Description

We provide tools for handling time series of satellite images as well as some statistical methods for spatio-temporal analysis.

Tools for handling time series of satellite images

transfer_bin_raster transfers data from images originally recorded in a binary format to images in any of the formats allowed by the raster package. Similarly, transfer_raster_RData extracts the entries (numbers) of images originally recorded as a tiff file, virtually storages them in an array object and, finally, this array is saved in an RData file. split_replace allows us to split Raster* objects, which can be arguably large, into smaller chunks. These chunks can be saved in any of the formats allowed by writeRaster. Often, satellite images come with missing values (or fill values assigned by other computer programs), split_replace allows to replace these values by values of users’ convenience; see also reclassify.

Methods for analyzing time series of satellite images

haRmonics allows us to fit classical harmonic regression to numeric vectors; the method hants is based on Roerink et al. (2000) whereas the method haRm is based on Jakubauskas et al. (2001).

Author(s)

Tecuapetla-Gomez, I. <itecuapetla@conabio.gob.mx>

References


The Matlab implementation of HANTS can be found here.
harmonics

Harmonic analysis for time series

Description

Fits harmonic regression (harmR) model, that is, computes amplitudes and phase angles in the typical harmonic regression framework. Based on these estimates a harmonic regression function is fitted. Also fits hants, a popular iterative algorithm that computes amplitudes and phase angles in the harmonic regression framework. As part of the iterative algorithm, observations are being excluded from the design matrix of the regression model if the distance between them and the fitted curve exceeds the value of the parameter fitErrorTol. hants is based on implementations with the same name written in Fortran and Matlab computer languages.

Usage

```r
harmonics(y, method = c("harmR", "hants"), ts = 1:length(y),
        lenBasePeriod = length(y), numFreq, HiLo = c("Hi", "Lo"), low, high,
        fitErrorTol, degreeOverDeter, delta)
```

Arguments

- **y**: numeric vector containing time series on which harmonic regression will be fitted. Missing values are not allowed.
- **method**: character specifying algorithm to apply: harmR (default) or hants.
- **ts**: numeric vector of length(y) with the sampling points for y. Default is ts[i] = i, i = 1, ..., length(y).
- **lenBasePeriod**: numeric giving the length of the base period, reported in samples, e.g. days, dekads, months, years, etc.
- **numFreq**: numeric indicating the total number of frequencies to be used in harmonic regression.
- **HiLo**: character indicating whether high or low outliers must be rejected when method=hants.
- **low**: numeric giving minimum valid value of fitted harmonic regression function when method=hants.
- **high**: numeric giving maximum valid value of fitted harmonic regression function when method=hants.
- **fitErrorTol**: numeric giving maximum allowed distance between observations and fitted curve; if difference between a given observation and its fitted value exceeds fitErrorTol then this observation will not be included in the fitting procedure in the next iteration of the algorithm.
- **degreeOverDeter**: numeric; iteration stops when number of observations equals number of observations for curve fitting plus degreeOverDeter; the latter in turns is by definition length(y) minus min(2 * numFreq + 1, length(y)).
- **delta**: numeric (positive) giving a (small) regularization parameter to prevent non-invertible hat matrix (see details), probably caused by high amplitudes.
Details

Method haRmonics does not allow missing values and utilizes parameters \(y, \text{lanBasePeriod, numFreq}\) and \(\delta\) only.

Method hants utilizes all the parameters presented above. This method does not allow missing values. Missing values in \(y\) must be substituted by values considerably out of observations range.

Value

A list containing:

- \(a\).coef: a numeric vector with estimates of cosine coefficients
- \(b\).coef: a numeric vector with estimates of sine coefficients
- amplitude: a numeric vector with amplitude estimates.
- phase: a numeric vector with phase estimates.
- fitted: a numeric vector with fitted values via harmonic regression.

References


The Matlab implementation of HANTS can be found [here](#).

