# Package ‘greenbrown’

November 18, 2016

**Title** Land Surface Phenology and Trend Analysis

**Version** 2.4.2

**Date** 2016-10-27

**Author** Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

**Maintainer** Matthias Forkel <matthias.forkel@geo.tuwien.ac.at>, Thomas Wutzler <twutz@bgc-jena.mpg.de>

**Description** Collection of functions to analyse trends, trend changes and phenology events in gridded time series like from satellite observations or climate model simulations. The package provides access to different methods for 1) trend and breakpoint analysis, 2) time series smoothing and interpolation, and 3) analysis of land surface phenology.

**Depends** R (>= 2.15.3), strucchange, raster, zoo

**Imports** Kendall, bfast, phenopix, fields, sp, ncdf4, RColorBrewer, parallel, plyr

**Suggests**

**License** GPL-2

**URL** http://greenbrown.r-forge.r-project.org/

**LazyLoad** yes

## R topics documented:

---

`greenbrown-package` *Land Surface Phenology and Trend Analysis*

---

### Description

Collection of functions to analyse trends, trend changes and phenology events in gridded time series like from satellite observations or climate model simulations. The package provides access to different methods for 1) trend and breakpoint analysis, 2) time series smoothing and interpolation, and 3) analysis of land surface phenology.

### Details

| | |
|---|---|
| Package: | greenbrown |
| Type: | Package |
| Version: | 2.4.3 |
| Date: | 2015-11-17 |
| License: | GPL-2 |

Satellite observations are used to monitor temporal changes of the terrestrial vegetation. Satellite-derived time series of vegetation indices such as Normalized Difference Vegetation Index (NDVI) are indicative of the coverage of green vegetation, photosynthetic activity and green biomass. However, the analysis of vegetation index time series is often dependent on the used analysis methods. The package provides access to different methods for 1) trend and breakpoint analysis, 2) time series smoothing and interpolation, and 3) analysis of land surface phenology.

The methods for trend and breakpoint analysis mostly refer to Forkel et al. (2013). For phenology methods refer to Forkel et al. (2015).

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

Maintainer: Matthias Forkel <matthias.forkel@geo.tuwien.ac.at>, Thomas Wutzler <twutz@bgc-jena.mpg.de>

### References

Forkel, M., Carvalhais, N., Verbesselt, J., Mahecha, M., Neigh, C., Reichstein, M., 2013. Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. Remote Sensing 5, 2113-2144. doi:10.3390/rs5052113

Forkel, M., Migliavacca, M., Thonicke, K., Reichstein, M., Schaphoff, S., Weber, U., Carvalhais, N., 2015. Codominant water control on global interannual variability and trends in land surface phenology and greenness. Glob Change Biol 21, 3414–3435. doi:10.1111/gcb.12950

### See Also

http://greenbrown.r-forge.r-project.org/

---

## `AccuracyAssessment` *Accuracy assessment from a contingency table*

---

### Description

This function takes a contingency table as calculated with `table` or `crosstab` and computes an accuracy assessment, including the total accuracy, the user accuracy and the producer accuracy.

### Usage

```
AccuracyAssessment(tab)
```

### Arguments

| | |
|---|---|
| `tab` | contingency table as calculated with `table` or `crosstab` |

### Value

The function returns the same frequency table as the input but with added row and column totals and total accuracy, user accuracy and producer accuracy.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Congalton, R.G. (1991): A review of assessing the accuracy of classifications of remotely sensed data. - Remote Sensing of Environment 1991, 37, 35-46.

### See Also

`CompareClassification`, `Kappa`, `TrendClassification`

### Examples

```
# two classifications:
a <- c(1, 1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 1, 1, 1, 2, 2, 2, 3, 4, 5, 5, 3, 3, 2, 2)
b <- c(1, 2, 1, 2, 2, 2, 3, 4, 2, 2, 5, 1, 2, 2, 2, 1, 2, 3, 4, 5, 5, 3, 3, 2, 2)

# calculate first a contingency table
tab <- table(a, b)

# calculate now the accuracy assessment
AccuracyAssessment(tab)

# calculate Kappa coeffcient
Kappa(tab)
```

---

## `AllEqual` *Check if all values in a vector are the same*

---

### Description

This function is used to check if all values in a vector are equal. It can be used for example to check if a time series contains only 0 or NA values.

### Usage

```
AllEqual(x)
```

### Arguments

| | |
|---|---|
| `x` | numeric, character vector, or time series of type ts |

### Value

The function returns TRUE if all values are equal and FALSE if it contains different values.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# check if all values are equal in the following vectors:
AllEqual(1:10)
AllEqual(rep(0, 10))
AllEqual(letters)
AllEqual(rep(NA, 10))
```

---

## AllTsteps — *Convert an irregular zoo time series to a zoo time series with all time steps*

---

### Description

Irregular 'zoo' time series with missing time steps are converted to a 'zoo' time series including all time steps. Observations at time steps that were missing in the original time series are filled with NA.

### Usage

```
AllTsteps(x, by = "day", start.jan = FALSE, end.dec = FALSE,
    exclude.feb29 = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `x` | a time series of class 'zoo' |
| `by` | time step of full time series, e.g. "day", "month" |
| `start.jan` | Should the full time series start in January? If FALSE, the full time series will start at the first observation in x. |
| `end.dec` | Should the full time series end in December? If FALSE, the full time series will end at the last observation in x. |
| `exclude.feb29` | Should 29th Februaries be excluded in case of a daily time series (i.e. if by = 'day')? |
| `...` | further arguments (currently not used) |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
x <- zoo(rnorm(5), as.Date(c("2010-01-15", "2010-02-15", "2010-07-15",
 "2010-08-15", "2010-09-15")))
x
AllTsteps(x, by="month")
```

---

## AnomaliesFiltersLags
*Calculate anomalies, lags and rolling windows*

---

### Description

This function computes several time-variant statisttics of a time series like seasonal anomalies, time lagged versions of time series, and filters time series based on running windows (using `rollapply`.

### Usage

```
AnomaliesFiltersLags(x, funSeasonalCycle = MeanSeasonalCycle,
    funFilter = median, alignFilter = c("center", "left", "right"),
    filters = c(3, 5, 7, 9, 11, 13), lags = -1:-7, ...)
```

### Arguments

| | |
|---|---|
| `x` | univariate time series of class `ts` |
| `funSeasonalCycle` | a function to estimate the seasonal cycle of the time series. |
| `funFilter` | a function to filter the time series based on rolling windows. |
| `alignFilter` | specifies whether the index of the running filter results should be left- or right-aligned or centered (default) compared to the rolling window of observations. See `rollapply` |
| `filters` | window sizes for rolling filters to be applied |
| `lags` | time lags to be applied for lagged time series |
| `...` | further arguments (currently not used) |

### Value

The function returns a multivariate time series of class 'mts' with the following columns:

- `orig` the original time series
- `msc` mean seasonal cycle as computed with `funSeasonalCycle` (repeated for the full time series length)
- `anom` anomalies releative to mean seasonal cycle
- `orig.filterX` rolling window result based on the original time series as computed with `funFilter` for the filter window size X

- `anom.filterX` rolling window result based on the anomaly time series as computed with `funFilter` for the filter window size X
- `orig.lagX` time lagged version of the original time series for the time lag X
- `msc.lagX` time lagged version of the mean seasonal cycle time series for the time lag X
- `anom.lagX` time lagged version of the anomaly time series for the time lag X

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`MeanSeasonalCycle`

### Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)

# do calculations
afl <- AnomaliesFiltersLags(ndvi)
colnames(afl)

# seasonal anomalies
plot(afl[,"anom"])

# running median filters on original time series
plot(afl[, grep("orig.filter", colnames(afl))], plot.type="single", col=rainbow(6))

# running median filters on anomalies
plot(afl[, grep("anom.filter", colnames(afl))], plot.type="single", col=rainbow(6))

# lagged versions of original time series
plot(window(afl[, grep("orig.lag", colnames(afl))], start=c(1995, 1),
   end=c(2000, 12)), plot.type="single", col=rainbow(7), type="l")

# lagged versions of anomaly time series
plot(afl[, grep("anom.lag", colnames(afl))], plot.type="single", col=rainbow(7))
```

---

## `brgr.colors` *Brown-to-green color palette for NDVI trend maps*

---

### Description

Positive trends in Normalized Difference Vegetation Index are called 'greening' whereas negative trends are called 'browning'. Creating maps of NDVI trends in these colors helps to read the map. This function provides a color scale from brown to green and can be used to plot NDVI trend maps.

### Usage

```
brgr.colors(n)
```

### Arguments

| | |
|---|---|
| n | Number of color levels |

### Value

A character vector of color names.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TrendRaster`

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)

# calculate trends and plot the result in nice brown-to-green colors
ndvitrend <- TrendRaster(ndvimap)
cols <- brgr.colors(15)
plot(ndvitrend, 2, col=cols, zlim=c(-0.004, 0.004))

classbreaks <- seq(-0.0035, 0.0035, by=0.001)
cols <- brgr.colors(length(classbreaks)-1)
plot(ndvitrend, 2, col=cols, breaks=classbreaks)
```

---

## ColorMatrix *Create a square matrix of colors*

---

### Description

This function creates a square matrix with two diagonal crossing color ramps. It can be used to plot contingency maps of two classifications.

### Usage

```
ColorMatrix(dim = 3, ul = "burlywood4", lr = "darkgreen", ll = "khaki1",
    ur = "royalblue1", ctr = "gray87")
```

### Arguments

| | |
|---|---|
| dim | number of rows and number of columns of the matrix (only square matrix are possible, i.e. number of rows = number columns) |
| ul | starting color in the upper left corner of the matrix |
| lr | ending color in the lower right corner of the matrix |
| ll | starting color in the lower left corner of the matrix |
| ur | ending color in the upper right corner of the matrix |
| ctr | color in the center of the matrix |

### Value

The function returns a square matrix of color names.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
col.m <- ColorMatrix()
plot.new()
legend("topleft", as.vector(col.m), fill=col.m, ncol=3)

col.m <- ColorMatrix(dim=5, ul="red", ll="navy", ctr="purple")
plot.new()
legend("topleft", as.vector(col.m), fill=col.m, ncol=5)
```

---

## CompareClassification

*Compare two classification maps*

---

### Description

This function computes an agreement map of two classifications (RasterLayers with classified values). Additionally, it computes a frequency table with user, producer and total accuracies as well as the Kappa coefficient.

### Usage

```
CompareClassification(x, y, names = NULL, samplefrac = 1)
```

### Arguments

| | |
|---|---|
| `x` | First raster layer with classification. |
| `y` | Second raster layer with classification. |
| `names` | a list with names of the two classifications and class names. See example section for details. |
| `samplefrac` | fraction of grid cells to be sampled from both rasters in order to calculate the contingency table |

### Value

The function returns a list of class "CompareClassification" with the following components:

- `raster` a raster layer indicating the agreement of the two classifications.
- `table` a contingency table with user, producer and total accuracies. Rows in the table correpond to the classification x, columns to the classifcation y.
- `kappa` Kappa coefficient.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`plot.CompareClassification`, `AccuracyAssessment`, `TrendClassification`

### Examples

```
# Example: calculate NDVI trends from two methods and compare the significant trends

# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)

# calculate trends with two different methods
AATmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=0)
plot(AATmap)
STMmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="STM", breaks=0)
plot(STMmap)

# classify the trend estimates from the two methods into significant
# positive, negative and no trend
AATmap.cl <- TrendClassification(AATmap)
plot(AATmap.cl, col=brgr.colors(3))
STMmap.cl <- TrendClassification(STMmap)
plot(STMmap.cl, col=brgr.colors(3))

# compare the two classifications
compare <- CompareClassification(x=AATmap.cl, y=STMmap.cl,
   names=list('AAT'=c("Br", "No", "Gr"), 'STM'=c("Br", "No", "Gr")))
compare
```

```
# plot the comparison
plot(compare)
```

---

`CorrectDOY` *Correct day-of-year time series*

---

## Description

This function corrects a time series with days-of-years (e.g. start of growing season). For example, if the start of season occurs in one year at the end of the calendar year (doy > 305) and in another year at the beginning (doy < 60), the DOYs are corrected so that all values occur at the beginning of the year (e.g. negative DOYs will be produced) or at the end of the year (e.g. DOY > 365 will be produced). This function is applied in `Phenology` after phenology detection on sos, eos, pop and pot time series (see examples).

## Usage

```
CorrectDOY(doy, check.outliers = TRUE)
```

## Arguments

`doy` a vector or time series representing DOYs

`check.outliers`
Set outliers to NA after correction? Outliers are defined here as: doy < (median - IQR*2) | doy > (median + IQR*2))

## Value

a vector or time series

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`Phenology`

## Examples

```
# imagine the following start of season DOYs in 10 years
sos <- ts(c(15, 10, 12, 8, 10, 3, 362, 2, 1, 365), start=1982)
plot(sos)
# Visually, there seems to be big differences in the start of season. However,
# there is actually only one day between the last two values (DOY 1 = 1st January,
# DOY 365 = 31st December). Trend calculation fails on this time series:
plot(Trend(sos), ylab="SOS")
```

```
# The DOY time series needs to be corrected to analyze
# the true differences between days.
sos2 <- CorrectDOY(sos)
plot(Trend(sos2), ylab="SOS")
# The correction now allows trend analysis.
# Negative DOYs indicate days at the end of the previous year!

# other example
sos <- ts(c(5, 12, 15, 120, 363, 3, 362, 365, 360, 358), start=1982)
plot(sos) # one value seems like an outlier
sos2 <- CorrectDOY(sos)
plot(Trend(sos2), ylab="SOS")
# The outlier is removed.
# DOYs > 365 indicate days in the next year!
```

---

`CropNA` *Crop outer NA values from a raster*

---

### Description

This function cuts NA values around an 'island' of real values in a Raster* object.

### Usage

```
CropNA(r, ...)
```

### Arguments

| | |
|---|---|
| `r` | Raster* object |
| `...` | other arguments, see writeRaster. |

### Value

a Raster* object with smaller extent.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

`Decompose` *Simple decomposition of time series*

## Description

This function decomposes time series in different components using a simple step-wise approach.

## Usage

```
Decompose(Yt, breaks = 0, mosum.pval = 0.05)
```

## Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `breaks` | maximal number of breaks in the trend component to be calculated (integer number). |
| `mosum.pval` | Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. |

## Details

The decomposition of the time series is based on a simple step-wise approach:

- The mean of the NDVI time series is calculated.
- In the second step, monthly values are aggregated per year by using the average value and the trend is calculated based on annual aggregated values using `TrendAAT`.
- The mean of the time series and the derived trend component from step (2) are subtracted from the annual values to derive the trend-removed and mean-centred annual values (annual anomalies). If the trend slope is not significant (p > 0.05), only the mean is subtracted.
- In the next step, the mean, the trend component and the annual anomalies are subtracted from the original time series to calculate a detrended, mean-centered and for annual anomalies adjusted time series. From this time series the seasonal cycle is estimated as the mean seasonal cycle.
- In the last step, the short-term anomalies are computed. For this, the mean, the trend component, the annual anomalies and the mean seasonal cycle are subtracted from the original time series.

## Value

The function returns a multi-variate object of class ts including the time series components.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

### See Also

`GetTsStatisticsRaster`

### Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)

# decompose this time series
ndvi.dc <- Decompose(ndvi)
plot(ndvi.dc)

ndvi.dc2 <- Decompose(ndvi, breaks=2, mosum.pval=1)
plot(ndvi.dc2)
```

---

## `FillPermanentGaps` *Fill permanent gaps in time series*

---

### Description

Satellite time series are often affected by permanent gaps like missing observations during winter periods. Often time series methods can not deal with missing observations and require gap-free data. This function fills winter gaps with a constant fill value or according to the approach described in Beck et al. (2006).

### Usage

```
FillPermanentGaps(Yt, min.gapfrac = 0.2, lower = TRUE, fillval = NA,
    fun = min, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `min.gapfrac` | How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4. |
| `lower` | fill lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL) |

`fillval` constant fill values for gaps. If NA the fill value will be estimated from the data using fun.

`fun` function to be used to compute fill values. By default, minimum.

`...` further arguments (currently not used)

### Value

The function returns a time series with filled permanent gaps.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TsPP`

### Examples

```
# load NDVI data
data(ndvi)
plot(ndvi)

# sample some winter months to be set as gaps
winter <- (1:length(ndvi))[cycle(ndvi) == 1 | cycle(ndvi) == 2 | cycle(ndvi) == 12]
gaps <- sample(winter, length(winter)*0.3)

# introduce systematic winter gaps in time series
ndvi2 <- ndvi
ndvi2[gaps] <- NA
plot(ndvi2)
IsPermanentGap(ndvi2)

# fill winter with observed minimum
fill <- FillPermanentGaps(ndvi2)
plot(fill, col="red"); lines(ndvi)

# fill winter with mean
fill <- FillPermanentGaps(ndvi2, fun=mean)
plot(fill, col="red"); lines(ndvi)

# fill winter with 0
fill <- FillPermanentGaps(ndvi2, fillval=0)
plot(fill, col="red"); lines(ndvi)
```

`FitDoubleLogBeck` *Fit a double logisitic function to a vector according to Beck et al. (2006)*

## Description

This function fits a double logistic curve to observed values using the function as described in Beck et al. (2006) (equation 3).

