a WUX data.frame is the result of running `userinput_CMIP5_changesignal` with `models2wux`.

Usage

```
data(CMIP5_example_changesignal)
```

Details

You can download the NetCDF files from ESGF using `CMIP5fromESGF`.

See Also

`models2wux`

Examples

```
## thats what CMIP5_changesignal looks like
data("CMIP5_example_changesignal")
CMIP5_example_changesignal

## You can run models2wux to get the same result as
## above.
data(userinput_CMIP5_changesignal)
data(modelinput_test)
## Not run:
## You must have downloaded the example NetCDF files according to
## "modelinput_test" in order to run "models2wux", or you will get an
error message. See the examples of ?CMIP5fromESGF or ?modelinput_test.
CMIP5_example_changesignal <- models2wux(userinput_CMIP5_changesignal,
                                modelinput = modelinput_test)
## End(Not run)
```

`CMIP5_example_timeseries` *Climate change signals of example userinput for models2wux*

Description

This example of a WUX data.frame is the result of running `userinput_CMIP5_timeseries` with `models2wux`.

Usage

```
data(CMIP5_example_timeseries)
```

Details

You can download the NetCDF files from ESGF using `CMIP5fromESGF`.

See Also

`models2wux`

Examples

```
## thats what CMIP5_timeseries looks like
data("CMIP5_example_timeseries")
head(CMIP5_example_timeseries)

## You can run models2wux to get the same result as
## above.
data(userinput_CMIP5_timeseries)
data(modelinput_test)
```

```
## Not run:
## You must have downloaded the example NetCDF files according to
## "modelinput_test" in order to run "models2wux". See the examples of
## ?CMIP5fromESGF or ?modelinput_test.
CMIP5_example_timeseries <- models2wux(userinput_CMIP5_timeseries,
                                modelinput = modelinput_test)
## End(Not run)
```

`ensembles` *ENSEMBLES dataset*

Description

This dataset contains air temperature and precipitation climate change signals of all 22 A1B forced climate simulations from the ENSEMBLES project from 1961-1990 to 2021-2050.

Usage

```
data(ensembles)
```

Source

The ENSEMBLES project: `http://www.ensembles-eu.org/`

References

van der Linden P, Mitchell JFB. 2009. ENSEMBLES: Climate Change and its Impacts: Summary of research and results from the ENSEMBLES project. Met Office Hadley Centre: Exeter.

Examples

```
require(wux)
data(ensembles)
ensembles <- droplevels(subset(ensembles, subreg == "EU.ENS"))

str(ensembles)
parms <- c("delta.air_temperature", "perc.delta.precipitation_amount",
           "delta.global_radiation", "delta.wind_speed")
summary(ensembles, parms = parms)

## Not run: plot(ensembles,  "perc.delta.precipitation_amount",
              "delta.air_temperature", boxplots = TRUE, xlim = c(-10,10),
              ylim = c(0, 3),
              label.only.these.models = c(""),
              xlab = "Precipitation Amount [%]",
              ylab = "2-m Air Temperature [K]",
              main = "CCS 2-m Air Temp. and Precip. Amount 1961-90 to 2021-50",
              subreg.subset = c("EU.ENS"))
## End(Not run)
## comparing ENSEMBLES RCMs with its driving GCMs
data(ensembles_gcms)
ensembles_gcms.eu <- gdata::drop.levels(subset(ensembles_gcms, subreg ==
"EU.ENS"))
gcm.names <- levels(ensembles_gcms.eu$acronym)

vars.of.interest <- !names(ensembles) %in% c("delta.global_radiation", "delta.wind_speed")
ensembles <- ensembles[vars.of.interest]
ensembles.merge <- rbind(ensembles, ensembles_gcms.eu)
summary(ensembles.merge)

## Not run: plot(ensembles.merge,  "perc.delta.precipitation_amount",
              "delta.air_temperature", boxplots = TRUE, xlim = c(-10,10),
              ylim = c(0, 3),
              label.only.these.models = gcm.names,
              xlab = "Precipitation Amount [%]", ylab = "2-m Air Temperature [K]",
              main = "CCS 2-m Air Temp. and Precip. Amount 1961-90 to 2021-50",
              subreg.subset = c("EU.ENS"), draw.median.lines = FALSE)

## End(Not run)
```

ensembles_gcms *GCM forcing data from the ENSEMBLES project*

Description

This dataset contains air temperature and precipitation climate change signals of the 8 A1B driving GCMs used as boundary conditions for the ENSEMBLES RCMs. The climate change signal is from 1961-1990 to 2021-2050.

Usage

```
data(ensembles_gcms)
```

Source

ENSEMBLES project: http://www.ensembles-eu.org/ CMIP3 project: http://www-pcmdi.llnl.gov

References

van der Linden P, Mitchell JFB. 2009. ENSEMBLES: Climate Change and its Impacts: Summary of research and results from the ENSEMBLES project. Met Office Hadley Centre: Exeter.

See Also

cmip3_2050, ensembles, models2wux

Examples

```
require(wux)
data(ensembles_gcms)
ensembles.gcm.names <- levels(ensembles_gcms$acronym) #8 GCM names
summary(ensembles_gcms)

## Now lets compare this dataset to the CMIP3 ensemble
data(cmip3_2050)     # GCMs of CMIP3 ensemble
cmip3_2050.sub <- subset(cmip3_2050, subreg %in% c("World", "EU.ENS")
                       & em.scn == "A1B")
cmip3_2050.sub <- droplevels(cmip3_2050.sub)
## "mpi_echam5-r3", "bccr_bcm2_0-r1", "ipsl_cm4-r2" can be found
## in the ensembles_gcms dataset as well as in the cmip3_2050 dataset
## so we delete it from one of these dataset
ensembles_gcms.sub <- subset(ensembles_gcms, !acronym %in%
                                c("mpi_echam5-r3", "bccr_bcm2_0-r1",
                                  "ipsl_cm4-r2"))
ensembles_gcms.sub <- gdata::drop.levels(ensembles_gcms.sub)
## combine cmip3 and ENSEMBLES GCMs in one data.frame
gcms.combined <- rbind(ensembles_gcms.sub, cmip3_2050.sub)

## Scatterplot
prec.range <- range(gcms.combined$perc.delta.precipitation_amount) + c(-1, 1)
tas.range <- range(gcms.combined$delta.air_temperature)
## Not run: plot(gcms.combined,
               "perc.delta.precipitation_amount", "delta.air_temperature",
               subreg.subset = "EU.ENS", draw.median.lines = FALSE,
               label.only.these.models = ensembles.gcm.names,
               xlim = prec.range,
               ylim = tas.range,
               main = "GCMs from ENSEMBLES project within CMIP3 SRESA1B ensemble",
               draw.seperate.legend = TRUE)

## End(Not run)
```

---

hist *Plots histograms and kernel density estimates*

Description

`hist` plots either one or two histograms and the according kernel density estimates using `density`.

This plotting routine extracts all the information from the input dataframe which has to be 'WUX-style' (see `models2wux`).