Examples

```r
y <- c(5, 2, 5, 10, 12, 18, 20, 23, 27, 30, 40, 60, 66, 70, 90, 120, 160, 190, 105, 210, 104, 200, 90, 170, 50, 120, 80, 60, 50, 40, 30, 28, 24, 20, 15, 10)
# ------------------------------------------------------------------
fit_harmR <- haRmonics(y = y, numFreq = 3, delta = 0.1)
fitLow_hants <- haRmonics(y = y, method = "hants", numFreq = 3, HiLo = "Lo",
                          low = 0, high = 255, fitErrorTol = 5, degreeOverDeter = 1,
                          delta = 0.1)
fitHigh_hants <- haRmonics(y = y, method = "hants", numFreq = 3, HiLo = "Hi",
                          low = 0, high = 255, fitErrorTol = 5, degreeOverDeter = 1,
                          delta = 0.1)
plot(y, pch = 16, main = "haRmonics fitting")
lines(fit_harmR$fitted, lty = 4, col = "green")
lines(fitLow_hants$fitted, lty = 4, col = "red")
lines(fitHigh_hants$fitted, lty = 2, col = "blue")
# ------------------------------------------------------------------

# Substituting missing value by a number outside observations range
# ------------------------------------------------------------------
y1 <- y
y1[20] <- -10
fitLow_hants_missing <- haRmonics(y = y1, method = "hants", numFreq = 3, HiLo = "Lo",
                                 low = 0, high = 255, fitErrorTol = 5, degreeOverDeter = 1,
                                 delta = 0.1)
```
delta = 0.1)
fitHigh_hants_missing <- haRmonics(y = y1, method = "hants", numFreq = 3, HiLo = "Hi",
        low = 0, high = 255, fitErrorTol = 5, degreeOverDeter = 1,
        delta = 0.1)
fit_harmR_missing <- haRmonics(y = y1, numFreq = 3, delta = 0.1)

plot(y1, pch = 16, main = "haRmonics fitting (missing values)", ylim = c(-1,210))
lines(fitLow_hants_missing$fitted, lty = 4, col = "red")
lines(fitHigh_hants_missing$fitted, lty = 2, col = "blue")
lines(fit_harmR_missing$fitted, lty = 4, col = "green")
maxLagMissVal

Get maximum lag of missing values

Description
This function computes the maximum amount of consecutive missing values in a vector. This quantity is also known as maximum lag, run, or record, and can be used as a rough estimate of the quality of a dataset.

Usage
maxLagMissVal(x, type = c("NA", "numeric"), value)

Arguments
- **x**: numeric vector.
- **type**: character specifying the type of missing value to consider. Default is type = "NA"; when type == "numeric", value must be provided.
- **value**: numeric giving a figure to be used to fill missing values; often as part of a pre-processing, missing values in a dataset (vector, time series, etc.) are fill in with pre-established values.

Value
A list containing:
- **maxLag**: numeric giving the maximum lag of missing values in x
- **x**: numeric vector with the original data
- **value**: a numeric when type == numeric, NA otherwise

See Also
- rle

Examples
```r
v <- c(NA, 0.12, 0.58, 0.75, NA, NA, NA, NA, 0.46, 0.97, 0.39, NA, 0.13, 0.46, 0.95, 0.30, 0.98, 0.23, 0.98, 0.68, NA, NA, NA, NA, 0.11, 0.10, 0.79, 0.46, 0.27, 0.44, 0.93, 0.20, 0.44, 0.66, 0.11, 0.88)
maxLagMissVal(x=v, type="NA")

w <- c(23, 3, 14, 3, 8, 3, 3, 3, 3, 3, 3, 3, 3, 10, 14, 15, 3, 10, 3, 3, 6)
maxLagMissVal(x = w, type = "numeric", value = 3)
```
split_replace

Splits a Raster* object into smaller chunks and allows to replace cell values

Description

This function will split a Raster* object into smaller chunks. The size of these chunks (number of cells) is controlled by partPerSide, h or v. Additionally, it allows to replace cell values (valToReplace) within Raster* object by another value of user’s choice (replacedBy). When save = TRUE, the resulting cellsToProcess Raster* objects are saved in directory outputPath.

Usage

split_replace(raster, partPerSide, h, v, outputPath, name, save = TRUE, replace = FALSE, valToReplace, replacedBy, dataType, format = "GTiff", parallelProcessing = FALSE, numCores = 20, cellsToProcess, ...)

Arguments

raster Raster* object.
partPerSide integer indicating number of cells in which raster will be split in each direction (horizontally and vertically). Use when nrow(raster) and ncol(raster) are multiples of partPerSide.
h integer indicating number of horizontal cells in which raster will be split.
v integer indicating number of vertical cells in which raster will be split.
outputPath character with full path name where the resulting Raster* objects will be saved.
name character with the name to assign to final products.
save logical, should the output be saved, default is TRUE.
replace logical, default FALSE, when TRUE, valToReplace and replacedBy must be specified.
valToReplace indicates a value to be replaced across raster cells.
replacedBy indicates the value by which valToReplace is replaced.
dataType character, output data type. See dataType.
format character, output file type, default "GTiff". See writeFormats.
parallelProcessing logical, default FALSE, when TRUE raster splitting is done in parallel. See details.
numCores numeric indicating the number of cores used in parallel processing.
cellsToProcess numeric vector indicating which smaller cells should be processed/saved. See details.
... additional arguments used by writeRaster.
Details

Before processing any of the cellsToProcess the temporary raster directory is re-directed. Basically, prior to process the i-th cell, at outputPath a new subdirectory is created, which, in turn, is erased automatically once the i-th cell has been processed. As a result of several tests we found that this measure avoids memory overflow.