## Usage

```
FitDoubleLogBeck(x, t = 1:length(x), tout = t, weighting = TRUE,
    hessian = FALSE, plot = FALSE, ninit = 30, ...)
```

## Arguments

| | |
|---|---|
| `x` | vector or time series to fit |
| `t` | time steps |
| `tout` | time steps of output (can be used for interpolation) |
| `weighting` | apply the weighting scheme to the observed values as described in Beck et al. 2006? This is useful for NDVI observations because higher values will get an higher weight in the estimation of the double logisitic function than lower values. |
| `hessian` | compute standard errors of parameters based on the Hessian? |
| `plot` | plot iterations for logistic fit? |
| `ninit` | number of inital parameter sets from which to start optimization |
| `...` | further arguments (currently not used) |

## Value

The function returns a list with fitted values, parameters, fitting formula and standard errors if hessian is TRUE

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Beck, P.S.A., C. Atzberger, K.A. Hodga, B. Johansen, A. Skidmore (2006): Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. - Remote Sensing of Environment 100:321-334.

## See Also

`TSGFdoublelog`, `Phenology`

### Examples

```
# select one year of data
data(ndvi)
x <- as.vector(window(ndvi, start=c(1994,1), end=c(1994, 12)))
plot(x)

# fit double-logistic function to one year of data
fit <- FitDoubleLogBeck(x)
lines(fit$predicted, col="blue")

# do more inital trials, plot iterations and compute parameter uncertainties
FitDoubleLogBeck(x, hessian=TRUE, plot=TRUE, ninit=100)

# fit double-logistic function to one year of data,
# interpolate to daily time steps and calculate phenology metrics
tout <- seq(1, 12, length=365) # time steps for output (daily)
fit <- FitDoubleLogBeck(x, tout=tout)
PhenoDeriv(fit$predicted, plot=TRUE)
```

---

## `FitDoubleLogElmore` *Fit a double logisitic function to a vector according to Elmore et al. (2012)*

---

### Description

This function fits a double logistic curve to observed values using the function as described in Elmore et al. (2012) (equation 4).

### Usage

```
FitDoubleLogElmore(x, t = 1:length(x), tout = t, hessian = FALSE,
    plot = FALSE, ninit = 100, ...)
```

### Arguments

| | |
|---|---|
| `x` | vector or time series to fit |
| `t` | time steps |
| `tout` | time steps of output (can be used for interpolation) |
| `hessian` | compute standard errors of parameters based on the Hessian? |
| `plot` | plot iterations for logistic fit? |
| `ninit` | number of inital parameter sets from which to start optimization |
| `...` | further arguments (currently not used) |

### Value

The function returns a list with fitted values, parameters, fitting formula and standard errors if hessian is TRUE

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Elmore, A.J., S.M. Guinn, B.J. Minsley and A.D. Richardson (2012): Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. - Global Change Biology 18, 656-674.

### See Also

`TSGFdoublelog`, `Phenology`

### Examples

```
# select one year of NDVi data
data(ndvi)
x <- as.vector(window(ndvi, start=c(1991,1), end=c(1991, 12)))
plot(x)

# fit double-logistic function to one year of data
fit <- FitDoubleLogElmore(x)
fit
plot(x)
lines(fit$predicted, col="blue")

# do more inital trials, plot iterations and compute parameter uncertainties
FitDoubleLogElmore(x, hessian=TRUE, plot=TRUE, ninit=1000)

# fit double-logistic function to one year of data,
# interpolate to daily time steps and calculate phenology metrics
tout <- seq(1, 12, length=365) # time steps for output (daily)
fit <- FitDoubleLogElmore(x, tout=tout)
plot(x)
lines(tout, fit$predicted, col="blue")
PhenoDeriv(fit$predicted, plot=TRUE)
```

## `GetInfoVI3g` *Get meta-information from GIMMS VI3g binary file names*

### Description

This function extracts the date and satellite from the GIMMS VI3g file names.

### Usage

```
GetInfoVI3g(file)
```

### Arguments

| | |
|---|---|
| `file` | GIMMS VI3g file name |

### Value

The function returns a list with $date and $sat.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# GetInfoVI3g("geo00oct15a.n14-VI3g")
```

## `GetTsStatisticsRaster` *Estimate statistical properties of time series in a multi-temporal raster dataset*

### Description

This function computes statistical properties of the time series in a multi-temporal raster dataset. It calls `Decompose` to decompose the time series of each grid cell of a raster brick into a trend, inter-annual variability, seasonal and short-term variability time series components. In a next step the mean, the trend slope, the range and standard deviation of the inter-annual variability, the range of the seasonal cycle as well as the range and standard devaition of the short-term variability are calculated.

### Usage

```
GetTsStatisticsRaster(r, start = c(1982, 1), freq = 12)
```

### Arguments

| | |
|---|---|
| `r` | object of class `brick` with multi-temporal data. |
| `start` | first time step, e.g. c(1982, 1) for January 1982. See `ts` for details. |
| `freq` | the number of observations per unit of time, e.g. 12 for monthly data or 24 for bi-monthly data. See `ts` for details. |

### Value

The function returns a RasterBrick with 7 layers: mean, trend slope, range of inter-annual variabililty, standard deviation of inter-annual variabililty, range of seasonal cycle, range and standard deviation of short-term variabililty.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`Decompose`

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# calculate time series statistics
ndvimap.tsstat <- GetTsStatisticsRaster(ndvimap)
plot(ndvimap.tsstat)
```

---

## `Greenup` *Identify greenup and senescence periods in time series*

---

### Description

The function identifies 'greenup' (i.e. periods with increase) and 'senescence' (i.e. periods with decrease) in time series. This function implements threshold methods for phenology. Please use the function `Phenology` to apply this method.

### Usage

```
Greenup(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | vector of values |
| `...` | further arguments (currently not used) |

### Value

The function returns a boolean vector: TRUE (greenup) and FALSE (senescence).

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`Phenology`

---

## `InterpolateMatrix` *Interpolate NA values in a matrix using a moving mean*

---

### Description

This function interpolates missing values in a matrix with the mean of the neighbouring matrix cells.

### Usage

```
InterpolateMatrix(m)
```

### Arguments

| | |
|---|---|
| `m` | a matrix with NA value to interpolate |

### Value

matrix with interpolated values

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
m <- matrix(1:25, 5, 10)
m[sample(1:50, 10)] <- NA
m
InterpolateMatrix(m)
```

## IsPermanentGap — *Identify if a gap in a time series occurs permanently*

### Description

The function identifies obervations in time series as permanent gaps if the gap occurs during the same period in several years.

### Usage

```
IsPermanentGap(Yt, min.gapfrac = 0.2, lower = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `min.gapfrac` | How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4. |
| `lower` | identify lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL) |
| `...` | further arguments (currently not used) |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TsPP`

### Examples

```
# load NDVI data
data(ndvi)

# introduce some systematic gaps in january, february, december and july
gaps <- ndvi
winter <- cycle(ndvi) == 1 | cycle(ndvi) == 2 | cycle(ndvi) == 12 | cycle(ndvi) == 7
gaps[winter] <- NA
gaps[1] <- 0.2
gaps[7] <- 0.3
plot(gaps)

# identifiy permanent winter gaps only
IsPermanentGap(gaps, lower=TRUE)

# identify permanent summer gaps
```

```
IsPermanentGap(gaps, lower=FALSE)

# identify all permanent gaps
IsPermanentGap(gaps, lower=NULL)
```

---

`Kappa` *Calculate the Kappa coefficient of two classifications*

---

### Description

This function takes a contingency table as calculated with `table` or `crosstab` and computes the Kappa coefficient.

### Usage

```
Kappa(tab)
```

### Arguments

| | |
|---|---|
| `tab` | contingency table as calculated with `table` or `crosstab` |

### Value

Kappa coeffcient

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Congalton, R.G. (1991): A review of assessing the accuracy of classifications of remotely sensed data. - Remote Sensing of Environment 1991, 37, 35-46.

### See Also

`CompareClassification`, `AccuracyAssessment`, `TrendClassification`

### Examples

```
# two classifications:
a <- c(1, 1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 1, 1, 1, 2, 2, 2, 3, 4, 5, 5, 3, 3, 2, 2)
b <- c(1, 2, 1, 2, 2, 2, 3, 4, 2, 2, 5, 1, 2, 2, 2, 1, 2, 3, 4, 5, 5, 3, 3, 2, 2)

# calculate first a contingency table
tab <- table(a, b)

# calculate now the accuracy assessment
```

```
AccuracyAssessment(tab)

# calculate Kappa coeffcient
Kappa(tab)
```

---

KGE *Compute Kling-Gupta efficiency and related metrics of two time series*

---

### Description

This function is an implementation of the Kling-Gupta efficiency (KGE) (Gupta et al. 2009) for model evaluation. It was originally developed to compare modelled and observed time series. The KGE is a model evluation criterion that can be decomposed in the contribution of mean, variance and correlation on model performance. In this implemenation, the Kling-Gupta effciency is defined as following: KGE = 1 - eTotal eTotal is the euclidean distance of the actual effects of mean, variance, correlation and trend (optional) on the time series: eTotal = sqrt(eMean + eVar + eCor + eTrend) eTotal can be between 0 (perfect fit) and infinite (worst fit).

### Usage

```
KGE(x, ref, trend = FALSE, mosum.pval = 0.05, h = 0.15, breaks = 0,
    eTrend.ifsignif = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `x` | time series from model result or factorial model run |
| `ref` | reference time series (observation or standard model run) |
| `trend` | Include the effect of trend in the calculation? (default: FALSE) |
| `mosum.pval` | (only used if trend=TRUE) See `Trend` for details. |
| `h` | (only used if trend=TRUE) See `Trend` for details. |
| `breaks` | (only used if trend=TRUE) See `Trend` for details. |
| `eTrend.ifsignif` | compute effect on trend only if trend in reference series is significant, if FALSE compute always effect on trend (if trend = TRUE) |
| `...` | further arguments for the function `Trend` |

### Value

The function returns a vector with the following components:

- `KGE` Kling-Gupta effciency = 1 - eTotal
- `eTotal` total effect, i.e. euclidean distance
- `fMean` fraction of mean to the total effect
- `fVar` fraction of variance to the total effect

- `fCor` fraction of correlation to the total effect
- `fTrend` fraction of trend to the total effect (only if trend=TRUE)
- `eMean` effect of mean
- `eVar` effect of variance
- `eCor` effect of correlation
- `eTrend` effect of trend (only if trend=TRUE)

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Gupta, H. V., H. Kling, K. K. Yilmaz, G. F. Martinez (2009): Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling. Journal of Hydrology 377, 80-91.

### See Also

`Trend`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# change the variance and compute effect
x <- ndvi + rnorm(length(ndvi), 0, 0.01)
plot(x, ndvi); abline(0,1)
KGE(x, ndvi, trend=FALSE)

# change mean and variance and compute effect
x <- ndvi + rnorm(length(ndvi), 0.02, 0.01)
plot(x, ndvi); abline(0,1)
KGE(x, ndvi, trend=FALSE)

# be careful when using trends and breakpoints
# using trends is howver not part of the original implementation
# of the Kling-Gupta efficiency
KGE(x, ndvi, trend=TRUE, breaks=0)
KGE(x, ndvi, trend=TRUE, breaks=1)
```

`KGERaster` *Compute Kling-Gupta efficiency and related metrics of two multi-layer raster data sets*

## Description

This function can be used to apply the function `KGE` on raster data. See `KGE` for details.

## Usage

```
KGERaster(x, ref, trend = FALSE, start = c(1982, 1), freq = 12,
     ...)
```

## Arguments

| | |
|---|---|
| `x` | multi-layer raster object of class `brick` including modelled time series |
| `ref` | multi-layer raster object of class `brick` including reference (observed or standard model run) time series |
| `trend` | Include the effect of trend in the calculation? (default: FALSE). The calculation of breakpoints is currently not implemented for the function KGERaster. |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |
| `...` | further arguments for the function `calc` |

## Details

See `KGE` for details.

## Value

The function returns a raster brick with the following layers:

- `KGE` Kling-Gupta effciency = 1 - eTotal
- `eTotal` total effect, i.e. euclidean distance
- `fMean` fraction of mean to the total effect
- `fVar` fraction of variance to the total effect
- `fCor` fraction of correlation to the total effect
- `fTrend` fraction of trend to the total effect (only if trend=TRUE)
- `eMean` effect of mean
- `eVar` effect of variance
- `eCor` effect of correlation
- `eTrend` effect of trend (only if trend=TRUE)

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Gupta, H. V., H. Kling, K. K. Yilmaz, G. F. Martinez (2009): Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling. Journal of Hydrology 377, 80-91.

### See Also

KGE, Trend

### Examples

```
# # load a map of NDVI (normalized difference vegetation index) time series
# data(ndvimap)
# plot(ndvimap)

# # increase mean
# ndvimap2 <- ndvimap + 0.01
# kge1.r <- KGERaster(x=ndvimap2, ref=ndvimap)
# plot(kge1.r)

# # increase mean and variance
# ndvimap3 <- ndvimap + 0.01 + rnorm(1000, 0, 0.05)
# kge2.r <- KGERaster(ndvimap3, ndvimap)
# plot(kge2.r)

# # check also effects on trend (takes more time because of trend calculations)
# kge3.r <- KGERaster(ndvimap3, ndvimap, trend=TRUE)
# plot(kge3.r)
```

---

## KGETrendUncertainty *Compute uncertainty of Kling-Gupta efficiency based on beginning and end of time series*

---

### Description

This function samples time series for different combinations of start and end years and computes for each combination the KGE (see KGE).

### Usage

```
KGETrendUncertainty(x, ref, trend = TRUE, eTrend.ifsignif = FALSE,
    sample.method = c("sample", "all", "none"), sample.min.length = 0.75,
    sample.size = 30, ...)
```

### Arguments

| | |
|---|---|
| `x` | time series from model result or factorial model run |
| `ref` | reference time series (observation or standard model run) |
| `trend` | Include the effect of trend in the calculation? |
| `eTrend.ifsignif` | compute effect on trend only if trend in reference series is significant, if FALSE compute always effect on trend (if trend = TRUE) |
| `sample.method` | Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates). |
| `sample.min.length` | Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation. |
| `sample.size` | sample size (number of combinations of start and end dates) to be used if `method` is sample. |
| `...` | further arguments for the function `Trend` |

### Details

...

### Value

The function returns a data.frame with the following components:

- `start` start of the time series
- `end` end of the time series
- `length` length of the time series
- `KGE` Kling-Gupta effciency = 1 - eTotal
- `eTotal` total effect, i.e. euclidean distance
- `fMean` fraction of mean to the total effect
- `fVar` fraction of variance to the total effect
- `fCor` fraction of correlation to the total effect
- `fTrend` fraction of trend to the total effect (only if trend=TRUE)
- `eMean` effect of mean
- `eVar` effect of variance
- `eCor` effect of correlation
- `eTrend` effect of trend (only if trend=TRUE)

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Gupta, H. V., H. Kling, K. K. Yilmaz, G. F. Martinez (2009): Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling. Journal of Hydrology 377, 80-91.

### See Also

`Trend`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# change the variance and compute effect
x <- ndvi + rnorm(length(ndvi), 0, 0.01)
plot(x, ndvi); abline(0,1)
unc <- KGETrendUncertainty(x, ndvi)
hist(unc$KGE)
```

---

## `LmSeasonalCycle` *Calculate the mean seasonal cycle of a time series based on a linear model*

---

### Description

The function calculates the mean seasonal cycle of a time series based on a linear regression between the values and the time. Therefore a linear model with interactions is fitted to the original values Y of the form: Y = (a * m) * (b * sin(m)) * (c * cos(m)) + d where m are the the seasonal indices (e.g. months).

### Usage

```
LmSeasonalCycle(ts)
```

### Arguments

| | |
|---|---|
| `ts` | univariate time series of class `ts` |

### Value

Mean seasonal cycle of time series ts with the same length as ts, i.e. the mean seasonal cycle is repeated for each year. The mean seasonal cycle is centered to 0.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

Decompose, TrendSeasonalAdjusted, MeanSeasonalCycle

### Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)
ndvi.lmcycl <- LmSeasonalCycle(ndvi)
plot(ndvi.lmcycl)

ndvi.meancycl <- MeanSeasonalCycle(ndvi)
plot(ndvi.lmcycl[1:12], col="red", type="l")
lines(ndvi.meancycl[1:12], col="blue")
```

---

## MapBreakpoints — *Plot map of breakpoints*

---

### Description

This function plots a map of breakpoints or adds breakpoints as points and text to map of trends.

### Usage

```
MapBreakpoints(bp.r, add = TRUE, add.text = TRUE, ntext = NULL,
    breaks = NULL, col = NULL, cex = 0.6, lwd = 0.6, pch = 1,
    format.text, ...)
```

### Arguments

| | |
|---|---|
| bp.r | raster layer with breakpoints as computed with TrendRaster. |
| add | add breakpoint map to actual map (default TRUE) |
| add.text | add text (i.e. year of breakpoint) to regional groups of breakpoints |
| ntext | number of regional groups of breakpoints that should be labelled with text |
| breaks | class breaks to color breapoints (if NULL will be computed automatically) |
| col | colors for breakpoints |
| cex | size of point symbols |
| lwd | line width of point symbols |
| pch | type of point symbol |
| format.text | format of the text if add.text=TRUE, default: %y |
| ... | further arguments for plot |

### Value

The function returns a list with class colors and breaks that was used for plotting

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TrendRaster`, `TrendSegmentsRaster`

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
ndvimap
plot(ndvimap, 8)

# calculate trend and breakpoints
AATmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=1)
plot(AATmap)

# plot trend slope and add breakpoints
bp.r <- raster(AATmap, grep("BP1", names(AATmap)))
plot(AATmap, grep("SlopeSEG1", names(AATmap)), col=brgr.colors(15))
MapBreakpoints(bp.r)

plot(AATmap, grep("SlopeSEG1", names(AATmap)), col=brgr.colors(15))
lgd <- MapBreakpoints(bp.r, format.text="%Y", ntext=10, cex=0.8)
```

---

## `MeanSeasonalCycle` *Calculate the mean seasonal cycle of a time series*

---

### Description

The function calculates the mean seasonal cycle of a time series.