Usage

```
## S3 method for class 'wux.df'
hist(x, datain2.df = NULL, var.name = "delta.air_temperature",
subreg.subset = NULL, season.subset = NULL, plot.density = TRUE,
hist1.col = "red", hist2.col = "blue", bw = "nrd0", kernel = "gaussian",
mark.df = NULL, plot.legend = FALSE,  xlim = NULL, ylim = NULL,
xtick.number = 10, ytick.number = 10, xminor.tick = FALSE, yminor.tick =
FALSE, xlab = NULL, ylab = "Probability Density", main =
NULL, out.file.directory = NULL, out.file.name = NULL, copyright = FALSE, ...)
```

Arguments

`x`	1st WUX dataframe obtained from `models2wux`.
`datain2.df`	2nd WUX dataframe obtained from `models2wux`.
`var.name`	Character string of parameter in WUX dataset. Default is temperature change.
`subreg.subset`	Vector of subregions to be plotted (e.g. `c("EU.ENS", "GAR")`).
`season.subset`	Vector of seasons to be plotted (e.g. `c("MAM", "DJF")`).
`plot.density`	Boolean. Indicating if kernel density estimates should be plotted. Default is `TRUE`.
`hist1.col`	Character string of the 1st histogram color (e.g. `"red"`).
`hist2.col`	Character string of the 2nd histogram color (e.g. `"blue"`).
`bw`	The smoothing bandwidth to be used in `density`. Default is `"nrd0"`.
`kernel`	A character string giving the smoothing kernel to be used in `density`. This must be one of `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`,`"biweight"`, `"cosine"` or `"optcosine"` with default `"gaussian"`.
`mark.df`	Subset of WUX dataframe indicating the models to be marked.
`plot.legend`	Boolean. Indicating if a plot legend indicating the models of mark.df and sample size should be plotted. Default is `FALSE`.
`xlim`	Range vector for the x-axis.
`ylim`	Range vector for the y-axis.
`xtick.number`	Number of ticks for the x-axis with default `10`.
`ytick.number`	Number of ticks for the y-axis with default `10`.
`xminor.tick`	Boolean. Indicating if minor ticks for the x-axis should be plotted. Default is `FALSE`.
`yminor.tick`	Boolean. Indicating if minor ticks for the y-axis should be plotted. Default is `FALSE`.
`xlab`	Label for x-axis.
`ylab`	Label for y-axis with default `Probability Density`.
`main`	Main title.
`out.file.directory`	String of the directory where the plots are exported (e.g. `"/tmp/plots/"`). If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displayed on screen.

`out.file.name`	Prefix of the file names of the plots. Files will be stored as `out.file.name_subreg_season.eps`, where `subreg` is one realization of the `subreg.subset` argument and season is one realization of `season.subset`. For example: `out.file.name = "histogram"` will store to the files to `histogram_EUROPE_DJF.eps` and `histogram_EUROPE_JJA.eps`. If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displyed on screen.
`copyright`	Boolean. If a copyright message should be plotted. Default is FALSE.
`...`	Further optional arguments passed to `hist`.

Author(s)

Georg Heinrich <`g.heinrich@uni-graz.at`>

Examples

```
## load WUX and read WUX test data
require(wux)
data(ensembles)

wuxtest.df <- subset(ensembles, subreg == "GAR")

## set dataframe for model marks
mark.df <- subset(wuxtest.df, acronym %in% c("ICTP-REGCM3", "MPI-M-REMO"))
mark.df <- droplevels(mark.df)

## histogram plot
## Not run: hist(x = wuxtest.df, var.name =
  "perc.delta.precipitation_amount", xlim = c(-50,50), ylim = c(0,0.12),
  xtick.number = 9, xminor.tick = TRUE,  ytick.number = 5,
  yminor.tick = TRUE, xlab = "Precipitation Amount [%]", main =
  "WUX histogram", plot.legend = TRUE, mark.df = mark.df, hist1.col =
  "dark blue")

graphics.off()
## End(Not run)
```

`modelinput_test` *Example model specification file for models2wux*

Description

This is an example model specification for WUX, giving information on two example NetCDF files from the CMIP5 project. The datatype is a list. For specification details see the "Configfile modelinput" section in `models2wux`.

Usage

```
data(modelinput_test)
```

Details

The two CMIP5 simulations "NorESM1-M" and "CanESM2" (either having historical run and RCP 8.5 forcing) are assumed to be stored in the home directory "~/tmp/CMIP5". You can download them using CMIP5fromESGF. See the example for a typical workflow.

See Also

models2wux, CMIP5fromESGF

Examples

```
## Not run:
## Here is how to use the climate model specification file
## "modelinput_test" for models2wux.

## I) DOWNLOAD EXEMPLARY DATA
## download temperature fields of two example GCMs (NorESM1-M and
## CanESM2) with the  RCP 8.5 and the historical run
## into your temporary directory. This command will create a folder
## "CMIP5" in "~/tmp" with two subfolders for each model again with
## two subfolders for each experiment.
CMIP5fromESGF(save.to = "~/tmp/CMIP5/",
              models = c("NorESM1-M", "CanESM2"),
              variables = c("tas"),
              experiments= c("historical", "rcp85"))

## II) INTERFACE FOR THE DATA (type list)
## This is the information for models2wux to read in the data. Usually
## you have to create such a file for yourself, or add it to an existing
## one. This file assumes you have downloaded the two GCMs into ~/tmp/CMIP5 by
## CMIP5fromESGF, as shown above.
data(modelinput_test)

## III) CONTROL FILE FOR models2wux (type list)
## What climate data you want to read in (here it is the two example
## simluations mentioned above)? What subregion to analysze? What is you
## reference and what your scenario period? Aggregate to specific
## seasons?
data(userinput_CMIP5_timeseries)

## IV) CONVERT CLIMATE SIMULATIONS TO A data.frame
wux.test <- models2wux(userinput_CMIP5_timeseries, modelinput_test)

## V) ANALYZE data.frame
require(lattice)
wux.test$year <- as.integer(as.character(wux.test$year))
xyplot(air_temperature ~ year|season,
       groups=acronym,
       data = wux.test,
       type = c("l", "g"),
       main = "NorESM1-M and CanESM2 simulations over Alpine Region\n
       historical and RCP 8.5 forcing")
```

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

models2wux — *Processing climate model output*

Description

Reads various climate model NetCDF outputs, processes them according to `userinput`, and writes the processed data to a `data.frame`.

The `data.frame` output of WUX (the WUX data frame) contains the climate change signals for user-specified periods, regions, seasons, and parameters for each of the indicated climate models as defined in `userinput`.

The `userinput` is a named `list` object or a file containing a named `list`. It passes the controlling parameters to `models2wux`. The file paths, file names and meta-information on the climate simulations are stored in another list called `modelinput`. See the "Details" section and the "Configfile userinput" and "Configfile modelinput" section for a detailed description of these two lists.

Usage

```
models2wux(userinput, modelinput)
```

Arguments

`userinput`	The specification of e.g. the parameters, periods, aggregation statistics, seasons, subregions, and climate models to be processed. This is either a file name containing a list which will be sourced internally, or a list object.
`modelinput`	The specifications of file paths, file names and meta-information of every single climate simulation output you have stored on your HDD. This is either a file name containing a list which will be sourced internally, or a list object.