When partPerSide is used, cellsToProcess = 1:(partPerSide^2). When h and v are used, cellsToProcess = 1:(ncells(raster)/(h*v)). Since the code assumes that nrow(raster) and ncol(raster) are multiples of partPerSide or h and v, respectively, the user must be careful when selecting these parameters.

For parallelProcessing the backend doParallel is employed.

Value

At outputPath the user will find length(cellsToProcess) Raster* files

See Also

writeRaster, aggregate, rasterOptions

---

transfer_bin_raster  Transfer values from a binary image file to a raster file

Description

Get the values of a binary file (in integer format) and transfer them to a raster file. All formats considered in writeRaster are allowed.

Usage

```
transfer_bin_raster(inputPath, outputPath, master, what = integer(),
  signed = TRUE, endian = "little", size = 2, format = "GTiff",
  dataType = "INT2S", overwrite = TRUE)
```

Arguments

- `inputPath` character with full path name of input file(s).
- `outputPath` character with full path name (where the raster files will be saved).
- `master` character with full path name of a raster file; extent and projection of this file are applied to this function output.
- `what` See `readBin`. Default integer().
- `signed` See `readBin`. Default TRUE.
- `endian` See `readBin`. Default "little".
- `size` integer, number of bytes per element in the byte stream, default 2. See `readBin`.
- `format` character, output file type. See `writeFormats`.
- `dataType` character, output data type. See `dataType`.
- `overwrite` logical, default TRUE, should the resulting raster be overwritten.
transfer_raster_RData

Value
At the designated path (outputPath) the user will find TIF file(s).

Examples

```r
inputPath = system.file("extdata", package = "geoTS")
masterFile = system.file("extdata", "master.tif", package = "geoTS")
transfer_bin_raster(inputPath = inputPath, outputPath = inputPath,
master = masterFile, what = integer(),
signed = TRUE, endian = "little", size = 2,
format = "GTiff", dataType = "INT2S", overwrite = TRUE)
```

---

transfer_raster_RData  Transfer values from a Raster* object to an RData file

Description
Get the values of a Raster*, storage them into an array and finally save the array in an RData which allows for compatibility with multiple R functions as well as great portability.

Usage

```r
transfer_raster_RData(inputFile, outputPath, vmode = c("integer",
            "single", "double"))
```

Arguments

- **inputFile**: character with full path name of input file.
- **outputPath**: character with full path name (where the RData file will be saved). Do not include the extension .RData.
- **vmode**: a character specifying the type of virtual storage mode vmode needed. Only integer, single and double are allowed.

Details
Prior to embark the user in a transfer that may not be successful due to the lack of RAM, this function provides an estimate of the amount of bytes to be used in the transfer process. The estimate is obtained by multiplying the number of rows by the number of columns by the number of layers of the Raster* object to transfer by the amount of bites used by vmode (32-bit float for integer or single and 64-bit float for double). Should the user decide not to continue with the importation transfer_raster_RData returns the message "Did not transfer anything".

Value
At the designated path (outputPath) the user will find an RData file.
See Also

  vmode

Examples

```r
inputFile = system.file("extdata", "master.tif", package = "geoTS")
outputPath = paste0(system.file("extdata", package = "geoTS"), "/test")
transfer_raster_RData(inputFile = inputFile, outputPath = outputPath,
             vmode = "single")
```
Index

*Topic package
  geoTS-package, 2

aggregate, 8
array, 2, 9
coordinates, 5
dataType, 7, 8
doParallel, 8
geoTS-package, 2
haRmonics, 2, 3
matrixToRaster, 5
maxLagMissVal, 6
projection, 5
raster, 2
rasterOptions, 8
readBin, 8
reclassify, 2
rle, 6
split_replace, 2, 7
tiff, 2
transfer_bin_raster, 2, 8
transfer_raster_RData, 2, 9
vmode, 9, 10
writeFormats, 7, 8
writeRaster, 2, 7, 8