### Usage

```
MeanSeasonalCycle(ts)
```

### Arguments

| | |
|---|---|
| `ts` | univariate time series of class `ts` |

## Value

Mean seasonal cycle of time series ts with the same length as ts, i.e. the mean seasonal cycle is repeated for each year. The mean seasonal cycle is centered to 0.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

Decompose, TrendSeasonalAdjusted

## Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)
ndvi.cycle <- MeanSeasonalCycle(ndvi)
plot(ndvi.cycle)

# the mean seasonal cycle is centered to 0,
# add the mean of the time series if you want to overlay it with the original data:
plot(ndvi)
lines(ndvi.cycle + mean(ndvi, na.rm=TRUE), col="blue")
```

---

# NamesPhenologyRaster

*Get the layer names for a PhenologyRaster raster brick*

---

## Description

This function returns the layer names of a raster brick that was created using PhenologyRaster

## Usage

```
NamesPhenologyRaster(x, start = NULL)
```

## Arguments

| | |
|---|---|
| x | RasterBrick as created with PhenologyRaster or integer as the number of years of the input data when the function PhenologyRaster was called. |
| start | beginning of the time series. |

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`PhenologyRaster`

### Examples

```
# # load a multi-temporal raster dataset of Normalized Difference Vegetation Index
# data(ndvimap)
# plot(ndvimap, 8)

# # calculate phenology
# phenmap <- PhenologyRaster(ndvimap, start=c(1982, 1), freq=12)
# plot(phenmap)
# plot(phenmap)

# # layer names of the result
# NamesPhenologyRaster(30)
# NamesPhenologyRaster(phenmap)
# NamesPhenologyRaster(phenmap, start=1982)
# names(phenmap)
```

---

## `NamesTrendRaster` *Get the layer names for a TrendRaster raster brick*

---

### Description

This function returns the layer names of a raster brick that was created using `TrendRaster`

### Usage

```
NamesTrendRaster(x)
```

### Arguments

| | |
|---|---|
| x | `RasterBrick` as created with `TrendRaster` or `integer` as the maximum number of breakpoints that was used when the function `TrendRaster` was called. |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TrendRaster`

### Examples

```
# load a raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# calculate trend with maximum 2 breakpoints
breaks <- 2
trendmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12,
   method="AAT", breaks=breaks)
plot(trendmap)

# layer names ot the result
NamesTrendRaster(breaks)
NamesTrendRaster(trendmap)
names(trendmap)

# now imagine you are loosing the layer names ...
names(trendmap) <- 1:11
plot(trendmap) # X1, X2 ... is not meaningfull. How can you get the names back?

# re-create the layer names with NamesTrendRaster
names(trendmap) <- NamesTrendRaster(trendmap)
plot(trendmap)
```

---

## ndvi — *Time series of Normalized Difference Vegetation Index*

---

### Description

This is an example time series of Normalized Difference Vegetation Index (NDVI) from a grid cell in central Alaska. NDVI is a measure of vegetation greenness and is related to the coverage of green vegetation, photosynthetic activity and green biomass. NDVI ranges between -1 and 1. Bare ground and snow has usually NDVI values below 0.2 while vegetated areas have NDVI values above 0.2. This NDVI time series is from the GIMMS (Global Inventory, Monitoring, and Modeling Studies) dataset (Tucker et al. 2005) that provides NDVI estimates from AVHRR (Advanced Very High Resolution Radiometer) satellite observations.

### Format

A object of class `ts`.

### References

Tucker, C.; Pinzon, J.; Brown, M.; Slayback, D.; Pak, E.; Mahoney, R.; Vermote, E.; El Saleous, N., An extended AVHRR 8-km NDVI dataset compatible with MODIS and SPOT VEGETATION NDVI data. International Journal of Remote Sensing 2005, 26, 4485-4498.

### Examples

```
data(ndvi)
ndvi
plot(ndvi)
```

---

`ndvimap` *Map of Normalized Difference Vegetation Index*

---

### Description

This dataset is a multi-temporal map of Normalized Difference Vegetation Index (NDVI) from central Alaska. NDVI is a measure of vegetation greenness and is related to the coverage of green vegetation, photosynthetic activity and green biomass. NDVI ranges between -1 and 1. Bare ground and snow has usually NDVI values below 0.2, while vegetated areas have NDVI values above 0.2. Therefore, grid cells with long-term mean NDVI values below 0.2 were masked with NA. This mutli-temporal NDVI map is from the GIMMS (Global Inventory, Monitoring, and Modeling Studies) NDVI3g dataset that provides NDVI estimates from AVHRR (Advanced Very High Resolution Radiometer) satellite observations for 1982-2011. The original dataset has a bi-weekly temporal resolution. Some observations might be affected from snow or cloud cover. Therefore, the bi-weekly NDVI values were aggregated to monthly values using the maximum value. Each layer of this raster brick is a monthly time step between January 1982 and December 2011. The GIMMS NDVI3g dataset can be obtained from https://nex.nasa.gov/nex/projects/1349/.

### Format

A object of class `brick`.

### References

Tucker, C.; Pinzon, J.; Brown, M.; Slayback, D.; Pak, E.; Mahoney, R.; Vermote, E.; El Saleous, N., An extended AVHRR 8-km NDVI dataset compatible with MODIS and SPOT VEGETATION NDVI data. International Journal of Remote Sensing 2005, 26, 4485-4498.

### Examples

```
# load the NDVI map
data(ndvimap)

# print summary
ndvimap

# plot map
plot(ndvimap)
plot(ndvimap, 42) # plot selected time step

# extract time series of a specific grid cell
xy <- cbind(-152, 67) # coordinates of the grid cell
ndvi.xy <- extract(ndvimap, xy) # extract NDVI time series for this pixel
```

```
ndvi.xy <- as.vector(ndvi.xy)
date <- as.Date(ndvimap@z$Date) # get vector of dates
plot(date, ndvi.xy, type="l") # plot NDVI time series of the selected pixel
```

---

NoBP *Initialize an empty list with breakpoints*

---

### Description

This is an internal function to make an empty list of breakpoints. For the user there is usually no need to use this function.

### Usage

```
NoBP()
```

### Value

empty list with slot 'breakpoints'

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

---

NoTrend *Initialize an empty object of class "Trend"*

---

### Description

This is an internal function to make an empty list of class Trend. For the user there is usually no need to use this function.

### Usage

```
NoTrend(Yt)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

PhenoDeriv *Method 'Deriv' to calculate phenology metrics*

## Description

This function implements the derivative method for phenology. This is rather an internal function; please use the function `Phenology` to apply this method.

## Usage

```
PhenoDeriv(x, min.mean = 0.1, calc.pheno = TRUE, plot = FALSE,
    ...)
```

## Arguments

| | |
|---|---|
| `x` | seasonal cycle of one year |
| `min.mean` | minimum mean annual value in order to calculate phenology metrics. Use this threshold to suppress the calculation of metrics in grid cells with low average values |
| `calc.pheno` | calculate phenology metrics or return NA? |
| `plot` | plot results? |
| `...` | further arguments (currently not used) |

## Value

The function returns a vector with SOS, EOS, LOS, POP, MGS, RSP, RAU, PEAK, MSP and MAU.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`Phenology`

## Examples

```


data(ndvi)
plot(ndvi)

# perform time series preprocessing from first year of data
x <- TsPP(ndvi, interpolate=TRUE)[1:365]
plot(x)

# calculate phenology metrics for first year
```

```
PhenoDeriv(x, plot=TRUE)
```

---

`Phenology` *Calculate phenology metrics in time series*

---

**Description**

This function calculates from time series annual metrics of vegetation phenology:

- `sos` start of season
- `eos` end of season
- `los` length of season
- `pop` position of peak value (maximum)
- `pot` position of trough value (minimum)
- `mgs` mean growing season value
- `peak` peak value (maximum)
- `trough` trough value (minimum)
- `msp` mean spring value
- `mau` mean autumn value
- `rsp` rate of spring greenup (not all methods)
- `rau` rate of autumn senescence rates (not all methods)

The calculation of these metrics is performed in three steps and by using different methods:

- Step 1: Filling of permanent (winter) gaps. See `FillPermanentGaps`
- Step 2: Time series smoothing and interpolation. See `TsPP`
- Step 3: Detection of phenology metrics. Phenology metrics are estimated from the gap filled, smoothed and interpolated time series. This can be done by threshold methods (`PhenoTrs`) or by using the derivative of the time series (`PhenoDeriv`).
- Step 4: Correction of annual DOY (day of year) time series. sos, eos, pop, and pot time series are corrected to not jump between years and ouliers are removed. See (`CorrectDOY`).

**Usage**

```
Phenology(Yt, approach = c("White", "Trs", "Deriv"), min.mean = 0.1,
    trs = NULL, fpg = FillPermanentGaps, tsgf = "TSGFspline",
    interpolate = TRUE, min.gapfrac = 0.2, lower = TRUE, fillval = NA,
    fun = min, method = c("Elmore", "Beck"), check.seasonality = 1:3,
    backup = NULL, ...)
```

**Arguments**

- `Yt` — univariate time series of class `ts`
- `approach` — Approach to be used to calculate phenology metrics from smoothed time series. 'White' by sclaing annual cycles between 0 and 1 (White et al. 1997, see `PhenoTrs`); 'Trs' for simple thresholds (`PhenoTrs`); 'Deriv' by using the derivative of the smoothed function (`PhenoDeriv`).
- `min.mean` — minimum mean annual value in order to calculate phenology metrics. Use this threshold to suppress the calculation of metrics in grid cells with low average values
- `trs` — threshold to be used to determine SOS and EOS if method 'Trs' is used. If method 'Trs' is used but trs is NULL than trs will be computed from the long-term mean of Yt.
- `fpg` — Filling of permanent gaps: If NULL, permanent gaps will be not filled, else the function `FillPermanentGaps` will be applied.
- `tsgf` — Temporal smoothing and gap filling: Function to be used for temporal smoothing, gap filling and interpolation of the time series. If NULL, this step will be not applied. Otherwise a function needs to be specified. Exisiting functions that can be applied are `TSGFspline`, `TSGFlinear`, `TSGFssa`, `TSGFdoublelog`, `TSGFphenopix`
- `interpolate` — Should the smoothed and gap filled time series be interpolated to daily values?
- `min.gapfrac` — How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4.
- `lower` — For filling of permanent gaps: fill lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL)
- `fillval` — For filling of permanent gaps: constant fill values for gaps. If NA the fill value will be estimated from the data using fun.
- `fun` — For filling of permanent gaps: function to be used to compute fill values. By default, minimum.
- `method` — If 'tsgf' is TSGFdoublelog: Which kind of double logistic curve should be used to smooth the data? 'Elmore' (Elmore et al. 2012, see `FitDoubleLogElmore`) or 'Beck' (Beck et al. 2006, see `FitDoubleLogBeck`) .
- `check.seasonality` — Which methods in `Seasonality` should indicate TRUE (i.e. time series has seasonality) in order to calculate phenology metrics? 1:3 = all methods should indicate seasonality, Set to NULL in order to not perform seasonality checks.
- `backup` — Which backup algorithm should be used instead of TSGFdoublelog for temporal smoothing and gap filling if the time series has no seasonality? If a time series has no seasonal pattern, the fitting of double logistic functions is not meaningful. In this case another method can be used. Default: NULL (returns NA - no smoothing), other options: "TSGFspline", "TSGFssa", "TSGFlinear"
- `...` — further arguments (currently not used)

## Details

This function allows to calculate phenology metrics on time series. This method can be applied to gridded (raster) data using the function `PhenologyRaster`.

## Value

The function returns a "Phenology" object with the following components

- `method` Selected method.
- `series` gap-filled, smoothed and daily interpolated time series
- `sos` annual time series of start of season
- `eos` annual time series of end of season
- `los` annual time series of length of season
- `pop` annual time series of position of peak (maximum)
- `pot` annual time series of position of trough (minimum)
- `mgs` annual time series of mean growing season values
- `peak` annual time series of peak value
- `trough` annual time series of trough value
- `msp` annual time series of mean spring value
- `mau` annual time series of mean autumn value
- `rsp` annual time series of spring greenup rates (only for methods 'Deriv' and 'Logistic')
- `rau` annual time series of autumn senescence rates (only for methods 'Deriv' and 'Logistic')

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Beck, P.S.A., C. Atzberger, K.A. Hodga, B. Johansen, A. Skidmore (2006): Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. - Remote Sensing of Environment 100:321-334.
Elmore, A.J., S.M. Guinn, B.J. Minsley and A.D. Richardson (2012): Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. - Global Change Biology 18, 656-674.
White M.A., P.E. Thornton and S.W. Running (1997): A continental phenology model for monitoring vegetation responses to interannual climatic variability. - Global Biogeochemical Cycles 11, 217-234.

## See Also

`PhenologyRaster`, `TSGFspline`, `TSGFssa`, `TSGFdoublelog`, `FitDoubleLogElmore`, `FitDoubleLogBeck`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce some missing values
winter <- (1:length(ndvi))[cycle(ndvi) == 1 | cycle(ndvi) == 2 | cycle(ndvi) == 12]
ndvi[sample(winter, length(winter)*0.5)] <- NA
plot(ndvi)

# spline fit and threshold
spl.trs <- Phenology(ndvi, tsgf="TSGFspline", approach="White")
spl.trs
plot(spl.trs) # default plot: start of season, end of season, position of peak
plot(spl.trs, type=c("los")) # length of season

# plot mean spring, growing season, autumn and peak values
plot(spl.trs, type=c("msp", "mgs", "mau", "peak"))

# gap-filled and smoothed time series that was used to estimate phenology metrics
plot(spl.trs$series, col="red"); lines(ndvi)


# calculate phenology metrics using different smoothing methods and approaches
#-----------------------------------------------------------------------------

# linear interpolation/running median + threshold
lin.trs <- Phenology(ndvi, tsgf="TSGFlinear", approach="White")

# linear interpolation/running median + derivative
lin.deriv <- Phenology(ndvi, tsgf="TSGFlinear", approach="Deriv")

# spline + threshold
spl.trs <- Phenology(ndvi, tsgf="TSGFspline", approach="White")

# spline + derivative
spl.deriv <- Phenology(ndvi, tsgf="TSGFspline", approach="Deriv")

# double logistic fit + threshold
beck.trs <- Phenology(ndvi, tsgf="TSGFdoublelog", method="Beck", approach="White")

# double logistic fit + derivative
beck.deriv <- Phenology(ndvi, tsgf="TSGFdoublelog", method="Beck", approach="Deriv")

# double logistic fit + threshold
elmore.trs <- Phenology(ndvi, tsgf="TSGFdoublelog", method="Elmore", approach="White")

# double logistic fit + derivative
elmore.deriv <- Phenology(ndvi, tsgf="TSGFdoublelog", method="Elmore", approach="Deriv")

# compare results: SOS and EOS
type <- c("sos", "eos")
```

```
require(RColorBrewer)
cols <- brewer.pal(10, "Paired")
nms <- c("Lin+Trs", "Lin+Deriv", "Spline+Trs", "Spline+Deriv", "DoubleLog1+Trs",
 "DoubleLog1+Deriv", "DoubleLog2+Trs", "DoubleLog2+Deriv")
plot(lin.trs, col=cols[1], type=type, ylim=c(1, 365))
plot(lin.deriv, col=cols[2], type=type, add=TRUE)
plot(spl.trs, col=cols[3], type=type, add=TRUE)
plot(spl.deriv, col=cols[4], type=type, add=TRUE)
plot(beck.trs, col=cols[7], type=type, add=TRUE)
plot(beck.deriv, col=cols[8], type=type, add=TRUE)
plot(elmore.trs, col=cols[9], type=type, add=TRUE)
plot(elmore.deriv, col=cols[10], type=type, add=TRUE)
legend("center", nms, text.col=cols, ncol=3, bty="n")

cor(cbind(lin.trs$sos, lin.deriv$sos, spl.trs$sos, spl.deriv$sos, beck.trs$sos,
   beck.deriv$sos, elmore.trs$sos, elmore.deriv$sos), use="pairwise.complete.obs")
cor(cbind(lin.trs$eos, lin.deriv$eos, spl.trs$eos, spl.deriv$eos, beck.trs$eos,
   beck.deriv$eos, elmore.trs$eos, elmore.deriv$eos), use="pairwise.complete.obs")