Details

To process a climate multimodel ensemble of your choice, `models2wux` needs two config files `userinput` and `modelinput`, both being named list objects or files containing a named list.

`modelinput` stores general information about your climate data, i.e. the locations of the NetCDF files and their filenames. It also safes certain metainformation for the specific climate simulations (e.g. a unique acronym for the simulation; the developing institution; the radiative forcing). Usually the `modelinput` information should be stored in a single file on your system and should be updated when new climate simulations come in. It is advisable to share this file with your collegues if you work with the same NetCDF files on a shared IT infrastructure.

`userinput` contains information on what you actually want `models2wux` to be doing for you, mainly, which climate simulations defined in `modelinput` should be processed and what kind of statistic should be performed. You also define the geographical regions of interest you want to investigate and what time horizon you want to regard. Here is an overview of all possible tags a `userinput` list contains:

`parameter.names`	Specification of parameters to process.
`reference.period`	Specification of the reference period.
`scenario.period`	Specification of the scenario period.
`temporal.aggregation`	Specification of the temporal aggregation of the climate models (e.g. monthly mean or season sum
`subregions`	Specification of subregions.
`area.fraction`	Take parts of model-pixels according to subregion coverage.
`spatial.weighting`	Cosine areal weighting of regular grid.
`na.rm`	Behavior for missing values of timeslices.
`plot.subregions`	Specifies diagnostic plotting of grid points within the subregions.
`save.as.data`	Specification of output directory and filename.
`climate.models`	Specification of climate models to be processed.

This is what `models2wux` is doing: First, `models2wux` extracts attributes set in the `userinput` list and loads the corresponding model information (storage paths, filenames, ...) from the `modelinput` list. It then retrieves the geographical boundaries of the specified regions in `subregions` (here the model gridfiles are introduced) and reads the specified parameter data from the NetCDF files within the boundaries of the actual subregion. Subsequently, `models2wux` *aggregates over the time dimension* by the indicated months for the specified periods and calculates either the climatological mean values of the reference and future period and the according climate change signals or time series. Next, `models2wux` *aggregates over the spatial dimension*. `models2wux` repeats these processing steps for each model specified in `climate.models`, each parameter in `parameter.names`, each subregion in `subregions`, and each period in `reference.period` and `scenario.period`, respectively. Finally, the processed data is written to a `data.frame` and stored to the hard disk as indicated by `save.as.data`.

For more detailed information on `modelinput` and `userinput` see the corresponding sections `Configfile "modelinput"` and `Configfile "userinput"` in this help page.

Value

A `data.frame` of class `c("wux.df", "data.frame")` containing climate change signals for all models, subregions, and parameters specified in `userinput`. It also writes a csv file on your HDD.

Configfile "userinput"

Those are specifications the user provides to control `models2wux`.

`parameter.names`: A character vector of parameters to be processed according to the NetCDF Climate and Forecast (CF) Metadata Convention (`http://cfconventions.org/`),
e.g. `parameter.names = c("air_temperature", "precipitation_amount")`.

`reference.period`: A character specifying the climate change reference period defined by "from-to" ("YYYY-YYYY"),
e.g. `reference.period = "1961-1990"`.

`scenario.period`: A character specifying the climate change future period defined by "from-to" ("YYYY-YYYY"),
e.g. `scenario.period = "2021-2050"`.

`temporal.aggregation`**:** A named list containing the *n* different levels of statistical aggregation where the single list elements are sequentially named by stat.level.1, stat.level.2, stat.level.3, ... , stat.level.*n*. Each stat.level is again a list containing three elements: period, statistic, and time.series.

period: A named list containing the time period of temporal aggregation. The first aggregation level (stat.level.1) refers to the number of the month in the year. All subsequent aggregation levels refer to the list names of the previous stat.level (i.e. nested structure). For example, in stat.level.1 seasons are defined via
`period=list(DJF=c(12,1,2), MAM=c(3,4,5), JJA=c(6,7,8), SON=c(9,10,11))`.
Winter and summer half years can then be defined in stat.level.2 referring to the list names indicated in stat.level.1:
`period=list(winter=c(SON,DJF), summer=c(MAM,JJA))`

statistic: A string indicating the statistic which is used to aggregate the data. The statistic can be every statistic which is known to R (e.g., mean, sum, quantile).

time.series: `TRUE` or `FALSE` indicating if time series or climatological mean values of the reference and future period and the according climate change signals are calculated.

`subregions`**:** Named list containing information for geographical regions. You can specify the boundaries by passing

- a rectangular region by hand
- a shapefile with subregions of interest
- a NetCDF file containing subregions

All longitude coordinate values are forced to the range from -180 to 180 degrees. In case you want to define a subregion containing the (180,-180)-meridian, you should force the longitude values to the range from 0 to 360 degrees, as it could be the case for the Australasian domain. This can be done with the `wrap.to`-tag (currently defined only for shapefiles).

rectangle: A vector of the form c(lon.west, lon.east, lat.north, lat.south).
e.g. `World = c(-180, 180, 90, -90)`

shapefile: A named list containing the directory to the shapefiles `dirname` and the name of the files `filename` (without file extension). Optional: If no projection file is available, you can set a `projection` tag to
e.g. `projection = "+proj=longlat +ellps=WGS84"`.
In case there are more regions defined in the shapefile, one can give specific names to the `subregionnames` tag e.g `subregionnames = c("South_America", "Central_America")`.
However, sometimes these multiple regions form a set. Then the `category.variable` tag merges the subregions with the same category to a single subregion and `category.label` gives corresponding labels. `category.label` has to be a named vector, with the names being the category values from the `category.variable` and their values being the labels. Omitting the `category.label` vector when using `category.variable`, WUX tries to get the names of `category.variable`. Note that the `subregionnames` tag and the `category.label` should not be used together.
In case you want to wrap your longitudes to the 0-360-degrees grid, flag the named vector `wrap.to = c("my.subregion" = "360")`. Example:
`CORDEX = list(dirname = "/tmp/shapefiles/cordex", filename = "cordex_regions", subregionnames = c("South_America", "Central_America", "North_America", "EU.ENS", "Africa", "West_Asia", "East_Asia","Central_Asia", "Australasia", "Antarctica", "Arctic", "Mediterranean_domain"), wrap.to = c('Australasia' = "360"))`.

NetCDF subregionfile: A named list containing information about the NetCDF file defining the subregion by a constant value (e.g. all pixels flagged by 1 define a subregion). Names of the list have to be:

subreg.file	Name of the NetCDF subregions file.
subreg.dir	Path to the NetCDF subregions file.
grid.file	Name of NetCDF file with longitude and latitude coordinates of the subregions file.
grid.dir	Directory of grid.file.
mask.name	Variable name in subreg.file file defining the region.
mask.value	Value of mask.name defining the region. If more regions are defined, use a vector of values to analyse a set of

area.fraction: Dealing with gridded data, subregions almost never happen do be cut out exactly the way your subregion is specified. If the centroid of a single data pixel lies within the subregion, this datapoint will be taken into analysis, else the datapoint will be considered as lying outside of the subregion and set NA. This is WUX default behavior (area.fraction = FALSE). For very small subregions and/or very course data resolution however, it can happen you get very few data points or even none at all.