# compare results: LOS
type <- c("los")
plot(lin.trs, col=cols[1], type=type, ylim=c(130, 365))
plot(lin.deriv, col=cols[2], type=type, add=TRUE)
plot(spl.trs, col=cols[3], type=type, add=TRUE)
plot(spl.deriv, col=cols[4], type=type, add=TRUE)
plot(beck.trs, col=cols[7], type=type, add=TRUE)
plot(beck.deriv, col=cols[8], type=type, add=TRUE)
plot(elmore.trs, col=cols[9], type=type, add=TRUE)
plot(elmore.deriv, col=cols[10], type=type, add=TRUE)
legend("bottom", nms, text.col=cols, ncol=5, bty="n")

# compare results: MSP, PEAK, TROUGH
type <- c("msp", "peak", "trough")
plot(lin.trs, col=cols[1], type=type, ylim=c(0.17, 0.37))
plot(lin.deriv, col=cols[2], type=type, add=TRUE)
plot(spl.trs, col=cols[3], type=type, add=TRUE)
plot(spl.deriv, col=cols[4], type=type, add=TRUE)
plot(beck.trs, col=cols[7], type=type, add=TRUE)
plot(beck.deriv, col=cols[8], type=type, add=TRUE)
plot(elmore.trs, col=cols[9], type=type, add=TRUE)
plot(elmore.deriv, col=cols[10], type=type, add=TRUE)
legend("bottom", nms, text.col=cols, ncol=5, bty="n")
```

---

## `PhenologyNCDF` *Calculate phenology metrics on time series in gridded (raster) data stored in NetCDF files*

---

### Description

This function calculates metrics of vegetation phenology on multi-temporal raster data. See `Phenology`.

- `sos` start of season
- `eos` end of season
- `los` length of season
- `pop` position of peak value (maximum)
- `pot` position of trough value (minimum)
- `mgs` mean growing season value
- `peak` peak value (maximum)
- `trough` trough value (minimum)
- `msp` mean spring value
- `mau` mean autumn value
- `rsp` rate of spring greenup (not all methods)
- `rau` rate of autumn senescence rates (not all methods)

The calculation of these metrics is performed in three steps and by using different methods:

- Step 1: Filling of permanent (winter) gaps. See `FillPermanentGaps`
- Step 2: Time series smoothing and interpolation. See `TsPP`
- Step 3: Detection of phenology metrics. Phenology metrics are estimated from the gap filled, smoothed and interpolated time series. This can be done by treshold methods (`PhenoTrs`) or by using the derivative of the time series (`PhenoDeriv`).

Tiles of large raster datasets can be processed in parallel by setting the number of nodes.

### Usage

```
PhenologyNCDF(file, path.out = getwd(), start = c(1982, 1), freq = 12,
    approach = c("White", "Trs", "Deriv"), min.mean = 0.1, trs = NULL,
    fpg = FillPermanentGaps, tsgf = "TSGFspline", interpolate = TRUE,
    min.gapfrac = 0.2, lower = TRUE, fillval = NA, fun = min,
    method = c("Elmore", "Beck"), backup = NULL, check.seasonality = 1:3,
    trend = FALSE, nodes = 1, restart = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `file` | multi-layer raster file |
| `path.out` | directory for results |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

| | |
|---|---|
| `approach` | Approach to be used to calculate phenology metrics from smoothed time series. 'White' by sclaing annual cycles between 0 and 1 (White et al. 1997, see `PhenoTrs`); 'Trs' for simple tresholds (`PhenoTrs`); 'Deriv' by using the derivative of the smoothed function (`PhenoDeriv`). |
| `min.mean` | minimum mean annual value in order to calculate phenology metrics. Use this threshold to suppress the calculation of metrics in grid cells with low average values |
| `trs` | threshold to be used to determine SOS and EOS if method 'Trs' is used. If method 'Trs' is used but trs is NULL than trs will be computed from the long-term mean of Yt. |
| `fpg` | Filling of permanent gaps: If NULL, permanent gaps will be not filled, else the function `FillPermanentGaps` will be applied. |
| `tsgf` | Temporal smoothing and gap filling: Function to be used for temporal smoothing, gap filling and interpolation of the time series. If NULL, this step will be not applied. Otherwise a function needs to be specified. Exisiting functions that can be applied are `TSGFspline`, `TSGFlinear`, `TSGFssa`, `TSGFdoublelog` |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values? |
| `min.gapfrac` | How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4. |
| `lower` | For filling of permanent gaps: fill lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL) |
| `fillval` | For filling of permanent gaps: constant fill values for gaps. If NA the fill value will be estimated from the data using fun. |
| `fun` | For filling of permanent gaps: function to be used to compute fill values. By default, minimum. |
| `method` | If 'tsgf' is TSGFdoublelog: Which kind of double logistic curve should be used to smooth the data? 'Elmore' (Elmore et al. 2012, see `FitDoubleLogElmore`) or 'Beck' (Beck et al. 2006, see `FitDoubleLogBeck`) . |
| `backup` | Which backup algorithm should be used instead of TSGFdoublelog for temporal smoothing and gap filling if the time series has no seasonality? If a time series has no seasonal pattern, the fitting of double logistic functions is not meaningful. In this case another method can be used. Default: NULL (returns NA - no smoothing), other options: "TSGFspline", "TSGFssa", "TSGFlinear" |
| `check.seasonality` | Which methods in `Seasonality` should indicate TRUE (i.e. time series has seasonality) in order to calculate phenology metrics? 1:3 = all methods should indicate seasonality, Set to NULL in order to not perform seasonality checks. |
| `trend` | Compute trends on the results of phenology analysis? If TRUE, trends will be using `TrendAAT`. |
| `nodes` | How many cluster nodes should be used for parallel computing? `makeCluster` and `clusterApply` from the snow package are used for parallel computing. If nodes = 1, parallel computing is not used. |
| `restart` | load results from files of previously calculated tiles and stack results? |
| `...` | additional arguments as for `TrendNCDF` |

### Value

The function saves several NetCDF files in a directory on disc. The files are created based on the filename of the input `file`:

- `file.SOS.nc` file with annual layers of the start of season
- `file.EOS.nc` file with annual layers of the end of season
- and so on for "LOS", "POP", "POT", "MGS", "PEAK", "TROUGH", "MSP", "MAU", "RSP", "RAU"

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Beck, P.S.A., C. Atzberger, K.A. Hodga, B. Johansen, A. Skidmore (2006): Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. - Remote Sensing of Environment 100:321-334.
Elmore, A.J., S.M. Guinn, B.J. Minsley and A.D. Richardson (2012): Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. - Global Change Biology 18, 656-674.
White M.A., P.E. Thornton and S.W. Running (1997): A continental phenology model for monitoring vegetation responses to interannual climatic variability. - Global Biogeochemical Cycles 11, 217-234.

### See Also

`PhenologyRaster`, `Phenology`, `PhenologyNCDF`, `NamesPhenologyRaster`

---

## `PhenologyRaster` *Calculate phenology metrics on time series in gridded (raster) data*

---

### Description

This function calculates metrics of vegetation phenology on multi-temporal raster data. See `Phenology`.

- `sos` start of season
- `eos` end of season
- `los` length of season
- `pop` position of peak value (maximum)
- `pot` position of trough value (minimum)
- `mgs` mean growing season value
- `peak` peak value (maximum)
- `trough` trough value (minimum)

- `msp` mean spring value
- `mau` mean autumn value
- `rsp` rate of spring greenup (not all methods)
- `rau` rate of autumn senescence rates (not all methods)

The calculation of these metrics is performed in three steps and by using different methods:

- Step 1: Filling of permanent (winter) gaps. See `FillPermanentGaps`
- Step 2: Time series smoothing and interpolation. See `TsPP`
- Step 3: Detection of phenology metrics. Phenology metrics are estimated from the gap filled, smoothed and interpolated time series. This can be done by treshold methods (`PhenoTrs`) or by using the derivative of the time series (`PhenoDeriv`).

## Usage

```
PhenologyRaster(r, start = c(1982, 1), freq = 12, approach = c("White",
    "Trs", "Deriv"), min.mean = 0.1, trs = NULL, fpg = FillPermanentGaps,
    tsgf = "TSGFspline", interpolate = TRUE, min.gapfrac = 0.2,
    lower = TRUE, fillval = NA, fun = min, method = c("Elmore",
        "Beck"), backup = NULL, check.seasonality = 1:3, ...)
```

## Arguments

| | |
|---|---|
| `r` | multi-layer raster object of class `brick` |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |
| `approach` | Approach to be used to calculate phenology metrics from smoothed time series. 'White' by sclaing annual cycles between 0 and 1 (White et al. 1997, see `PhenoTrs`); 'Trs' for simple tresholds (`PhenoTrs`); 'Deriv' by using the derivative of the smoothed function (`PhenoDeriv`). |
| `min.mean` | minimum mean annual value in order to calculate phenology metrics. Use this threshold to suppress the calculation of metrics in grid cells with low average values |
| `trs` | threshold to be used to determine SOS and EOS if method 'Trs' is used. If method 'Trs' is used but trs is NULL than trs will be computed from the long-term mean of Yt. |
| `fpg` | Filling of permanent gaps: If NULL, permanent gaps will be not filled, else the function `FillPermanentGaps` will be applied. |
| `tsgf` | Temporal smoothing and gap filling: Function to be used for temporal smoothing, gap filling and interpolation of the time series. If NULL, this step will be not applied. Otherwise a function needs to be specified. Exisiting functions that can be applied are `TSGFspline`, `TSGFlinear`, `TSGFssa`, `TSGFdoublelog` |

| | |
|---|---|
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values? |
| `min.gapfrac` | How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4. |
| `lower` | For filling of permanent gaps: fill lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL) |
| `fillval` | For filling of permanent gaps: constant fill values for gaps. If NA the fill value will be estimated from the data using fun. |
| `fun` | For filling of permanent gaps: function to be used to compute fill values. By default, minimum. |
| `method` | If 'tsgf' is TSGFdoublelog: Which kind of double logistic curve should be used to smooth the data? 'Elmore' (Elmore et al. 2012, see `FitDoubleLogElmore`) or 'Beck' (Beck et al. 2006, see `FitDoubleLogBeck`) . |
| `backup` | Which backup algorithm should be used instead of TSGFdoublelog for temporal smoothing and gap filling if the time series has no seasonality? If a time series has no seasonal pattern, the fitting of double logistic functions is not meaningful. In this case another method can be used. Default: NULL (returns NA - no smoothing), other options: "TSGFspline", "TSGFssa", "TSGFlinear" |
| `check.seasonality` | Which methods in `Seasonality` should indicate TRUE (i.e. time series has seasonality) in order to calculate phenology metrics? 1:3 = all methods should indicate seasonality, Set to NULL in order to not perform seasonality checks. |
| `...` | additional arguments as for `writeRaster` |

## Value

The function returns a RasterBrick with different phenology metrics statistics. The layers are named:

- `SOS.` start of season in year x
- `EOS.` end of season in year x
- `LOS.` length of season in year x
- `POP.` position of peak in year x
- `POT.` position of trough in year x
- `MGS.` mean growing season value in year x
- `PEAK.` peak value in year x
- `TROUGH.` trough value in year x
- `MSP.` mean spring value in year x
- `MAU.` mean autumn value in year x
- `RSP.` rate of spring greenup in year x (only if approach is 'Deriv')
- `RAU.` rate of autumn senescence in year x (only if approach is 'Deriv')

The number of years in the input raster will define the number of layers in the result.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Beck, P.S.A., C. Atzberger, K.A. Hodga, B. Johansen, A. Skidmore (2006): Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. - Remote Sensing of Environment 100:321-334.
Elmore, A.J., S.M. Guinn, B.J. Minsley and A.D. Richardson (2012): Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. - Global Change Biology 18, 656-674.
White M.A., P.E. Thornton and S.W. Running (1997): A continental phenology model for monitoring vegetation responses to interannual climatic variability. - Global Biogeochemical Cycles 11, 217-234.

### See Also

Phenology, PhenologyNCDF, NamesPhenologyRaster

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# calculate phenology metrics (this can take some time!)
phenmap <- PhenologyRaster(ndvimap, start=c(1982, 1), freq=12,
tsgf="TSGFspline", approach="Deriv")
# Select method by defining 'tsgf' (temporal smoothing and gap filling) and
# by 'approach' (method to summarize phenology metrics).
# See \code{\link{Phenology}} for examples and a comparison of methods.

plot(phenmap)
plot(phenmap, grep("SOS.1982", names(phenmap))) # start of season 1982
plot(phenmap, grep("EOS.1982", names(phenmap))) # end of season 1982
plot(phenmap, grep("LOS.1982", names(phenmap))) # length of season 1982
plot(phenmap, grep("POP.1982", names(phenmap))) # position of peak value 1982
plot(phenmap, grep("POT.1982", names(phenmap))) # position of trough value 1982
plot(phenmap, grep("MGS.1982", names(phenmap))) # mean growing season value 1982
plot(phenmap, grep("PEAK.1982", names(phenmap))) # peak value 1982
plot(phenmap, grep("TROUGH.1982", names(phenmap))) # trough value 1982
plot(phenmap, grep("MSP.1982", names(phenmap))) # mean spring value 1982
plot(phenmap, grep("MAU.1982", names(phenmap))) # mean autumn value 1982
plot(phenmap, grep("RSP.1982", names(phenmap))) # rate of spring greenup 1982
plot(phenmap, grep("RAU.1982", names(phenmap))) # rate of autumn senescence 1982

# calculate trends on length of season using TrendRaster
losmap <- subset(phenmap, grep("LOS", names(phenmap)))
plot(losmap)
lostrend <- TrendRaster(losmap, start=c(1982, 1), freq=1)
plot(lostrend)
```

```
# classify trends in length of season
lostrend.cl <- TrendClassification(lostrend)
plot(lostrend.cl, col=brgr.colors(3), breaks=c(-1.5, -0.5, 0.5, 1.5))
# only a few pixels have a positive trend in the length of growing season
```

---

`PhenopixMY` *Multi-year phenology analysis using phenopix*

---

## Description

This function takes a multi-year time series and applies curve fitting and phenology extraction functions based on the `greenProcess` function in the `phenopix` package. The function returns an object of class `PhenopixMY` (phenopix multi-year) which contains a list of `phenopix` objects. `PhenopixMY` can be plotted using `plot.PhenopixMY`.

## Usage

```
PhenopixMY(ts, fit, threshold = NULL, plot = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `ts` | a time series of class 'ts' or 'zoo' with multiple years of data |
| `fit` | fitting function to be applied, available options are: spline, beck, elmore, klosterman, gu (see `greenProcess`) |
| `threshold` | threshold to be applied to compute phenology metrics, available options are: trs, derivatives, klosterman, gu (see `greenProcess`) |
| `plot` | plot phenopix object of each year, using `plot.phenopix` |
| `...` | further arguments as in `greenProcess` |

## Value

An object of class `phenopixmy` with dedicated functions: plot(), print(). The structure is actually a list.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Filippa, G., Cremonese, E., Migliavacca, M., Galvagno, M., Forkel, M., Wingate, L., Tomelleri, E., Morra di Cella, U. and Richardson, A. D.: Phenopix: A R package for image-based vegetation phenology, Agricultural and Forest Meteorology, 220, 141-150, doi:10.1016/j.agrformet.2016.01.006, 2016.

### See Also

`greenProcess`, `plot.PhenopixMY`, `Phenology`

### Examples

```
data(ndvi)
plot(ndvi)

ppixmy <- PhenopixMY(ndvi, "spline", "trs")
plot(ppixmy)

plot(ppixmy, type="metrics")
```

---

## PhenoTrs *Method 'Trs' to calculate phenology metrics*

---

### Description

This function implements threshold methods for phenology. This is rather an internal function; please use the function `Phenology` to apply this method.

### Usage

```
PhenoTrs(x, approach = c("White", "Trs"), trs = NULL, min.mean = 0.1,
    calc.pheno = TRUE, plot = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `x` | seasonal cycle of one year |
| `approach` | approach to be used to calculate phenology metrics. 'White' (White et al. 1997) or 'Trs' for simple threshold. |
| `trs` | threshold to be used for approach "Trs" |
| `min.mean` | minimum mean annual value in order to calculate phenology metrics. Use this threshold to suppress the calculation of metrics in grid cells with low average values |
| `calc.pheno` | calculate phenology metrics or return NA? |
| `plot` | plot results? |
| `...` | further arguments (currently not used) |

### Value

The function returns a vector with SOS, EOS, LOS, POP, MGS, rsp, rau, PEAK, MSP and MAU.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

White MA, Thornton PE, Running SW (1997) A continental phenology model for monitoring vegetation responses to interannual climatic variability. Global Biogeochem Cycles 11, 217-234.

## See Also

`Phenology`

## Examples

```
data(ndvi)
plot(ndvi)

# perform time series processing for first year pof data
x <- TsPP(ndvi, interpolate=TRUE)[1:365]
plot(x)

# calculate phenology metrics for first year
PhenoTrs(x, plot=TRUE, approach="White")
PhenoTrs(x, plot=TRUE, approach="Trs", trs=0.25)
```

---

# plot.CompareClassification *plot a comparison of two classification rasters*

---

## Description

This function takes an object of class `CompareClassification` as input and plots a map of the class agreement of two classifications.

## Usage

```
## S3 method for class 'CompareClassification'
plot(x, xlab = "", ylab = "", main = "Classification agreement",
    names = NULL, ul = "burlywood4", lr = "darkgreen", ll = "khaki1",
    ur = "royalblue1", ctr = "gray87", mar = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `CompareClassification`. |
| `xlab` | A title for the x axis |
| `ylab` | A title for the y axis |
| `main` | A title for the plot |
| `names` | a list with names of the two classifications and class names. See example section for details. |
| `ul` | starting color in the upper left corner of the `ColorMatrix` |
| `lr` | ending color in the lower right corner of the `ColorMatrix` |
| `ll` | starting color in the lower left corner of the `ColorMatrix` |
| `ur` | ending color in the upper right corner of the `ColorMatrix` |
| `ctr` | color in the center of the `ColorMatrix` |
| `mar` | plot margins |
| `...` | Further arguments that can be passed `plot.default` |

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`CompareClassification`, `AccuracyAssessment`, `TrendClassification`

## Examples

```
# Example: calculate NDVI trends from two methods and compare the significant trends

# load a raster dataset of Normalized Difference Vegetation Index
data(ndvimap)

# calculate trends with two different methods
AATmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=0)
plot(AATmap)
STMmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="STM", breaks=0)
plot(STMmap)

# classify the trend estimates from the two methods into
# positive, negative and no trend
AATmap.cl <- TrendClassification(AATmap)
plot(AATmap.cl, col=brgr.colors(3))
STMmap.cl <- TrendClassification(STMmap)
plot(STMmap.cl, col=brgr.colors(3))

# compare the two classifications
compare <- CompareClassification(x=AATmap.cl, y=STMmap.cl,
names=list('AAT'=c("Br", "No", "Gr"), 'STM'=c("Br", "No", "Gr")))
compare
```

```
# plot the comparison
plot(compare)
```

---

`plot.Phenology` *Plot time series of phenology metrics*

---

### Description

This is the standard plot function for results of the `Phenology` function. See `plot.default` for further specifications of basic plots.

### Usage

```
## S3 method for class 'Phenology'
plot(x, type = c("sos", "eos", "pop"), ylab = NULL,
    ylim = NULL, add = FALSE, col = "black", add.trend = TRUE,
    ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class 'Phenology' as returned from function `Phenology` |
| `type` | varaible names that should be plotted from the `Phenology` object |
| `ylab` | a title for the y axis |
| `ylim` | limits for y-axis |
| `add` | add time series to exisiting plot? |
| `col` | line colors |
| `add.trend` | add trend lines to phenology time series? |
| `...` | Further arguments that can be passed `plot.default` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`plot.default`, `plot.ts`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate phenology metrics
phen <- Phenology(ndvi)
phen

# plot phenology metrics
plot(phen)
```

---

## plot.PhenopixMY *Plot multi-year phenopix objects*

---

### Description

Plotting methods for objects of class `PhenopixMY`

### Usage

```
## S3 method for class 'PhenopixMY'
plot(x, add = FALSE, col.fit = "black", type = "ts",
     ...)
```

### Arguments

| | |
|---|---|
| `x` | an object of class `PhenopixMY` |
| `add` | add to existing plot? |
| `col.fit` | color for fitting line |
| `type` | plot type: 'ts' plots the original data, the fitted curve and the metrics; 'metrics' plots only time series of the metrics |
| `...` | further arguments as in `plot.default` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Filippa, G., Cremonese, E., Migliavacca, M., Galvagno, M., Forkel, M., Wingate, L., Tomelleri, E., Morra di Cella, U. and Richardson, A. D.: Phenopix: A R package for image-based vegetation phenology, Agricultural and Forest Meteorology, 220, 141-150, doi:10.1016/j.agrformet.2016.01.006, 2016.

### See Also

`plot.phenopix`, `PhenopixMY`

### Examples

```
data(ndvi)
plot(ndvi)

ppixmy <- PhenopixMY(ndvi, "spline", "trs")
plot(ppixmy)

plot(ppixmy, type="metrics")
```

---

## `plot.Trend` *Plot trend and breakpoint results*

---

### Description

This is the standard plot function for results of the `Trend` function. See `plot.default` for further specifications of basic plots.

### Usage

```
## S3 method for class 'Trend'
plot(x, ylab = "NDVI", add = FALSE, col = c("black", "blue",
    "red", "blue"), lty = c(2, 1, 2, 3), lwd = 1, symbolic = TRUE,
    legend = FALSE, uncertainty = TRUE, axes = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class 'Trend' as returned from function `Trend` |
| `ylab` | A title for the y axis |
| `add` | add to exisiting plot |
| `col` | colors for (1) time series, (2) trend line, (3) breakpoints and (4) trend uncertainty |
| `lty` | line types for (1) time series, (2) trend line, (3) breakpoints and (4) trend uncertainty |
| `lwd` | |
| `symbolic` | add significance as symbols (TRUE). If TRUE the p-value of a trend slope is added as symbol as following: *** (p <= 0.001), ** (p <= 0.01), * (p <= 0.05), . (p <= 0.1) and no symbol if p > 0.1. |
| `legend` | add slope and p-value as legend |
| `uncertainty` | plot uncertainty in trend slopes? (only possible if the x 'Trend' object includes uncertainty estimates) |
| `axes` | plot axes? |
| `...` | Further arguments that can be passed `plot.default` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`plot.default`, `plot.ts`

### Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)

# calculate a trend and look at the results
ndvi.trend <- Trend(ndvi)
ndvi.trend
plot(ndvi.trend)
plot(ndvi.trend, uncertainty=FALSE)

ndvi.trend.aat <- Trend(ndvi, method="AAT", mosum.pval=1)
plot(ndvi.trend.aat)
plot(ndvi.trend.aat, symbolic=FALSE)
plot(ndvi.trend.aat, symbolic=FALSE, uncertainty=FALSE)

ndvi.trend.stm <- Trend(ndvi, method="STM", mosum.pval=1)
plot(ndvi.trend.stm)

plot(ndvi.trend.aat, symbolic=TRUE, ylim=c(0.23, 0.31),
   col=c("blue", "blue", "red"))
plot(ndvi.trend.stm, symbolic=TRUE, col=c("darkgreen", "darkgreen", "red"),
lty=c(0, 1, 1), add=TRUE)
```

---

## `plot.TrendGradient` *Plotting function for objects of class TrendGradient*

---

### Description

This function plots a gradient of trend slopes (e.g. latitudinal gradient).

### Usage

```
## S3 method for class 'TrendGradient'
plot(x, type = "xy", ylab = NULL, xlab = NULL,
     col = "black", ylim = NULL, xlim = NULL, add = FALSE, symbolic = TRUE,
     symbols = "standard", ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class 'TrendGradient' as returned from function `TrendGradient` |
| `type` | plotting type: 'xy' = gradient at x axis and slope at y axis, 'yx' = gradient at y axis and slope at x axis. |
| `ylab` | A title for the y axis |
| `xlab` | A title for the x axis |
| `col` | line colors |
| `ylim` | limits for y axis |
| `xlim` | limits for x axis |
| `add` | add to exisiting plot? |
| `symbolic` | Add p-value as symbols (TRUE) or not (FALSE). If TRUE the p-value of a trend slope is added as symbol to the plot. |
| `symbols` | Type of symbols for p-values. "standard": *** (p <= 0.001), ** (p <= 0.01), * (p <= 0.05), . (p <= 0.1) and no symbol if p > 0.1.; "simple": * (p <= 0.05), x (p < 0.1) |
| `...` | Further arguments that can be passed `plot.default` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`plot.default`, `plot.ts`

### Examples

```
# load a raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# compute a latitudinal gradient of trends (by default the method 'AAT' is used)
gradient <- TrendGradient(ndvimap, start=c(1982, 1), freq=12)
gradient
plot(gradient)
# shown is the trend at each latitudinal band, the area represents the 95%
# confidence interval of the trend (computed with function TrendUncertainty),
# symbols indicate the p-value of the trend at each latitude

plot(gradient, type="yx") # the gradient can be also plotted in reversed order

# compute gradients with different trend methods
gradient.mac <- TrendGradient(ndvimap, start=c(1982, 1), freq=12,
   method="SeasonalAdjusted", funSeasonalCycle=MeanSeasonalCycle)
plot(gradient.mac, col="blue", ylab="NDVI trend (month-1)")

# method AAT uses annual time steps, convert years -> months
```

```
gradient$Slope <- gradient$Slope / 12
gradient$SlopeUncLower <- gradient$SlopeUncLower / 12
gradient$SlopeUncUpper <- gradient$SlopeUncUpper / 12
gradient$SlopeUncMedian <- gradient$SlopeUncMedian / 12
plot(gradient, col="red", add=TRUE)
```

---

`plot.TrendSample` *Plot uncertainty of estimated trend dependent on start and end dates of time series*

---

## Description

Plotting function for objects of class `TrendSample`. The function plots a point scatter plot defined by first year (x-axis) and last year (y-axis) of the time series. For each combination of first and last year a point symbol is plotted that represents the estimated trend. The size of the point indicates the absolute value of the trend slope. The color of the point indicates the trend slope direction (blue = negative trend, red = positive trend). The symbol of the point indicates that p-value of the Mann-Kendall trend test (snowflake: p <= 0.05, cross: 0.05 < p <= 0.1, circle: p > 0.1). Additionally, a second plot is added to the main plot (only if full = TRUE). This second plot is a scatter plot of trend slope against p-value (Mann-Kendall trend test) using the same points symbols as in the main plot. Thus the second plot can serve as a legend for the symbols used in the main plot. A boxplot on top of the second plot shows the distribution of the trend slope.

## Usage

```
## S3 method for class 'TrendSample'
plot(x, full = TRUE, response = "slope", ...)
```

## Arguments

| | |
|---|---|
| `x` | objects of class `TrendSample` |
| `full` | make full plot or plot only main plot? |
| `response` | plot linear trend 'slope' or 'tau' from Mann-Kendall trend test as response variable. |
| `...` | further arguments to `plot` |

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`TrendSample`, `TrendUncertainty`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)

# calculate uncertainty of trend dependent on start and end dates
ndvi <- aggregate(ndvi, FUN=mean)
trd.ens <- TrendSample(ndvi)
trd.ens

# plot relations between start, end dates, length and trend statistics
plot(trd.ens)
plot(trd.ens, response="tau")
```

---

## PlotPhenCycle *Plot a easonal cycle with phenology metrics*

---

### Description

This function plots a seasonal cycle with phenology metrics.

### Usage

```
PlotPhenCycle(x, xpred = NULL, metrics, xlab = "DOY", ylab = "NDVI",
    trs = NULL, main = "", ...)
```

### Arguments

| | |
|---|---|
| `x` | values of one year |
| `xpred` | smoothed/predicted values |
| `metrics` | vector of pheology metrics |
| `xlab` | label for x-axis |
| `ylab` | label for y-axis |
| `trs` | threshold for threshold methods |
| `main` | title |
| `...` | further arguments (currently not used) |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`Phenology`

## Examples

```
data(ndvi)
plot(ndvi)

# perform time series preprocessing for first year of data
x <- TsPP(ndvi, interpolate=TRUE)[1:365]
plot(x)

# calculate phenology metrics for first year
metrics <- PhenoTrs(x, approach="White")
PlotPhenCycle(x, metrics=metrics)
```

---

# PolygonNA — *Plot a polygon by accounting for NA values (breaks in polygon)*

---

## Description

This function is an improved version of `polygon` that considers NA values in plotting.

## Usage

```
PolygonNA(x, lower, upper, col = "grey")
```

## Arguments

| | |
|---|---|
| `x` | vector of x-values |
| `lower` | vector of lower polygon range |
| `upper` | vector of upper polygon range |
| `col` | color of the polygon |

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## Examples

```
x <- 1:10
med <- rnorm(length(x))
lower <- med - 2
upper <- med + 2

# example 1: no NA values
plot(x, med, type="l", ylim=range(c(lower, upper), na.rm=TRUE))
PolygonNA(x, lower, upper)
lines(x, med)
```

```
# example 2: with some NA values
lower1 <- lower
upper1 <- upper
lower1[c(1, 6, 10)] <- NA
upper1[c(1:2, 6)] <- NA
plot(x, med, type="l", ylim=range(c(lower, upper), na.rm=TRUE))
PolygonNA(x, lower1, upper1)
lines(x, med)
```

---

## `print.Phenology` *Prints phenology metrics*

---

### Description

The function prints an object of class `Phenology`.

### Usage

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

### Arguments

| | |
|---|---|
| `x` | Object of class 'Phenology' as returned from function `Phenology` |
| `...` | further arguments (not used) |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate phenology metrics
phen <- Phenology(ndvi)
phen
print(phen)

# plot phenology metrics
plot(phen)
```

## print.Trend — *Prints trends*

### Description

The function prints an object of class `Trend`.

### Usage

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

### Arguments

| | |
|---|---|
| `x` | Object of class 'Trend' as returned from function `Trend` |
| `...` | further arguments (not used) |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# load a time series of Normalized Difference Vegetation Index
data(ndvi)
plot(ndvi)

# calculate a trend and look at the results
ndvi.trend <- Trend(ndvi)
ndvi.trend
print(ndvi.trend)
```

## ReadVI3g — *Read and pre-process GIMMS VI3g binary files*

### Description

This function reads GIMMS VI3g binary files, pre-processes the values (exclusion of flagged values, subset for area of interest) and returns the result as a raster layer. The function can be used to read the original GIMMS NDVI3g data files to R.

### Usage

```
ReadVI3g(file, flag = 2:7, ext = c(-180, 180, -90, 90))
```

### Arguments

| | |
|---|---|
| `file` | GIMMS VI3g file name |
| `flag` | vector of quality flags that should be excluded. Default: 2:7 (all values with reduced quality excluded). If you want to keep all values set `flag=NA`. |
| `ext` | extent (xmin, xmax, ymin, ymax) for which to extract the data |

### Details

The GIMMS NDVI3g dataset comes with the following quality flags:

- FLAG = 1 (Good value)
- FLAG = 2 (Good value, possibly snow)
- FLAG = 3 (NDVI retrieved from spline interpolation)
- FLAG = 4 (NDVI retrieved from spline interpolation, possibly snow)
- FLAG = 5 (NDVI retrieved from average seasonal profile)
- FLAG = 6 (NDVI retrieved from average seasonal profile, possibly snow)
- FLAG = 7 (missing data)

### Value

raster layer

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# data <- ReadVI3g("geo00oct15a.n14-VI3g")
# plot(data)
```

---

`Seasonality` *Check a time series for seasonality*

---

### Description

This function checks a time series for seasonality using three different approaches:

- `'pgram'` computes a periodogram using fast fourier transformation (`spec.pgram`) and checks at which frequency the periodogram has a maximum. A maximum at a frequency of 1 indicates seasonality and the function returns TRUE.
- `'acf'` computes the auto-correlation function of the de-trended time series using `acf`. A minimum acf value at a lag of 0.5 indicates seasonality and the function returns TRUE.
- `'lm'` fits two linear models to the time series. The first model includes the trend and the seasonal cycle as factorial variable. The second model includes only the trend. Based on the `BIC` the better model is selected and the function returns TRUE if the first model (including a seasonal term) is better.

### Usage

```
Seasonality(Yt, return.freq = FALSE, plot = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `return.freq` | if return.freq is TRUE the function returns the frequency at the maximum of the periodogram. |
| `plot` | plot periodogram and acf? (see `spec.pgram` and `acf`) |
| `...` | further arguments (currently not used) |

### Value

The function returns a boolean vector of length 3 including TRUE if a method detected seasonality or FALSE if a method did not detect seasonality.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`spec.pgram`, `acf`, `lm`, `BIC`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)

# time series with strong Seasonality:
Yt <- SimTs(Srange = 0.2, Tslope=c(0.0004, 0))[,1]
plot(Yt)
Seasonality(Yt)

# time series with Seasonality and some noise
Yt <- SimTs(Srange = 0.1, Tslope=c(0.0004, 0), Rsd=0.18, Rrange=0.25)[,1]
plot(Yt)
Seasonality(Yt)

# time series with Seasonality but many noise
Yt <- SimTs(Srange = 0.1, Tslope=c(0.0004, 0), Rsd=0.22, Rrange=0.4)[,1]
plot(Yt)
Seasonality(Yt)

# time series without Seasonality
Yt <- SimTs(Srange = 0.01, Tslope=c(0.0004, 0), Rsd=0.2, Rrange=0.4)[,1]
plot(Yt)
Seasonality(Yt)

# plot results for each seasonality method
Yt <- SimTs(Srange = 0.1, Tslope=c(0.0004, 0), Rsd=0.18, Rrange=0.25)[,1]
```

```
Seasonality(Yt, plot=TRUE)
```

---

SimIAV *Simulate the inter-annual variability component of a surrogate time series*

---

## Description

The function simulates the inter-annual variability component of a time series based on normal-distributed random values.

## Usage

```
SimIAV(sd = 0.015, range = sd * 2, nyears = 30, start = c(1982,
    1), freq = 12)
```

## Arguments

| | |
|---|---|
| sd | standard deviation of the annual mean values |
| range | range of the annual mean values |
| nyears | number of years |
| start | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| freq | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

## Value

time series of class `ts`

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`SimTs`

## Examples

```
It <- SimIAV(sd=0.015, range=0.05, nyears=30)
plot(It)
```

`SimRem` *Simulate the short-term variability component of a surrogate time series*

## Description

The function simulates the short-term variability component (remainder component) of a time series (remainder component) based on normal-distributed random values.

## Usage