However, if you want to take every data pixel which just 'touches' your subregion, use area.fraction. The pixel's centroid doesn't have to be necessarily inside the subregion to be taken into analysis then. With area.fraction = TRUE WUX does a weighted spatial average of all these pixels. The weight is the ratio of the pixel area lying within the subregion and the entire pixel area. So if one quarter of a data point is wihin the subregion (but its centroid for example is not), the data pixel value will be taken into analysis and weighted by 0.25 when averaging spatially. Pixels being covered completely in the subregion have weight 1. area.fraction is useful if you are dealing with very small subregions and/or small data resolution, resulting in just a few pixels.

spatial.weighting: When averaging data over its spatial component, the simple arithmetic mean can result in strongly biased areal estimates. The reason for this is due the geographical projection of the data. The globe has 360 longitudinal degrees and 180 degrees in latitude. The real distance (km) between latitudes remains the same on the entire globe, whereas the distances between longitudes depend on the latitude considered. One degree in longitude near equator represents much more distance (km) than one degree in Norway as the longitudes converge at the poles.

This fact has to be considered especially when dealing with global data (e.g. GCMs). GCM data is usually (within WUX so far 100%) stored on a rectangular lon-lat grid. Therefore the poles seem overproportionaly large in area. Common practice is cosine weighting of latides, resulting in smaller weights near the poles and largest weights at the equator. See http://www.grassaf.org/general-documents/gsr/gsr_10.pdf for more details.

spatial.weighting = TRUE enables cosine weighting of latitudes, whereas omitting or setting FALSE results in unweighted arithmetic areal mean (default). This option is valid only for data on a regular grid.

na.rm: It may happen that time slices of NetCDF data may be missing and the user does not know anything about it. Reason for these artifacts might be short time series (e.g. some models project only until 2035, so an analysis unitl 2050 would be biased) or simply missing values due to corrupt or missing NetCDF files.

If na.rm = TRUE is set in the user input, missing values are filled with NA, but the temporal statistics are calculated using the na.rm = TRUE flag. na.rm = FALSE keeps the NA values and thus leads to NA statistics.

`plot.subregions`**:** A list containing information about diagnostic plotting of grid points within the subregions. `png` plots are generated showing the grid points within a subregion. The size of the drawn circles correspond to the weighting factor of `area.fraction`. The list contains three elements: `save.subregion.plots`, `xlim`, and `ylim`.

save.subregion.plots: A character containing only the output path as the filenames are automatically generated via the model and subregion names. For example `save.subregion.plots = "/tmp/"` will save the plots in the directory `/tmp/`. If `save.subregion.plots` is not specified no plots will be drawn!

xlim: A vector containing the longitudional boundaries of the plots. For example `xlim=c(10,50)` draws the plot from 10 to 50 degrees East. If `xlim` is not specified the boundaries will be automatically generated.

ylim: A vector containing the longitudional boundaries of the plots. For example `xlim=c(10,50)` draws the plot from 10 to 50 degrees North. If `ylim` is not specified the boundaries will be automatically generated.

cex: Factor for pointsize relative to the default.

`save.as.data`**:** A character containing both the output path and filename. For example `save.as.data = "/tmp/cmip3"` will save files in the directory `/tmp/` as `cmip3.csv` (data frame containing model climatologies), `cmip3_diff.csv` (data frame containing the differences of the climatologies, i.e. the climate change signals) and `cmip3.Rdata` (a R binary file which can be loaded into the next R session containing variables `wux.data` and `wux.data.diff` data frames analog to the csv-files).

`climate.models`**:** A character vector containing the names of the models to be processed. The names must be identical to the unique acronyms in the `modelinput` list. Read the next section if you want to add a model in the `modelinput` file.

Configfile "modelinput"

When you want to read in a new climate simulation WUX does not know so far, all you need to do is to specify this model in the `modelinput` list (which should be stored in a file). You don't need to write tedious input routines, WUX does that for you. The `modelinput` list is a named list of climate models and contains meta-information of all currently known climate models. Sometimes models indicate wrong attributes in their NetCDF files needed by `modelinput`. Therfore: KNOW YOUR MODEL YOU WANT TO ADD AND TAKE CARE OF THE META-INFORMATION YOU ARE INDICATING IN `modelinput`.

Each tag consists of a named list with the following mandatory tags (i.e. names):

`institute`**:** Character indicating the institute which is developing the model.

`rcm`**:** Character name indicating the RCM acronym; if you are processing a gcm type "".

`gcm`**:** Character name indicating the GCM acronym.

`emission`**:** Type of emission scenario used for the simulation.

`gridfile.filename`**:** Name of NetCDF grid file containing the lon/lat variables.

`gridfile.path`**:** Directory of the NetCDF grid file.

`file.path.default`: Default directory of the NetCDF data files. If the files are stored not only in one directory, use the `file.path.alt` tag (see below).

`file.path.alt`: If your files are stored not only in one directory, here you can enter a named vector of paths. If files are scattered by parameter, pass the parameter name (CF Metadata convention) as the vector name. If they are split by periods, then pass `historical` and `scenario` as vector names. If files are seperated by both period and parameter, you can use nested named lists instead of vectors.

`file.name`: Character vector of file names of the NetCDF data files. If there are different file names for parameters (which will be mostly the case) and/or file names in scenario- and historical period are of different nature as well, use named or nested lists as in the `file.path.alt` tag. You can set this tag `NA` if this climate model has no files. This makes sense for example for the GKSS model for global radiation, as this ENSEMBLES model does not provide this parameter. Values for this model will be `NA` in the WUX dataframe.

These tags are optional:

`resolution`: Grid resolution character.

`gcm.run`: GCM run. Default is blank "".

`what.timesteps`: Default are daily time steps, type `"monthly"` for monthly data.

`calendar`: Define the NetCDF time:calender attribute by hand. This is necessary if the NetCDF file contains wrong information. You can pass `360_days`, `no_leap` or `julian`.

`time.units`: Define the NetCDF time:units attribute by hand. E.g. `days since 1950-01-06 00:00:00`.

`count.first.time.value`: The time variable in NetCDF files is a vector of time steps relative to the "time:units" attribute with calendar according to the "time:calendar" attribute. However, there are cases where certain climate models are dealing with two calendar types at once! Yes, that's possible... For example: Data claim to have a "360 days" calendar. The "time:units" attribute is set to `days since 1961-01-01 00:00:00` and the time vector looks like `365, 366, ..., 723, 724`. The 365th day since 1961-01-01 is definetely not the 1st January of 1962 concerning the 360-days calendar but is correctly in terms of "julian" dates.
In such a case we would set `count.first.time.value = "julian"` and `calendar` remains 360 days. Other possibilities are `count.first.time.value = "noleap"` (or = `"360days"`). Currently this property is defined for `calendar = "360 days"` only, but can easily be extended to other calendars as well.