```
SimRem(sd = 0.05, range = sd * 3, n = 360, start = c(1982, 1),
    freq = 12)
```

## Arguments

| | |
|---|---|
| `sd` | standard deviation of short-term anomalies |
| `range` | range of short-term anomalies |
| `n` | length of the time series |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

## Value

time series of class `ts`

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`SimTs`

## Examples

```
Rt <- SimRem(sd=0.02, range=0.08)
plot(Rt)
```

## SimSeas — *Simulate the seasonal component of a surrogate time series*

### Description

The function simulates the seasonal component of a time series based on a cosinus harmonic term.

### Usage

```
SimSeas(range, n = 360, start = c(1982, 1), freq = 12)
```

### Arguments

| | |
|---|---|
| `range` | range of the seasonal cycle (seasonal amplitude) |
| `n` | length of the time series |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

### Value

time series of class `ts`

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`SimTs`

### Examples

```
St <- SimSeas(range=0.6)
plot(St)
```

SimTrend *Simulate trend and breakpoints of a surrogate time series*

## Description

The function simulates the trend component with breakpoints of a time series.

## Usage

```
SimTrend(slope = c(0.002, -0.004), breaks = 165, abrupt = TRUE,
    n = 360, start = c(1982, 1), freq = 12)
```

## Arguments

| | |
|---|---|
| slope | slope of the trend in each time series segment. slope should be a numeric vector. The length of this vector determines the number of segments. |
| breaks | position of the breakpoints in the time series. You should specify one more slope than breakpoint. |
| abrupt | Should the trend at the breakpoints change abrupt (TRUE) or gradual (FALSE)? |
| n | length of the time series |
| start | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See ts for further examples. |
| freq | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See ts for further examples. |

## Value

time series of class ts

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

SimTs

## Examples

```
Tt <- SimTrend(slope=c(0.003, -0.001), breaks=150)
plot(Tt)
```

`SimTs` *Simulate surrogate time series*

## Description

The function simulates a surrogate (artificial) time series based on the defined properties. See Forkel et al. 2013 for a description how time series are simulated with this function.

## Usage

```
SimTs(M = 0.35, Tslope = c(0.002, -0.004), Isd = 0.015, Irange = 0.03,
    Srange = 0.5, Rsd = 0.05, Rrange = 0.1, breaks = 120, abrupt = TRUE,
    n = 360, start = c(1982, 1), freq = 12)
```

## Arguments

| | |
|---|---|
| `M` | mean of the time series |
| `Tslope` | slope of the trend in each time series segment. `slope` should be a numeric vector. The length of this vector determines the number of segements. |
| `Isd` | standard deviation of the annual mean values (inter-annual variability) |
| `Irange` | range of the annual mean values (inter-annual variability) |
| `Srange` | range of the seasonal cycle (seasonal amplitude) |
| `Rsd` | standard deviation of short-term anomalies |
| `Rrange` | range of short-term anomalies |
| `breaks` | position of the breakpoints in the time series. You should specify one more slope than breakpoint. |
| `abrupt` | Should the trend at the breakpoints change abrupt (`TRUE`) or gradual (`FALSE`)? |
| `n` | length of the time series |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

## Value

The function returns multiple time series of class `ts` with the following components: total time series, mean, trend component, inter-annual variability component, seasonal component, short-term component.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

### See Also

SimTs

### Examples

```
# simulate artificial time series
x <- SimTs(M=0.4, Tslope=0.0008, Isd=0.015, Irange=0.03, Srange=0.5, Rsd=0.05,
Rrange=0.1, breaks=NULL, abrupt=TRUE, n=360, start=c(1982, 1), freq=12)
plot(x)

x <- SimTs(M=0.35, Tslope=c(0.002, -0.0015), Isd=0.015, Irange=0.03, Srange=0.5,
   Rsd=0.05, Rrange=0.1, breaks=120, abrupt=TRUE, n=360, start=c(1982, 1), freq=12)
plot(x)

x <- SimTs(M=0.4, Tslope=c(0.003, -0.001, 0), Isd=0.03, Irange=0.08, Srange=0.3,
   Rsd=0.06,  Rrange=0.2, breaks=c(100, 210), abrupt=FALSE,
   n=360, start=c(1982, 1), freq=12)
plot(x)
```

---

## SplitRasterEqually *Splits a raster in equal-area parts*

---

### Description

This function splits a raster object in parts with ~ equal area.

### Usage

```
SplitRasterEqually(data.r, n)
```

### Arguments

| | |
|---|---|
| data.r | raster, raster brick or raster stack. |
| n | number of parts |

### Value

the function returns a list of raster layers

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# data(ndvimap)
# tiles.l <- SplitRasterEqually(ndvimap, n=4)
# tiles.l
```

---

## SSASeasonalCycle — *Calculate a modulated seasonal cycle of a time series based on singular spectrum analysis (SSA)*

---

### Description

The function calculates a seasonal cycle based on 1-D singular spectrum analysis (Golyandina et al. 2001) as implemented in the Rssa package. See `ssa` for details.

### Usage

```
SSASeasonalCycle(ts)
```

### Arguments

| | |
|---|---|
| `ts` | univariate time series of class `ts` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): Analysis of Time Series Structure: SSA and related techniques. Chapman and Hall/CRC. ISBN 1584881941

### See Also

`TrendSeasonalAdjusted`, `ssa`, `reconstruct`

### Examples

```
## load a time series of Normalized Difference Vegetation Index
#data(ndvi)
#plot(ndvi)

## estimate the seasonal cycle using SSA
#ndvi.cycle <- SSASeasonalCycle(ndvi)
#plot(ndvi.cycle)

## the mean seasonal cycle is centered to 0,
## add the mean of the time series if you want to overlay it with the original data
#plot(ndvi)
#lines(ndvi.cycle + mean(ndvi, na.rm=TRUE), col="blue")
```

`Trend` *Calculate trends and trend changes in time series*

## Description

This function calculates trends and trend changes (breakpoints) in a time series. It is a common interface to the functions `TrendAAT`, `TrendSTM` and `TrendSeasonalAdjusted`. With `TrendRaster` all trend analysis functions can be applied to gridded (raster) data. A detailed description of these methods can be found in Forkel et al. (2013).

## Usage

```
Trend(Yt, method = c("AAT", "STM", "SeasonalAdjusted"), mosum.pval = 0.05,
    h = 0.15, breaks = NULL, funSeasonalCycle = MeanSeasonalCycle,
    funAnnual = mean, sample.method = c("sample", "all", "none"),
    sample.min.length = 0.75, sample.size = 30)
```

## Arguments

`Yt` univariate time series of class `ts`

`method` method to be used for trend calculation with the following options:

- `AAT` (default) calculates trends on annual aggregated time series (see `TrendAAT` for details). This method will be automatically choosen if the time series has a frequency of 1 (e.g. in case of annual time steps). If the time series has a frequency > 1, the time series will be aggregated to annual time steps using the mean.
- `STM` fits harmonics to the seasonal time series to model the seasonal cycle and to calculate trends based on a multiple linear regression (see `TrendSTM` for details).
- `SeasonalAdjusted` removes first the seasonal cycle from the time series and calculates the trend on the reaminder series (see `TrendSeasonalAdjusted` for details).

`mosum.pval` Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details.

`h` minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details.

`breaks` maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details.

`funSeasonalCycle` a function to estimate the seasonal cycle of the time series if `SeasonalAdjusted` is selected as method. A own function can be defined to estimate the seasonal cycle which has to return the seasonal cycle as a time series of class `ts`. Currently two approaches are part of this package:

- `MeanSeasonalCycle` is the default which computes the mean seasonal cycle.
- `SSASeasonalCycle` detects a modulated seasonal cycle based on Singular Spectrum Analysis.

`funAnnual` function to aggregate time series to annual values if `AAT` is selected as method. The default function is the mean (i.e. trend calculated on mean annual time series). See `TrendAAT` for other examples

`sample.method`
Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates).

`sample.min.length`
Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation.

`sample.size` sample size (number of combinations of start and end dates) to be used if `method` is sample.

## Details

This function allows to calculate trends and trend changes based on different methods: see `TrendAAT`, `TrendSTM` or `TrendSeasonalAdjusted` for more details on these methods. These methods can be applied to gridded (raster) data using the function `TrendRaster`.

## Value

The function returns a list of class "Trend".

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

## See Also

`plot.Trend`, `TrendAAT`, `TrendSTM`, `TrendSeasonalAdjusted`, `TrendRaster`, `breakpoints`

## Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend (default method: trend calculated based on annual aggregated data)
trd <- Trend(ndvi)
trd
plot(trd)

# an important parameter is mosum.pval: if the p-value is changed to 1,
# breakpoints can be detected in the time series regardless if
# there is significant structural change
trd <- Trend(ndvi, mosum.pval=1)
trd
plot(trd)

# calculate trend based on modelling the seasonal cycle
trd <- Trend(ndvi, method="STM")
trd
plot(trd)

# calculate trend based on removal of the seasonal cycle
trd <- Trend(ndvi, method="SeasonalAdjusted", funSeasonalCycle=MeanSeasonalCycle)
plot(trd)
lines(trd$adjusted, col="green")
trd

# modify maximal number of breakpoints
trd <- Trend(ndvi, method="SeasonalAdjusted", breaks=1,
funSeasonalCycle=MeanSeasonalCycle, sample.method="sample")
plot(trd)
trd
```

---

`TrendAAT` *Trend estimation based on annual aggregated time series*

---

## Description

The function aggregates a time series to annual values and computes breakpoints and trends on the annual aggregated time series. The function can be applied to gridded (raster) data using the function `TrendRaster`. A detailed description of this method can be found in Forkel et al. (2013).

## Usage

```
TrendAAT(Yt, mosum.pval = 0.05, h = 0.15, breaks = NULL, funAnnual = mean,
    sample.method = c("sample", "all", "none"), sample.min.length = 0.75,
    sample.size = 30)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `mosum.pval` | Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details. |
| `h` | minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details. |
| `breaks` | maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details. |
| `funAnnual` | function to aggregate time series to annual values The default function is the mean (i.e. trend calculated on mean annual time series). See example section for other examples. |
| `sample.method` | Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates). |
| `sample.min.length` | Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation. |
| `sample.size` | sample size (number of combinations of start and end dates) to be used if `method` is sample. |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

### See Also

`Trend`, `TrendRaster`

## Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on mean annual NDVI values
trd.annualmean <- TrendAAT(ndvi)
trd.annualmean
plot(trd.annualmean)

# calculate annual trend but don't apply MOSUM test for structural change
trd.annualmean <- TrendAAT(ndvi, mosum.pval=1)
trd.annualmean
plot(trd.annualmean)

# calculate trend on annual peak (maximum) NDVI
trd.annualmax <- TrendAAT(ndvi, funAnnual=max, mosum.pval=1)
trd.annualmax
plot(trd.annualmax)

# calculate trend on an annual quantile NDVI (e.g. upper 0.9 quantile)
fun <- function(x, ...) { quantile(x, 0.9, ...) }
trd.annualquantile9 <- TrendAAT(ndvi, funAnnual=fun, mosum.pval=1)
trd.annualquantile9
plot(trd.annualquantile9)

# calculate trend on an winter NDVI (e.g. upper 0.1 quantile)
fun <- function(x, ...) { quantile(x, 0.1, ...) }
trd.annualquantile1 <- TrendAAT(ndvi, funAnnual=fun, mosum.pval=1)
trd.annualquantile1
plot(trd.annualquantile1)

# compare trends
plot(ndvi)
plot(trd.annualmean, add=TRUE, col="darkgreen", symbolic=TRUE)
plot(trd.annualmax, add=TRUE, col="red", symbolic=TRUE)
plot(trd.annualquantile9, add=TRUE, col="orange", symbolic=TRUE)
plot(trd.annualquantile1, add=TRUE, col="blue", symbolic=TRUE)
```

---

# `TrendClassification` *Classify a raster in greening and browning trends*

---

## Description

This function classifies a RasterBrick with trend estimates as computed with `TrendRaster` into positive, negative and no trend per each time series segment.

### Usage

```
TrendClassification(r, min.length = 0, max.pval = 0.05, ...)
```

### Arguments

| | |
|---|---|
| `r` | multi-layer raster object of class `brick` as computed with `TrendRaster` |
| `min.length` | Minimum duration of a trend in time steps of the input raster (see Details). |
| `max.pval` | Maximum p-value to classify a trend as being significant. |
| `...` | additional arguments as for `writeRaster` |

### Details

This function expects a RasterBrick as created with `TrendRaster` as input and classifies for each pixel and each time series segment if a trend is significant positive, significant negative or not significant (no trend). Per default a p-value of 0.05 is used to classify trends as significant. Additionally, the minimum duration of a trend can be specified with min.length: For example, if only time series segments longer than 10 years should be considered as trend, set min.length=11 in case of annual data. In case of monthly data set it to 132 (12 observations per year * 11 years). The function `CompareClassification` can be used to compare classified trends from different methods or data sets.

### Value

The function returns a RasterLayer in case of one time series segment or a RasterBrick in case of multiple time series segments. Pixels with a significant positive trend have the value 1; pixels with significant negative trends -1 and non-significant trends 0.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TrendRaster`, `CompareClassification`

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
ndvimap
plot(ndvimap, 8)

# calculate trends on the raster
trendmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=2)
plot(trendmap)

# classify the trends in greening and browning
greenbrownmap <- TrendClassification(trendmap, min.length=10, max.pval=0.05)
plot(greenbrownmap, col=brgr.colors(3))
```

| | |
|---|---|
| TrendGradient | *Calculate temporal trends along a spatial gradient* |

## Description

This function extracts along a spatial gradient (e.g. along latitude) time series from a raster brick and computes for each position a temporal trend.

## Usage

```
TrendGradient(r, start = c(1982, 1), freq = 12, gradient.r = NULL,
    gradient.brks = NULL, funSpatial = "mean", cor.area = FALSE,
    scalar = 1, method = c("AAT", "STM", "SeasonalAdjusted"),
    mosum.pval = 0.05, h = 0.15, breaks = 0, funAnnual = mean,
    funSeasonalCycle = MeanSeasonalCycle, percent = FALSE)
```

## Arguments

`r` multi-layer raster object of class `brick`

`start` beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples.

`freq` The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples.

`gradient.r` raster layer with the variable that has a spatial gradient. If NULL (default) a gradient along latitude will be used. Alternatively, one could provide here for example a raster layer with a gradient along longitude for longitudinal gradients of trends or a raster layer with mean annual temperatures to compute trends along a temperature gradient.

`gradient.brks` breaks for the gradient. These breaks define the class intervals for which time series will be extracted and trends computed. If NULL (default) 15 class breaks between the minimum and maximum values of the gradient will be used.

`funSpatial` function that should be used for spatial aggregation of grid cells that belong to the same interval.

`cor.area` If TRUE grid cell values are multiplied by grid cell area to correct for area.

`scalar` Multiplier to be applied to time series (e.g. for unit conversions).

`method` method to be used for trend calculation with the following options:

- `AAT` (default) calculates trends on annual aggregated time series (see `TrendAAT` for details). This method will be automatically choosen if the time series has a frequency of 1 (e.g. in case of annual time steps). If the time series has a frequency > 1, the time series will be aggregated to annual time steps using the mean.

- `STM` fits harmonics to the seasonal time series to model the seasonal cycle and to calculate trends based on a multiple linear regression (see `TrendSTM` for details).
- `SeasonalAdjusted` removes first the seasonal cycle from the time series and calculates the trend on the reaminder series (see `TrendSeasonalAdjusted` for details).

`mosum.pval` Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details.

`h` minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details.

`breaks` maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details.

`funAnnual` function to aggregate time series to annual values if `AAT` is selected as method. The default function is the mean (i.e. trend calculated on mean annual time series). See `TrendAAT` for other examples

`funSeasonalCycle`
a function to estimate the seasonal cycle of the time series if `SeasonalAdjusted` is selected as method. A own function can be defined to estimate the seasonal cycle which has to return the seasonal cycle as a time series of class `ts`. Currently two approaches are part of this package:

- `MeanSeasonalCycle` is the default which computes the mean seasonal cycle.
- `SSASeasonalCycle` detects a modulated seasonal cycle based on Singular Spectrum Analysis.