`parameters`: A named vector indicating parameter long- and shortname which belong together, e.g. `parameters = c(air_temperature = "tas_dm", precipitation_amount = "pr_24hc")`. This is important if the NetCDF internal variable name deviates from the WUX default parameter shortname:

`tas`	for `air_temperature`
`pr`	for `precipitation_amount`
`hurs`	for `relative_humidity`
`rsds`	for `global_radiation`

`wss`	for `wind_speed`
`ua`	for `eastward_wind`
`va`	for `northward_wind`
`psl`	for `air_pressure_at_sea_level`
`hus`	for `specific_humidity`
`hfss`	for `surface_upward_sensible_heat_flux`
`tasmin`	for `air_temperature_minimum`
`tasmax`	for `air_temperature_maximum`
`ts`	for `surface_temperature`

Note

This is an awesome tool (rfp).

Author(s)

Thomas Mendlik <`thomas.mendlik@uni-graz.at`> and Georg Heinrich <`g.heinrich@uni-graz.at`>

See Also

`modelinput_test`, `userinput_CMIP5_changesignal`, `cmip5_2050`, `cmip5_2100`, `ensembles`, `ensembles_gcms`

Examples

```
## This example shows a typical workflow for models2wux, the workhorse of
## the wux package. Going through this example step-by-step, you will
## retrieve NetCDF files of two CMIP5 simulations and aggregate them to
## an R data.frame for further analysis.

## I) Load wux functions and example datasets...
library("wux")

## II) You need to obtain the climate simulations first. You can get
## started with downloading some example CMIP5 NetCDF files from the
## ESGF visiting for example http://pcmdi9.llnl.gov or using the
## CMIP5fromESGF function. Here, we dowload two simulations "NorESM1-M" and
## "CanESM2" into your home directory "~/tmp/CMIP5/" which will be
## created automatically. You will need a valid account at any ESGF
## node for this function to run. See ?CMIP5fromESGF for further help.
## Not run: CMIP5fromESGF(save.to = "~/tmp/CMIP5/",
                models = c("NorESM1-M", "CanESM2"),
                variables = c("tas"),
                experiments= c("historical", "rcp85"))

## End(Not run)

## III) Specify those downloaded data for models2wux. models2wux needs
## to know where the data is stored on your HDD and needs to have access
## to certain metadata of the climate simulator, which you have to
## provide as well. This information is stored in a list, which should
## be saved as ONE file somewhere on your computer. We call this
## information "modelinput". You should share this
## file with you collegues using the same IT infrastructure to share
## synergies. You can create such a file based on the data downloade
## by "CMIP5fromESGF":
## Not run: CMIP5toModelinput(filedir = "~/tmp/CMIP5",
                  save.to = "~/modelinput.R")

## End(Not run)
## This file then would look this:
data(modelinput_test)

## It specifies temperature and precipitation files for the two
## simulations "NorESM1-M" and "CanESM2" (RCP8.5), stored in
## "~/tmp/CMIP5/".
str(modelinput_test)

## IV) Next, you need to specify which simulations you want to read in
## with models2wux, what kind of statistics to calculate, what subregion
## to analyze, what time periods and seasons to define, and so on. This
## is done with a user input file, which cntains a list with all the
## necessary information. You typically use different userinput files
## for different analysis, whereas your modelinput should remain in ONE
## file which will be updated each time you obtain a new climate
## simulation. One example user input file, which reads in both
## simulations specified above for the Alpine domain and returns their
## projected climate change signal, could look like follows:
data(userinput_CMIP5_changesignal)
str(userinput_CMIP5_changesignal)

## alternatively following userinput returns a timeseries of both
## models, which only differs by the "time.series" tag and differently
## specified periods:
data(userinput_CMIP5_timeseries)
str(userinput_CMIP5_timeseries)

## V) At last you can run models2wux to obtain a data.frame of the
## specified climatic change features defined above:
## Not run: climchange.df <- models2wux(userinput = userinput_CMIP5_changesignal,
                            modelinput = modelinput_test)
## End(Not run)
## A better practice is to safe both input files containing a named
## list each somewhere on your disk and pass the files directly to the
## models2wux function. If you  had stored the two files in your home
## directory as e.g. "~/userinput.R" and "~/modelinput.R" you can call:
## Not run: climchange.df <- models2wux(userinput = "~/userinput.R",
                            modelinput = "~/modelinput.R")
## End(Not run)
## if you downloaded the data correctly, you should obtain a data.frame:
## Not run:
  climchange.df

## End(Not run)
## which should be identical to this example data.frame:
data(CMIP5_example_changesignal)
CMIP5_example_changesignal

## Instead of calculating the climate change signals, you can also
## generate time series of the two models aggregated over the Alpine
## domain, using a different user input file:
## Not run: climchange.df <- models2wux(userinput = userinput_CMIP5_timeseries,
                            modelinput = modelinput_test)
## End(Not run)


## VI) Finally you can make all kind of analysis you are interested in,
## using either functions from wux or from any other R funtionality
summary(CMIP5_example_changesignal, parms = "delta.air_temperature")

## or plot timeseries as
require(lattice)
data(CMIP5_example_timeseries)
## Not run: xyplot(air_temperature ~ year|season,
       groups = acronym,
       data = CMIP5_example_timeseries,
       type = c("l", "g"),
       main = "NorESM1-M and CanESM2 simulations over Alpine Region\nRCP 8.5 forcing")
## End(Not run)
```

`plot.wux.aov` *Barplots of the ANOVA results*

Description

Barplots of the `aovWux` results displaying the relative or absolute contribution of the individual factors to the overall variance.

Usage

```
## S3 method for class 'wux.aov'
plot(x, ss.relative = TRUE, subreg.subset =
NULL, cex.names = 1.2, cex.lab = 1.2, legend.text = NULL, sd.text =
TRUE, sd.unit = "", ylim = NULL, ylab = NULL, main = NULL,
out.file.directory = NULL, out.file.name = NULL, copyright = FALSE, ...)
```

Arguments

`x`	Object of class `wux.aov` obtained from the ANOVA `aovWux`.
`ss.relative`	Boolean. Indicating if the relative contribution of the factors to the overall variance should be calculated. Default is `TRUE`.
`subreg.subset`	Vector of subregions to be plotted (e.g. `c("EU.ENS", "GAR")`).

`cex.names`	Expansion factor for numeric axis labels in bxp. Default is `1.2`.
`cex.lab`	Expansion factor for axis names (bar labels) in bxp. Default is `1.2`.
`legend.text`	String vector of the factors (e.g. `c("GCM", "RCM", "RES")`).
`sd.text`	Boolean. Indicating if the overall standard deviation should be displayed. Default is `TRUE`.
`sd.unit`	Character string of the standard deviation unit with default `""` (e.g. "K").
`ylim`	Range vector for the y-axis.
`ylab`	Label for y-axis.
`main`	Main title.
`out.file.directory`	String of the directory where the plots are exported (e.g. `"/tmp/plots/"`). If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displayed on screen.
`out.file.name`	Prefix of the file names of the plots. Files will be stored as `out.file.name_subreg_season.eps`, where `subreg` is one realization of the `subreg.subset` argument and season is one realization of `season.subset`. For example: `out.file.name = "Barplot"` will store to the files to `Barplot_EUROPE_DJF.eps` and `Barplot_EUROPE_JJA.eps`. If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displyed on screen.
`copyright`	Boolean. If a copyright message should be plotted. Default is FALSE.
`...`	Further optional arguments passed to `barplot`.