`percent` return trend as percentage change

## Details

The function returns a list of class 'TrendGradient'

## Value

The function returns a data.frame of class 'TrendGradient'.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`Trend`, `TrendRaster`

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# compute a latitudinal gradient of trends (by default the method 'AAT' is used):
gradient <- TrendGradient(ndvimap, start=c(1982, 1), freq=12)
gradient
plot(gradient)
# shown is the trend at each latitudinal band, the area represents the 95%
# confidence interval of the trend (computed with function TrendUncertainty),
# symbols indicate the p-value of the trend at each latitude

plot(gradient, type="yx") # the gradient can be also plotted in reversed order

# latitudinal gradient with different number of intervals:
gradient <- TrendGradient(ndvimap, start=c(1982, 1), freq=12,
   gradient.brks=seq(66, 69, length=5))
plot(gradient)

# example for a longitudinal gradient:
gradient.r <- raster(ndvimap, 1) # create a raster layer with longitudes:
gradient.r[] <- xFromCell(gradient.r, 1:ncell(gradient.r))
plot(gradient.r)
gradient <- TrendGradient(ndvimap, start=c(1982, 1), freq=12,
   gradient.r=gradient.r)
plot(gradient, xlab="Longitude (E)")
```

---

## `TrendLongestSEG` *Extract slope and p-value for the longest time series segment from a TrendRaster raster brick*

---

### Description

This function extracts the slope and p-value of a trend for the longest time series segment from a raster brick that was created with `TrendRaster`

### Usage

```
TrendLongestSEG(r)
```

### Arguments

| | |
|---|---|
| `r` | `RasterBrick` as created with `TrendRaster` or object of class 'Trend' as returned by `Trend` |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`TrendRaster`

## Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
plot(ndvimap, 8)

# calculate trend
trendmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=1)
plot(trendmap)

# select trend and p-value only for the longest time series segment
trendmap.longestseg <- TrendLongestSEG(trendmap)
plot(trendmap.longestseg)
```

---

`TrendNCDF` *Calculate trends and trend statistics on time series in gridded (raster) data stored in a NetCDF file*

---

## Description

This function computes temporal trend and trend breakpoints on multi-temporal raster data that is stored in a NetCDF file. To calculate trends on the values of each grid cell the function `Trend` is used. Before using these methods on satellite time series (especially NDVI time series) the descriptions and recommendations in Forkel et al. (2013) should be considered. The function applies the function `TrendRaster` on a NetCDF file and saves the results as NetCDF files. Additionally, several summary raster layers are saved as NetCDF files too. Thus, it can potentially simplify the workflow.

## Usage

```
TrendNCDF(file, start = c(1982, 1), freq = 12, method = "AAT",
    mosum.pval = 0.05, h = 0.15, breaks = 1, funSeasonalCycle = MeanSeasonalCycl
    funAnnual = mean, ...)
```

## Arguments

| | |
|---|---|
| `file` | NetCDF file with file extention *.nc |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |

`method` method to be used for trend calculation with the following options:

- `AAT` (default) calculates trends on annual aggregated time series (see `TrendAAT` for details). This method will be automatically choosen if the time series has a frequency of 1 (e.g. in case of annual time steps). If the time series has a frequency > 1, the time series will be aggregated to annual time steps using the mean.
- `STM` fits harmonics to the seasonal time series to model the seasonal cycle and to calculate trends based on a multiple linear regression (see `TrendSTM` for details).
- `SeasonalAdjusted` removes first the seasonal cycle from the time series and calculates the trend on the reaminder series (see `TrendSeasonalAdjusted` for details).

`mosum.pval` Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details.

`h` minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details.

`breaks` maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details.

`funSeasonalCycle`
a function to estimate the seasonal cycle of the time series if `SeasonalAdjusted` is selected as method. An own function can be defined to estimate the seasonal cycle which has to return the seasonal cycle as a time series of class `ts`. Currently two approaches are part of this package:

- `MeanSeasonalCycle` is the default which computes the mean seasonal cycle.
- `SSASeasonalCycle` detects a modulated seasonal cycle based on Singular Spectrum Analysis.

`funAnnual` function to aggregate time series to annual values if `AAT` is selected as method. The default function is the mean (i.e. trend calculated on mean annual time series). See `TrendAAT` for other examples

`...`

## Details

The maximum number of breakpoints should be specified in this function. If `breaks=0` no breakpoints will be computed. If `breaks=1` one breakpoint can be detected at maximum per grid cell. In this case the result will be reported for two time series segments (SEG1 before the breakpoint, SEG2 after the breakpoint). Some of the trend methods are very slow. Applying them on multitemporal raster datasets can take some time. Especially the methods that work on the full temporal resolution time series (STM and SeasonalAdjusted) are slower than the method AAT. Especially if breakpoints are computed the computations take longer. The computation of breakpoints can be suppressed by choosing breaks=0. For large rasters it is recommended to first split the raster dataset in several tiles and to compute the trends on each tile separately. The use of a high performance computing infrastructure it also advantageous. All methods work with missing observations (for example missing NDVI observation in winter months with snow cover). Missing observation have to be flagged with NA. All time steps have to be included in the RasterBrick for trend analysis. If complete time steps are missing, they need to be included as layers (filled with NA values) in the RasterBrick to form a continuous time series.

## Value

The function saves several NetCDF files in directory on disc. The files are created based on the filename of the input `file`:

- `file.Trend.nc` NetCDF file with result of trend and breakpoints detection (from `TrendRaster`)
- `file.Trend.Classif.nc` NetCDF file with classified trends in each time series segment (from `TrendClassification`)
- `file.Trend.BP.nc` NetCDF file with time of breakpoints
- `file.Trend.LongestSEG.nc` NetCDF file with slope and p-values of the longest time series segment (from `TrendLongestSEG`)
- `file.Trend.SlopeLongestSEG.nc` NetCDF file with slope of the longest time series segment (from `TrendLongestSEG`)
- `file.Trend.PvalLongestSEG.nc` NetCDF file with p-value of the longest time series segment (from `TrendLongestSEG`)
- `file.Trend.LongestSEGClassif.nc` NetCDF file with classified trend of the longest time series segment (i.e. `TrendClassification` applied on `TrendLongestSEG`)

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

## See Also

`TrendRaster`, `Trend`, `TrendClassification`, `TrendLongestSEG`, `TrendSegmentsRaster`, `NamesTrendRaster`

## TrendPoly — *Trend estimation based on a 4th order polynomial*

### Description

The function computes a trend based on a 4th order polynomial function.

### Usage

```
TrendPoly(Yt, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `...` | additional arguments (currently not used) |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`stl`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on mean annual NDVI values
trd <- TrendPoly(ndvi)
trd
plot(trd)
```

TrendRaster *Calculate trends on time series in gridded (raster) data*

## Description

This function computes temporal trend and trend breakpoints on multi-temporal raster data. To calculate trends on the values of each grid cell the function `Trend` is used. Before using these methods on satellite time series (especially NDVI time series) the descriptions and recommendations in Forkel et al. (2013) should be considered.

## Usage

```
TrendRaster(r, start = c(1982, 1), freq = 12, method = c("AAT",
    "STM", "SeasonalAdjusted"), mosum.pval = 0.05, h = 0.15,
    breaks = 0, funSeasonalCycle = MeanSeasonalCycle, funAnnual = mean,
    ...)
```

## Arguments

| | |
|---|---|
| `r` | multi-layer raster object of class `brick` |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |
| `method` | method to be used for trend calculation with the following options:<br>• `AAT` (default) calculates trends on annual aggregated time series (see `TrendAAT` for details). This method will be automatically choosen if the time series has a frequency of 1 (e.g. in case of annual time steps). If the time series has a frequency > 1, the time series will be aggregated to annual time steps using the mean.<br>• `STM` fits harmonics to the seasonal time series to model the seasonal cycle and to calculate trends based on a multiple linear regression (see `TrendSTM` for details).<br>• `SeasonalAdjusted` removes first the seasonal cycle from the time series and calculates the trend on the reaminder series (see `TrendSeasonalAdjusted` for details). |
| `mosum.pval` | Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details. |
| `h` | minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details. |

`breaks` maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details.

`funSeasonalCycle` a function to estimate the seasonal cycle of the time series if `SeasonalAdjusted` is selected as method. An own function can be defined to estimate the seasonal cycle which has to return the seasonal cycle as a time series of class `ts`. Currently two approaches are part of this package:

- `MeanSeasonalCycle` is the default which computes the mean seasonal cycle.
- `SSASeasonalCycle` detects a modulated seasonal cycle based on Singular Spectrum Analysis.

`funAnnual` function to aggregate time series to annual values if `AAT` is selected as method. The default function is the mean (i.e. trend calculated on mean annual time series). See `TrendAAT` for other examples

`...` additional arguments as for `writeRaster`

**Details**

The maximum number of breakpoints should be specified in this function. If `breaks=0` no breakpoints will be computed. If `breaks=1` one breakpoint can be detected at maximum per grid cell. In this case the result will be reported for two time series segments (SEG1 before the breakpoint, SEG2 after the breakpoint). Some of the trend methods are very slow. Applying them on multi-temporal raster datasets can take some time. Especially the methods that work on the full temporal resolution time series (STM and SeasonalAdjusted) are slower than the method AAT. Especially if breakpoints are computed the computations take longer. The computation of breakpoints can be suppressed by choosing breaks=0. For large rasters it is recommended to first split the raster dataset in several tiles and to compute the trends on each tile separately. The use of a high performance computing infrastructure it also advantageous. All methods work with missing observations (for example missing NDVI observation in winter months with snow cover). Missing observation have to be flagged with NA. All time steps have to be included in the RasterBrick for trend analysis. If complete time steps are missing, they need to be included as layers (filled with NA values) in the RasterBrick to form a continuous time series.

**Value**

The function returns a RasterBrick with different trend and breakpoint statistics. The layers are named:

- `LengthSEG` length of the time series segment
- `BP` date of the trend breakpoints
- `SlopeSEG` slope of the trend in each segment
- `PvalSEG` p-value of the trend in each segment

The choosen number of `breaks` will define the number of raster layers of the result.

**Author(s)**

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

## See Also

`Trend`, `TrendClassification`, `TrendSegmentsRaster`, `NamesTrendRaster`

## Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
ndvimap
plot(ndvimap, 8)

# calculate trend: annual aggregation method
AATmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=1)
plot(AATmap)

# trend on seasonal adjusted time series based, no breakpoints
MACmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12,
   method="SeasonalAdjusted", breaks=0, funSeasonalCycle=MeanSeasonalCycle)
plot(MACmap)

# trend based on season-trend model
STMmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="STM", breaks=0)
plot(STMmap)

# classify the results in greening/browning/no trend
MACmap.cl <- TrendClassification(MACmap, min.length=(8*12))
STMmap.cl <- TrendClassification(STMmap, min.length=(8*12))
par(mfrow=c(1,2)) # set the tiles of the plot
plot(MACmap.cl, col=brgr.colors(3), main="Method MAC")
plot(STMmap.cl, col=brgr.colors(3), main="Method STM")
```

---

# `TrendRunmed` *Trend estimation based on a running median*

---

## Description

The function computes a non-linear trend based a running median.

## Usage

```
TrendRunmed(Yt, k = NULL, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `k` | integer width of median window; must be odd. If NULL a window size of 20 years (i.e. frequency * 20) will be used. |
| `...` | additional arguments (currently not used) |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`stl`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on mean annual NDVI values
trd <- TrendRunmed(ndvi)
trd
plot(trd)
```

---

## `TrendSample` *Compute trend statistics by sampling a time series according to different start and end dates*

---

### Description

The function computes an ensemble of trend statistics (linear trend slope, Mann-Kendall tau and p-value) on a time series by sampling different start and end dates of the time series. This ensemble can be used to compute uncertainties in trend statistics. Results can be plotted using the function `plot.TrendSample`.

### Usage

```
TrendSample(Yt, sample.method = c("all", "sample", "none"), sample.min.length =
    sample.size = 30)
```

### Arguments

- `Yt` univariate time series of class `ts`
- `sample.method`
  Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates).
- `sample.min.length`
  Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation.
- `sample.size` sample size (number of combinations of start and end dates) to be used if `method` is sample.

### Value

The function returns a data.frame with the start date, end date and length of the sample from the time series and the correspondig Mann-Kendall tau, p-value, slope, intercept, and percentage slope of a linear trend.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`Trend`, `plot.TrendSample`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)

# calculate uncertainty of trend dependent on start and end dates
ndvi <- aggregate(ndvi, FUN=mean)
trd.ens <- TrendSample(ndvi)
trd.ens

# plot relations between start, end dates, length and trend statistics
plot(trd.ens)
```

---

`TrendSeasonalAdjusted` *Trend estimation based on seasonal-adjusted time series*

---

## Description

The function computes and substracts the seasonal cycle from a time series. Then a trend is estimated on the seasonal-adjusted time series. The function can be applied to gridded (raster) data using the function `TrendRaster`. A detailed description of this method can be found in Forkel et al. (2013).

## Usage

```
TrendSeasonalAdjusted(Yt, mosum.pval = 0.05, h = 0.15, breaks = NULL,
    funSeasonalCycle = MeanSeasonalCycle, sample.method = c("sample",
        "all", "none"), sample.min.length = 0.75, sample.size = 30)
```

## Arguments

`Yt` univariate time series of class `ts`

`mosum.pval` Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details.

`h` minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. See `breakpoints` for details.

`breaks` maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. See `breakpoints` for details.

`funSeasonalCycle`
a function to estimate the seasonal cycle of the time series. A own function can be defined to estimate the seasonal cycle which has to return the seasonal cycle as a time series of class "ts". Currently two approaches are part of this package:

- `MeanSeasonalCycle` is the default which computes the average seasonal cycle from all years.
- `SSASeasonalCycle` can be used which detects a modulated seasonal cycle based on Singular Spectrum Analysis.

`sample.method`
Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates).

`sample.min.length`
: Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation.

`sample.size`
: sample size (number of combinations of start and end dates) to be used if `method` is sample.

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Forkel, M., N. Carvalhais, J. Verbesselt, M. Mahecha, C. Neigh and M. Reichstein (2013): Trend Change Detection in NDVI Time Series: Effects of Inter-Annual Variability and Methodology. - Remote Sensing 5.

### See Also

`Trend`, `TrendRaster`, `MeanSeasonalCycle`, `SSASeasonalCycle`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on time series with removed mean seasonal cycle
MACtrend <- TrendSeasonalAdjusted(ndvi, funSeasonalCycle=MeanSeasonalCycle)
MACtrend
plot(MACtrend)

# plot the seasonal-adjusted time series
plot(ndvi)
lines(MACtrend$adjusted, col="orange")

# calculate trend on time series with removed mean seasonal cycle
# but with limited number of breakpoints
MACtrend <- TrendSeasonalAdjusted(ndvi, breaks=1, funSeasonalCycle=MeanSeasonalCycle)
plot(MACtrend)

## calculate trend on time series with removed seasonal cycle but seasonal cycle computed
## on singular spectrum analysis
#SSAtrend <- TrendSeasonalAdjusted(ndvi, funSeasonalCycle=SSASeasonalCycle)
#SSAtrend
#plot(SSAtrend)
#lines(SSAtrend$adjusted, col="orange")
```

## TrendSegmentsRaster

*Identify for each multi-temporal raster layer the number of the trend segment*

### Description

Imagine you have a multi-temporal raster brick with 30 years of data. Now you compute trends using the function `TrendRaster`, which will return the timing of breakpoints as well as the slopes and p-values in each trend segment. But now you want to know for each pixel and each time step if it belongs to the first, second or Nth trend segment. For this you can use this function!

### Usage

```
TrendSegmentsRaster(trend.rb, start = c(1982, 1), end = c(2011,
    12), freq = 12, min.length = 0, max.pval = 0.05, ...)
```

### Arguments

| | |
|---|---|
| `trend.rb` | multi-layer raster object of class `brick` as computed with `TrendRaster` |
| `start` | beginning of the time series (i.e. the time of the first observation). The default is c(1982, 1), i.e. January 1982 which is the usual start date to compute trends on long-term series of satellite observations of NDVI. See `ts` for further examples. |
| `end` | end of the time series (i.e. the time of the last observation). The default is c(2008, 12), i.e. December 2008 as the last observation |
| `freq` | The frequency of observations. The default is 12 for monthly observations. Use 24 for bi-monthly observations, 365 for daily observations or 1 for annual observations. See `ts` for further examples. |
| `min.length` | Minimum duration of a trend in time steps of the input raster (see Details). |
| `max.pval` | Maximum p-value to classify a trend as being significant. |
| `...` | additional arguments as for `writeRaster` |

### Details

This function expects a RasterBrick as created with `TrendRaster` as input and assigns for each pixel and each time step the number of the trend segment. If a trend is not significant too short the time step will be flagged with NA. Per default a p-value of 0.05 is used to classify trends as significant. Additionally, the minimum duration of a trend can be specified with min.length: For example, if only time series segments longer than 10 years should be considered as trend, set min.length=11 in case of annual data. In case of monthly data set it to 132 (12 observations per year * 11 years)

### Value

The function returns a RasterBrick.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### Examples

```
# load a multi-temporal raster dataset of Normalized Difference Vegetation Index
data(ndvimap)
ndvimap
plot(ndvimap, 8)

# calculate trend
trendmap <- TrendRaster(ndvimap, start=c(1982, 1), freq=12, method="AAT", breaks=2)
plot(trendmap)

# indicate for each time step the trend segment number
trendsegmentsmap <- TrendSegmentsRaster(trendmap, min.length=5, max.pval=0.05,
start=c(1982, 1), end=c(2011, 1), freq=1)
plot(trendsegmentsmap, 1:2, col=c("blue", "red"))
# first 2 years: everthing belongs to time series segment 1
plot(trendsegmentsmap, 29:30, col=c("blue", "red"))
# last 2 years: most pixel belong still to first time series segment
# (i.e. no breakpoints were detected), but some pixels are in the second
# time series segment (i.e. after the first breakpoint)
```

---

## `TrendSpline` *Trend estimation based on a smoothing splines*

---

### Description

The function computes a non-linear trend based on `smooth.spline`.

### Usage

```
TrendSpline(Yt, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `...` | additional arguments (currently not used) |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`stl`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on mean annual NDVI values
trd <- TrendSpline(ndvi)
trd
plot(trd)
```

---

`TrendSSA` *Trend estimation based on SSA (singluar spectrum analysis)*

---

### Description

The function computes a non-linear trend based on `ssa`. Please note: Use the function `TrendSeasonalAdjusted` with the option funSeasonalCycle=SSASeasonalCylce to compute a linear trend with breakpoint detection based on a seasonal adjusted time series (method "SSA" as desribed in Forkel et al. 2013).

### Usage

```
TrendSSA(Yt, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `...` | additional arguments (currently not used) |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`ssa`

### Examples

```
## load a time series of NDVI (normalized difference vegetation index)
#data(ndvi)
#plot(ndvi)
#
## calculate trend on mean annual NDVI values
#trd <- TrendSSA(ndvi)
#trd
#plot(trd)
```

---

## TrendSTL — *Trend estimation based on STL (Seasonal Decomposition of Time Series by Loess)*

---

### Description

The function computes a non-linear trend based on `stl`.