Author(s)

Georg Heinrich <g.heinrich@uni-graz.at>

Examples

```
## load WUX and read WUX test data
require(wux)
data(ensembles)

wuxtest.df <- subset(ensembles, subreg == "GAR")

## unique model acronyms are required for reconstruction
wuxtest.df$acronym <- factor(paste(wuxtest.df$institute, "_",
wuxtest.df$rcm, sep=""))

## reconstruction of the  missing data
reconstructLES.df <- reconstruct(wuxtest.df, factor1.name =
"acronym", factor2.name = "gcm", data.name =
"perc.delta.precipitation_amount")

## calculate ANOVA
anova.list <- aovWux(perc.delta.precipitation_amount ~ acronym +
gcm, reconstructLES.df)
```

```
## barplot of ANOVA results
## Not run: plot(anova.list, ss.relative = TRUE, las = 1,
sd.unit = "%", legend.text = c("RCM", "GCM", "RES"), mgp = c(2.5,1,0),
main = "ANOVA Barplot", ylim = c(0,110))

## End(Not run)
```

---

plot.wux.df *X - Y Scatterplot of climate change signals*

Description

plot.wux.df plots one or more scatterplots containing climate change signals of selected meteorological parameters.

This plotting routine extracts all the information from the input data frame which has to be 'WUX-style' (see `models2wux`).

Usage

```
## S3 method for class 'wux.df'
plot(x,
                var1.name = "delta.air_temperature",
                var2.name = "perc.delta.precipitation_amount",
                subreg.subset = NULL,
                season.subset = NULL, boxplots = TRUE,
                label.only.these.models = NULL, highlight.models = NULL,
                 no.text = FALSE,
                vert.box.col = "cyan", horiz.box.col = "coral",
                zero.line.col = "gray80", median.line.col = "black",
                draw.legend = TRUE, draw.seperate.legend = FALSE,
                draw.median.lines = TRUE, use.rainbow.colors = TRUE,
                xlim = NULL, ylim = NULL, xlab = NULL, ylab = NULL,
                main = NULL, out.file.directory = NULL,
                out.file.name = NULL, copyright = FALSE, ...)
```

Arguments

`x`	WUX data.frame (class `wux.df`) obtained from `models2wux`'.
`var1.name`	Character string of 1st parameter in WUX dataset. Default is temperature change.
`var2.name`	Character string of 2nd parameter in WUX dataset. Default is precipitation change.
`subreg.subset`	Vector of subregions to be plotted (e.g. `c("EU.ENS", "GAR")`).
`season.subset`	Vector of seasons to be plotted (e.g. `c("MAM", "DJF")`).
`boxplots`	Boolean. Indicating if marginal boxplots for the two input parameters should be plotted. Default is `TRUE`.

`label.only.these.models`	Character vector of modelnames (acronyms) to be labeled in the scatterplot.
`highlight.models`	Character vector of modelnames (acronyms) to be highlighted in the scatterplot.
`no.text`	Boolean. Indicating if no models should be labeled. Default is `FALSE`.
`vert.box.col`	Color character for vertical boxplot. Default is `coral`.
`horiz.box.col`	Color character for horizontal boxplot. Default is `cyan`.
`zero.line.col`	Color character for the zero lines. Default is `gray80`.
`median.line.col`	Color character for the median lines. Default is `black`.
`use.rainbow.colors`	Boolean. Use `rainbow()` color palette if `TRUE`, otherwise a custom color palette with 17 colors is used. Default is `TRUE`.
`xlim`	Range vector for 1st parameter (x-axis).
`ylim`	Range vector for 2nd parameter (y-axis).
`xlab`	Label of 1st parameter (x-axis).
`ylab`	Label of 2nd parameter (y-axis).
`draw.legend`	Boolean. Indicating if legend with GCMs should be plotted. Default is `TRUE`.
`draw.seperate.legend`	Boolean. Should legend with GCMs be plotted on a seperate screen? Default is `FALSE`. Draws legend even if `draw.legend` is set `FALSE`.
`draw.median.lines`	Draw median lines for both parameters. Default is `TRUE`.
`main`	Main title.
`out.file.directory`	Directory where the plots shall be exported (e.g. "/tmp/plots/"). If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displyed on screen.
`out.file.name`	Prefix of the file names of the plots. Files will be stored as `out.file.name_subreg_season.eps`, where `subreg` is one realization of the `subreg.subset` argument and `season` is one realization of `season.subset`. For example: `out.file.name = "scatterplot"` will store to the files `scatterplot_EUROPE_DJF.eps` and `scatterplot_EUROPE_JJA.eps`. If neither `out.file.name` nor `out.file.directory` are passed, the plot will be displyed on screen.
`copyright`	Boolean. If a copyright message should be plotted. Default is FALSE.
`...`	Further optional arguments to be passed to `plot`, such as graphical parameters (see `par`).

Author(s)

Thomas Mendlik <`thomas.mendlik@uni-graz.at`> and Georg Heinrich <`g.heinrich@uni-graz.at`>

Examples

```
require(wux)

### ENSEMBLES RCM analysis
data(ensembles)

## Not run: plot(ensembles, "perc.delta.precipitation_amount",
  "delta.air_temperature", boxplots = TRUE, xlim = c(-40,40),
  ylim = c(0, 4), label.only.these.models = c("ICTP-REGCM3", "MPI-M-REMO"),
  xlab = "Precipitation Amount [%]", ylab = "2-m Air Temperature [K]",
  main = "Scatterplot", subreg.subset = c("GAR"))

## End(Not run)

### now see where ENSMEBLES GCMs lie within CMIP3 ensemble
data(ensembles_gcms) # GCMs for forcing of ENSEMBLES RCMs
data(cmip3_2050)     # GCMs of CMIP3 ensemble

ensembles.gcm.names <- levels(ensembles_gcms$acronym) #8 GCM names

cmip3_2050.sub <- subset(cmip3_2050, subreg %in% c("World", "EU.ENS")
                       & em.scn == "A1B")
cmip3_2050.sub <- droplevels(cmip3_2050.sub)
ensembles_gcms.sub <- subset(ensembles_gcms, !acronym %in%
                                c("mpi_echam5-r3", "bccr_bcm2_0-r1",
                                  "ipsl_cm4-r2"))
ensembles_gcms.sub <- droplevels(ensembles_gcms.sub)
## combine cmip3 and ENSEMBLES GCMs in one data.frame
gcms.combined <- rbind(ensembles_gcms.sub, cmip3_2050.sub)

## Scatterplot
prec.range <- range(gcms.combined$perc.delta.precipitation_amount) + c(-1, 1)
tas.range <- range(gcms.combined$delta.air_temperature)
## Not run: plot(gcms.combined,
               "perc.delta.precipitation_amount", "delta.air_temperature",
               subreg.subset = "EU.ENS", draw.median.lines = FALSE,
               label.only.these.models = ensembles.gcm.names,
               xlim = prec.range,
               ylim = tas.range,
               main = "GCMs from ENSEMBLES project within CMIP3 SRESA1B ensemble",
               draw.seperate.legend = TRUE)
## End(Not run)
```

`plotAnnualCycle` *Plots the annual cycle*

Description

plotAnnualCycle plots the monthly or seasonal annual cycle of indicated models and the box-whisker plots of the underlying distribution.

This plotting routine extracts all the information from the input data frame which has to be 'WUX-style' (see `models2wux`).

Usage

```
plotAnnualCycle(datain.df, var.name = NULL, subreg.subset = NULL,
season.subset = NULL, plot.quantiles = NULL, quantile.method = 7,
mark.df = NULL, plot.legend = FALSE, cex.names = 1.2, cex.lab = 1.2,
ylab = NULL, main = NULL, out.file.directory = NULL, out.file.name =
NULL, copyright = FALSE, ...)
```

Arguments

`datain.df`	WUX data frame obtained from `models2wux`.
`var.name`	Character string of parameter in WUX dataset.
`subreg.subset`	Vector of subregions to be plotted (e.g. `c("EU.ENS", "GAR")`).
`season.subset`	Vector of seasons to be plotted (e.g. `c("MAM", "DJF")`).
`plot.quantiles`	5 element vector indictaing the quantiles to be plotted (e.g. `c(0.02,0.25,0.5,0.75,0.98)`).
`quantile.method`	An integer between 1 and 9 selecting one of the nine quantile types in `quantiles` with default 7.
`mark.df`	Subset of WUX data frame indicating the models to be marked.
`plot.legend`	Boolean. Indicating if a plot legend indicating the models of `mark.df` and sample size should be plotted. Default is `FALSE`.
`cex.names`	Expansion factor for numeric axis labels in bxp. Default is `1.2`.
`cex.lab`	Expansion factor for axis names (bar labels) in bxp. Default is `1.2`.
`ylab`	Label for y-axis.
`main`	Main title.
`out.file.directory`	String of the directory where the plots are exported (e.g. `"/tmp/plots/"`).
`out.file.name`	Prefix of the file names of the plots. Files will be stored as `out.file.name_subreg_season.eps`, where `subreg` is one realization of the `subreg.subset` argument and season is one realization of `season.subset`. For example: `out.file.name = "AnnualCycle"` will store to the files to `AnnualCycle_EUROPE_DJF.eps` and `AnnualCycle_EUROPE_JJA.eps`.
`copyright`	Boolean. If a copyright message should be plotted. Default is FALSE.
`...`	Further optional arguments passed to bxp.

Author(s)

Georg Heinrich <`g.heinrich@uni-graz.at`>

Examples

```
## load WUX and read WUX test data
require(wux)
data(ensembles)

wuxtest.df <- subset(ensembles, subreg == "GAR")

## set data frame for model marks
mark.df <- subset(wuxtest.df, acronym %in% c("ICTP-REGCM3", "MPI-M-REMO"))
mark.df2 <- gdata::drop.levels(mark.df)

## Not run: plotAnnualCycle(wuxtest.df, "perc.delta.precipitation_amount", mark.df =
mark.df, plot.legend = TRUE, boxfill = "light yellow", notch =
FALSE,  boxwex = 0.5, ylim = c(-60,60), plot.quantiles =
c(0.02,0.25,0.5,0.75,0.98), boxcol = "red", ylab = "Precipitation
Amount [%]", main = "Annual cycle ", las = 1, copyright = TRUE)

## End(Not run)
```

`read.wux.table` *Reads in wux data.frame from harddisk*

Description

Reads in wux csv file obtained from `models2wux` from harddisk and creates a `wux.df` object (data frame) from it.

Usage

```
read.wux.table(file, ...)
```

Arguments

`file`	the name of the file which the data are to be read from.
`...`	Further arguments to be passed to `read.table`.

Value

A `wux.df` data.frame object.

Author(s)

Thomas Mendlik `<thomas.mendlik@uni-graz.at>`

Examples

```
## read WUX test data
## Not run: wux.data.frame <- read.wux.table("~/dir/to/data/ensembles_diff.csv")
```

reconstruct — *Missing value reconstruction based on ANOVA*

Description

Performs a simple missing value reconstruction based on an ANOVA with two factors using different methods.

Usage

```
reconstruct(x, factor1.name, factor2.name,
 data.name, method = "LES", iterations.num = 100)
```

Arguments

`x`	WUX data frame of class `"wux.df"` obtained from `models2wux`
`factor1.name`	Name of the 1st factor.
`factor2.name`	Name of the 2nd factor.
`data.name`	Name of the variable to be reconstructed.
`method`	One of the currently implemented methods: "LES", "Iterative" or "IterativeCC". See details section.
`iterations.num`	Number of iterations to be performed. Used only for `method` "Iterative" or "IterativeCC".

Details

Tools for filling missing values of an unbalanced climate model simulation matrix (e.g. missing RCM-GCM combinations of ENSEMBLES) in order to avoid biased ensemble estimates. Following methods are currently implemented:

`method = "LES"` **(default):** Performs a simple missing value reconstruction with two factors based on solving the linear equation system (LES) of the ANOVA. The algorithm follows Déqué et al. (2007) but the reconstruction is based on solving the linear equation system (LES) of the ANOVA instead of reconstructing iteratively. The main advantages of this method are that it is much faster and can be more easily extended to more factors than the original one. However, keep in mind that the results slightly differ from the iterative procedure proposed by Déqué et al. (2007). The reconstruction algorithm is based on unique factor combinations (i.e. one element per combination of `factor1.name` and `factor2.name`).

`method = "Iterative"`**:** The data reconstruction follows the iterative procedure based on the ANOVA proposed by Déqué et al. (2007). The reconstruction algorithm is based on unique factor combinations (i.e. one element per combination of `factor1.name` and `factor2.name`).

`method = "IterativeCC"`**:** Performs a leave one out cross calculation (CC) of the ANOVA based missing value reconstruction with two factors based on and following the iterative procedure of `method = "Iterative"`.

Value

Returns a WUX data frame containing the reconstructed data of class `c("rwux.df", "wux.df", "data.frame")`.

Author(s)

Georg Heinrich `<g.heinrich@uni-graz.at>` and Thomas Mendlik `<thomas.mendlik@uni-graz.at>`

References

Déqué M, Rowell DP, Lüthi D, Giorgi F, Christensen JH, Rockel B, Jacob D, Kjellström E, de Castro M, van den Hurk B. 2007. An intercomparison of regional climate simulations for Europe: Assessing uncertainties in model projections. Climatic Change 81: 53–70. DOI:10.1007/s10584-006-9228-x.

Examples

```
## load WUX and read WUX test data
require(wux)
data(ensembles)

wuxtest.df <- subset(ensembles, subreg == "GAR")

## unique model acronyms are required for reconstruction
wuxtest.df$acronym <- factor(paste(wuxtest.df$institute, "_", wuxtest.df$rcm, sep=""))

## reconstruction of the  missing data
reconstructLES.df <- reconstruct(wuxtest.df,
  factor1.name = "acronym", factor2.name = "gcm", data.name =
  "perc.delta.precipitation_amount", method = "LES")

## reconstruction of the  missing data using iterative apporach from
## Deque et al (2007)
reconstructIterative.df <- reconstruct(wuxtest.df,
  factor1.name = "acronym", factor2.name = "gcm", data.name =
  "perc.delta.precipitation_amount", method = "Iterative",
  iterations.num = 10)

## reconstruction of the  missing data using iterative apporach with
## cross-calculation. This can take some time.
## Not run: reconstructIterative.df <- reconstruct(wuxtest.df,
  factor1.name = "acronym", factor2.name = "gcm", data.name =
  "perc.delta.precipitation_amount", method = "IterativeCC",
  iterations.num = 10)
## End(Not run)
```

summary — *Summary of a WUX data.frame*

Description

Prints a screen summary of a WUX `data.frame` (class `wux.df`).

Usage

```
## S3 method for class 'wux.df'
summary(object, parms = c("perc.delta.precipitation_amount",
"delta.air_temperature"), average.over.gcm.runs = FALSE, ...)
```

Arguments

`object`	WUX `data.frame` (`wux.df` object) obtained from `models2wux`
`parms`	String vector specifying the parameters to be evaluated. Default is `perc.delta.precipitation_amount` (percentage of precipitation change) and `delta.air_temperature` (air temperature change in K).
`average.over.gcm.runs`	Boolean. Should the runs of the same GCM be averaged out? This is recommended, as same GCMs tend to behave very similarly when run with different initial conditions and would thus lead to biased statistics when regarding as independent. Only available for GCM analysis.
`...`	Further optional arguments. Not active now.

Details

`summary.wux.df` gives an overview of model frequenzy and calculates statistics for each meteorological parameter within each season in each subregion for all emission scenarios.

`print.summaryWuxdf` prints the result to the screen.

Value

Returns a `summaryWuxdf` object, which is a list, but will be printed in a special way. The list has two elements, namely `overall.stats` and `parms.stats`. Both are lists again. `overall.stats` stores all categorical statistics (climate model counts, emmission scenarios, rcm-gcm crosstables, ...). `parms.stats` is a list with statistics of continuous climate change signals (mean, standard deviation, coeficent of variation and quantiles) split by season, emission scenario, meteorological parameters and subregions.

Author(s)

Thomas Mendlik <`thomas.mendlik@uni-graz.at`>

Examples

```
## ENSEMBLES data summary
data(ensembles)
summary(ensembles)

## CMIP3 data summary
data(cmip3_2100)
summary(cmip3_2100, average.over.gcm.runs = TRUE) # Average GCMs with different
                                                       # initial conditions

## structure of summaryWuxdf object
data(ensembles_gcms)
ensembles.gcms.summary <- summary(ensembles_gcms)
ensembles.gcms.summary  # summary of 8 GCMs
str(ensembles.gcms.summary)
```

`userinput_CMIP5_changesignal`

Example userinput for models2wux

Description

This example `userinput_CMIP5_changesignal` can be used to test the `models2wux` functionality. A userinput is a list of configurations used to read and process climate model data. In general, you should store it as an own file somewhere on your system. It calculates the climate change signal of 1971-2000 and 2071-2100 for temperature over the Alpine region of 2 CMIP5 models "NorESM1-M" and "CanESM2". It aggregates the monthly NetCDF model output to boreal seasons, winter, spirng, summer and autumn. It also stores the output as a csv-file in your "/tmp" directory.

Usage

```
data(userinput_CMIP5_changesignal)
```

Details

See "Configfile userinput" section in `models2wux`.

See Also

`models2wux`, `userinput_CMIP5_timeseries`

Examples

```
## thats what userinput_CMIP5_changesignal looks like:
## it contains a single list named user.input
## describing 2 CMIP5 models in the alpine region
data("userinput_CMIP5_changesignal")
is.list(userinput_CMIP5_changesignal)
str(userinput_CMIP5_changesignal)
```

```
data(modelinput_test)

## reading in these data and process them:
## Not run: wux.test <- models2wux(userinput_CMIP5_changesignal,
                                modelinput = model.input)
## End(Not run)
## if you had a file "/tmp/userinput_CMIP5_changesignal.R" which contains a
## list 'user.input with the same content as 'userinput_CMIP5_changesignal'
## you could read the data also like this:
## Not run: wux.test <- models2wux("/tmp/userinput_CMIP5_changesignal.R",
                        modelinput = model.input)
## End(Not run)

## the result is what the data.set would look like, if you ran the code
## above:
data(CMIP5_example_changesignal)
wux.test <- CMIP5_example_changesignal
wux.test

## example summary though the statistics not make much sense with 2 models
summary(wux.test, parms = "delta.air_temperature")
```

---

userinput_CMIP5_timeseries

Example userinput for models2wux

Description

This example `userinput_CMIP5_changesignal` can be used to test the `models2wux` functionality. A userinput is a list of configurations used to read and process climate model data. In general, you should store it as an own file somewhere on your system. It calculates a time series of the historical run 1971-2005 and RCP 8.5 2006-2100 for temperature over the Alpine region of 2 CMIP5 models "MIROC5" and "CanESM2". It aggregates the monthly NetCDF model output to boreal seasons, winter, spirng, summer and autumn. It also stores the output as a csv-file in your "/tmp" directory.

Usage

```
data(userinput_CMIP5_timeseries)
```

Details

See "Configfile userinput" section in `models2wux`.

See Also

`models2wux`, `userinput_CMIP5_changesignal`

Examples

```
## thats what userinput_CMIP5_timeseries looks like:
## it contains a single list named user.input
## describing 2 CMIP5 models in the alpine region
data("userinput_CMIP5_timeseries")
is.list(userinput_CMIP5_timeseries)
str(userinput_CMIP5_timeseries)

data(modelinput_test)

## reading in these data and process them:
## Not run: wux.test <- models2wux(userinput_CMIP5_timeseries,
                                modelinput = model.input)
## End(Not run)
## if you had a file "/tmp/userinput_CMIP5_timeseries.R" which contains a
## list 'user.input with the same content as 'userinput_CMIP5_timeseries'
## you could read the data also like this:
## Not run: wux.test <- models2wux("/tmp/userinput_CMIP5_timeseries.R",
                        modelinput = model.input)
## End(Not run)

## the result is what the data.set would look like, if you ran the code
## above:
data(CMIP5_example_timeseries)
wux.test <- CMIP5_example_timeseries

## Not run: require(lattice)
xyplot(air_temperature ~ year|season,
       groups=acronym,
       data = wux.test,
       type = c("l", "g"),
       main = "Temperature trends for Alpine Region" )

## End(Not run)
```

Index