### Usage

```
TrendSTL(Yt, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `...` | additional arguments (currently not used) |

### Value

The function returns a list of class "Trend".

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`stl`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend on mean annual NDVI values
trd <- TrendSTL(ndvi)
```

```
trd
plot(trd)
```

---

TrendSTM *Trend estimation based on a season-trend model*

---

## Description

The trend and breakpoint estimation in method STM is based on the classical additive decomposition model and is following the implementation as in the `bfast` approach (Verbesselt et al. 2010, 2012). Linear and harmonic terms are fitted to the original time series using ordinary least squares regression. This method can be also used to detect breakpoints in the seasonal component of a time series. The function can be applied to gridded (raster) data using the function `TrendRaster`.

## Usage

```
TrendSTM(Yt, h = 0.15, breaks = NULL, mosum.pval = 0.05)
```

## Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `h` | minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment. |
| `breaks` | maximal number of breaks to be calculated (integer number). By default the maximal number allowed by h is used. |
| `mosum.pval` | Maximum p-value for the OLS-MOSUM test in order to search for breakpoints. If p = 0.05, breakpoints will be only searched in the time series trend component if the OLS-MOSUM test indicates a significant structural change in the time series. If p = 1 breakpoints will be always searched regardless if there is a significant structural change in the time series or not. See `sctest` for details. |

## Value

The function returns a list of class "Trend".

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## References

Verbesselt, J.; Hyndman, R.; Zeileis, A.; Culvenor, D., Phenological change detection while accounting for abrupt and gradual trends in satellite image time series. Remote Sensing of Environment 2010, 114, 2970-2980.

Verbesselt, J.; Zeileis, A.; Herold, M., Near real-time disturbance detection using satellite image time series. Remote Sensing of Environment 2012, 123, 98-108.

### See Also

`Trend`, `TrendRaster`, `TSGFstm`,

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# calculate trend
trd <- TrendSTM(ndvi)
trd
plot(trd)

# plot the fitted season-trend model
plot(ndvi)
lines(trd$fit, col="red")
```

---

## `TrendUncertainty` *Compute uncertainties in trend statistics according to different start and end dates*

---

### Description

The function computes trend statistics (linear trend slope and intercept, Mann-Kendall tau and p-value) with associated uncertainties (standard deviation) by sampling the time series according to different start and end dates using the function `TrendSample`

### Usage

```
TrendUncertainty(Yt, bp = NoBP(), sample.method = c("sample",
    "all", "none"), sample.min.length = 0.75, sample.size = 30,
    fun.unc = NULL)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts` |
| `bp` | detected breakpoints in the time series as returned by `breakpoints` |
| `sample.method` | Sampling method for combinations of start and end dates to compute uncertainties in trends. If "sample" (default), trend statistics are computed for a sample of combinations of start and end dates according to `sample.size`. If "all", trend statistics are computed for all combinations of start and end dates longer than `sample.min.length`. If "none", trend statistics will be only computed for the entire time series (i.e. no sampling of different start and end dates). |

`sample.min.length`
: Minimum length of the time series (as a fraction of total length) that should be used to compute trend statistics. Time windows between start and end that are shorter than min.length will be not used for trend computation.

`sample.size`
: sample size (number of combinations of start and end dates) to be used if `method` is sample.

`fun.unc`
: function to summarize the uncertainty of the trend (default: quantile 0.05 and 0.95). Can be also 'sd' or other functions.

## Value

The function returns a data.frame with the estimated Mann-Kendall tau, p-value and slope and intercept of a linear trend with uncertainties defined as the standard deviation of these estimates dependent on different start and end dates.

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`Trend`

## Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)

# aggregate time series to annual time steps
ndvi <- aggregate(ndvi, FUN=mean)
plot(ndvi)

# compute trend statistics dependent on start and end of the time series
trd.ens <- TrendSample(ndvi)
plot(trd.ens)

# compute statistics for trend
TrendUncertainty(ndvi)

# compute trend statistics with uncertainties by considering breakpoints
bp <- breakpoints(ndvi ~ time(ndvi))
trd.unc <- TrendUncertainty(ndvi, bp)
trd.unc
trd.unc[[1]]$slope_unc
```

## TSGFdoublelog — *Temporal smoothing and gap filling using double logisitic functions*

### Description

This function fills gaps and smoothes a time series by fitting for each year a double logisitic function. Two definitions for the shape of the double logistic function are available: 'Elmore' fits a function according to Elmore et al. (2012) and 'Beck' fits a according to Beck et al. (2006). If the time series has no Seasonality, double logistic fitting will not be performed but smoothing and interpolation will be done according to the selected `backup` algorithm.

### Usage

```
TSGFdoublelog(Yt, interpolate = FALSE, method = c("Elmore", "Beck"),
    backup = NULL, check.seasonality = 1:3, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values by using the logistic fit function? |
| `method` | Which kind of double logistic curve should be used? 'Elmore' (Elmore et al. 2012) or 'Beck' (Beck et al. 2006). |
| `backup` | Which backup algorithm should be used for temporal smoothing and gap filling if the time series has no seasonality? If a time series has no seasonal pattern, the fitting of double logistic functions is not meaningful. In this case another method can be used. Default: NULL (returns NA - no smoothing), other options: "TSGFspline", "TSGFssa", "TSGFlinear" |
| `check.seasonality` | Which methods in `Seasonality` should indicate TRUE (i.e. time series has seasonality) in order to calculate phenology metrics? 1:3 = all methods should indicate seasonality, Set to NULL in order to not perform seasonality checks. |
| `...` | further arguments (currently not used) |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Beck, P.S.A., C. Atzberger, K.A. Hodga, B. Johansen, A. Skidmore (2006): Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. - Remote Sensing of Environment 100:321-334.
Elmore, A.J., S.M. Guinn, B.J. Minsley and A.D. Richardson (2012): Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. - Global Change Biology 18, 656-674.

### See Also

`FitDoubleLogBeck`, `FitDoubleLogElmore`, `TsPP`, `Phenology`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce random gaps
gaps <- ndvi
gaps[runif(100, 1, length(ndvi))] <- NA
plot(gaps)

# do smoothing and gap filling
tsgf1 <- TSGFdoublelog(gaps, method="Elmore")
tsgf2 <- TSGFdoublelog(gaps, method="Beck")
plot(gaps)
lines(tsgf1, col="red")
lines(tsgf2, col="blue")

# compare original data with gap-filled data
cols <- c("red", "blue")
all <- ts.union(ndvi, tsgf1, tsgf2)
all[!is.na(gaps),] <- NA
plot(all[,1], all[,2], col=cols[1], xlab="original", ylab="gap filled")
points(all[,1], all[,3], col=cols[2])
abline(0,1)
r <- c(cor(all[,1], all[,2], use="pairwise.complete.obs"),
cor(all[,1], all[,3], use="pairwise.complete.obs"))
lgd <- paste(c("Elmore Cor =", "Beck Cor ="), round(r, 3))
legend("topleft", lgd, text.col=cols)
```

---

## `TSGFlinear` *Temporal smoothing and gap filling using linear interpolation*

---

### Description

This function fills gaps in a time series by using linear interpolation `na.approx` and smoothes the time series by using running median window of size 3 `runmed`

### Usage

```
TSGFlinear(Yt, interpolate = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values by using `approx`? |
| `...` | further arguments (currently not used) |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TsPP`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce random gaps
gaps <- ndvi
gaps[runif(100, 1, length(ndvi))] <- NA
plot(gaps)

# do smoothing and gap filling
tsgf <- TSGFlinear(gaps)
plot(gaps)
lines(tsgf, col="red")

# compare original data with gap-filled data
plot(ndvi[is.na(gaps)], tsgf[is.na(gaps)], xlab="original", ylab="gap filled")
abline(0,1)
r <- cor(ndvi[is.na(gaps)], tsgf[is.na(gaps)])
legend("topleft", paste("Cor =", round(r, 3)))
```

## TSGFphenopix — *Temporal smoothing and gap filling using phenopix*

### Description

Time series smoothing and gap filling using fitting methods as provided in the `greenProcess` function of the `phenopix` package. Function fits are performed for each year separately for which `PhenopixMY` is used.

### Usage

```
TSGFphenopix(Yt, interpolate = FALSE, fit = "spline", ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values? |
| `fit` | fitting function to be applied, available options are: spline, beck, elmore, klosterman, gu (see `greenProcess`) |
| `...` | further arguments (currently not used) |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`PhenopixMY`, `TsPP`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce random gaps
gaps <- ndvi
gaps[runif(100, 1, length(ndvi))] <- NA
plot(gaps)

# do smoothing and gap filling
tsgf <- TSGFphenopix(gaps, fit="spline")
plot(gaps)
lines(tsgf, col="red")
```

```
# compare original data with gap-filled data
plot(ndvi[is.na(gaps)], tsgf[is.na(gaps)], xlab="original", ylab="gap filled")
abline(0,1)
r <- cor(ndvi[is.na(gaps)], tsgf[is.na(gaps)])
legend("topleft", paste("Cor =", round(r, 3)))

# compare spline from phenopix with TSGFspline
spl <- TSGFspline(gaps)
plot(gaps)
lines(tsgf, col="red")
lines(spl, col="blue")
legend("topleft", c("TSGFphenopix.spline", "TSGFspline"), text.col=c("red", "blue"))
# Note that the differences originate from the fact that TSGFspline is applied on
# the full time series whereas spline within phenopix is applied for each year
# separetely. Yearly fits for TSGFphenopix.spline are afterwards combined to a full
# time series. This can cause jumps or peaks between two years. Thus, TSGFspline is
# the better choice for multi-year time series. This is also seen in cross-validation:
plot(ndvi[is.na(gaps)], tsgf[is.na(gaps)], xlab="original", ylab="gap filled", col="red")
points(ndvi[is.na(gaps)], spl[is.na(gaps)], col="blue")
abline(0,1)
r <- cor(cbind(ndvi[is.na(gaps)], tsgf[is.na(gaps)], spl[is.na(gaps)]))
lgd <- paste(c("TSGFphenopix.spline", "TSGFspline"), "Cor =", round(r[1,2:3], 3))
legend("topleft", lgd, text.col=c("red", "blue"))

# Other fits wihtin phenopix might be usefull but are rather computationally expensive:
tsgf <- TSGFphenopix(gaps, fit="klosterman")
plot(gaps)
lines(tsgf, col="red")
```

---

`TSGFspline` *Temporal smoothing and gap filling using splines*

---

## Description

This function fills gaps in a time series by using `na.spline` and smoothes the time series by using `smooth.spline`

## Usage

```
TSGFspline(Yt, interpolate = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values by using `na.spline`? |
| `...` | further arguments (currently not used) |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

TsPP

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce random gaps
gaps <- ndvi
gaps[runif(100, 1, length(ndvi))] <- NA
plot(gaps)

# do smoothing and gap filling
tsgf <- TSGFspline(gaps)
plot(gaps)
lines(tsgf, col="red")

# compare original data with gap-filled data
plot(ndvi[is.na(gaps)], tsgf[is.na(gaps)], xlab="original", ylab="gap filled")
abline(0,1)
r <- cor(ndvi[is.na(gaps)], tsgf[is.na(gaps)])
legend("topleft", paste("Cor =", round(r, 3)))
```

---

## TSGFssa — *Temporal smoothing and gap filling using singular spectrum analysis*

---

### Description

This function fills gaps and smoothes a time series by using 1-dimensional singular spectrum analysis.

### Usage

```
TSGFssa(Yt, interpolate = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values? |
| `...` | further arguments (currently not used) |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### See Also

`TsPP`

### Examples

```
## load a time series of NDVI (normalized difference vegetation index)
#data(ndvi)
#plot(ndvi)

## introduce random gaps
#gaps <- ndvi
#gaps[runif(100, 1, length(ndvi))] <- NA
#plot(gaps)

## do smoothing and gap filling
#tsgf <- TSGFssa(gaps)
#plot(gaps)
#lines(tsgf, col="red")

## compare original data with gap-filled data
#plot(ndvi[is.na(gaps)], window(tsgf[is.na(gaps)], end=c(2008, 11)),
# xlab="original", ylab="gap filled")
#abline(0,1)
#r <- cor(ndvi[is.na(gaps)], tsgf[is.na(gaps)])
#legend("topleft", paste("Cor =", round(r, 3)))
```

---

## `TSGFstm` *Temporal smoothing and gap filling based on a season-trend model*

---

### Description

This function fills gaps in a time series by using a season-trend model as in `TrendSTM` (Verbesselt et al. 2010, 2012).

### Usage

```
TSGFstm(Yt, interpolate = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values by using `na.spline`? |
| `...` | further arguments to `TrendSTM`. |

### Value

The function returns a gap-filled and smoothed version of the time series.

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

### References

Verbesselt, J.; Hyndman, R.; Zeileis, A.; Culvenor, D., Phenological change detection while accounting for abrupt and gradual trends in satellite image time series. Remote Sensing of Environment 2010, 114, 2970-2980.
Verbesselt, J.; Zeileis, A.; Herold, M., Near real-time disturbance detection using satellite image time series. Remote Sensing of Environment 2012, 123, 98-108.

### See Also

`TsPP`, `TrendSTM`

### Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce random gaps
gaps <- ndvi
gaps[runif(100, 1, length(ndvi))] <- NA
plot(gaps)

# do smoothing and gap filling
tsgf <- TSGFstm(gaps)
plot(gaps)
lines(tsgf, col="red")

# compare original data with gap-filled data
plot(ndvi[is.na(gaps)], tsgf[is.na(gaps)], xlab="original", ylab="gap filled")
abline(0,1)
r <- cor(ndvi[is.na(gaps)], tsgf[is.na(gaps)])
```

```
legend("topleft", paste("Cor =", round(r, 3)))
```

---

TsPP *Pre-processing of time series*

---

## Description

This function can be used for pre-processing of time series before the analyzing phenology or trends. The pre-processing involves the following steps:

- Step 1. Filling of permanent gaps. Values that are missing in each year will be filled using the function `FillPermanentGaps`.
- Step 2. Temporal smoothing, gap filling and interpolation. The time series will be smoothed and remaining gaps will be filled. Optionally, time series will be interpolated to daily values.

## Usage

```
TsPP(Yt, fpg = FillPermanentGaps, tsgf = TSGFspline, interpolate = FALSE,
    min.gapfrac = 0.2, lower = TRUE, fillval = NA, fun = min,
    backup = NULL, check.seasonality = 1:3, ...)
```

## Arguments

| | |
|---|---|
| `Yt` | univariate time series of class `ts`. |
| `fpg` | Filling of permanent gaps: If NULL, permanent gaps will be not filled, else the function `FillPermanentGaps` will be applied. |
| `tsgf` | Temporal smoothing and gap filling: Function to be used for temporal smoothing, gap filling and interpolation of the time series. If NULL, this step will be not applied. Otherwise a function needs to be specified. Exisiting functions that can be applied are `TSGFspline`, `TSGFssa`, `TSGFdoublelog` |
| `interpolate` | Should the smoothed and gap filled time series be interpolated to daily values? |
| `min.gapfrac` | How often has an observation to be NA to be considered as a permanent gap? (fraction of time series length) Example: If the month January is 5 times NA in a 10 year time series (= 0.5), then the month January is considered as permanent gap if min.gapfrac = 0.4. |
| `lower` | For filling of permanent gaps: fill lower gaps (TRUE), upper gaps (FALSE) or lower and upper gaps (NULL) |
| `fillval` | For filling of permanent gaps: constant fill values for gaps. If NA the fill value will be estimated from the data using fun. |
| `fun` | For filling of permanent gaps: function to be used to compute fill values. By default, minimum. |

| | |
|---|---|
| `backup` | Which backup algorithm should be used instead of TSGFdoublelog for temporal smoothing and gap filling if the time series has no seasonality? If a time series has no seasonal pattern, the fitting of double logistic functions is not meaningful. In this case another method can be used. Default: NULL (returns NA - no smoothing), other options: "TSGFspline", "TSGFssa", "TSGFlinear" |
| `check.seasonality` | Which methods in `Seasonality` should indicate TRUE (i.e. time series has seasonality) in order to calculate phenology metrics? 1:3 = all methods should indicate seasonality, Set to NULL in order to not perform seasonality checks. |
| `...` | further arguments (currently not used) |

## Value

pre-processed time series

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

## See Also

`FillPermanentGaps`

## Examples

```
# load a time series of NDVI (normalized difference vegetation index)
data(ndvi)
plot(ndvi)

# introduce systematic gaps in winter and random gaps
gaps <- ndvi
gaps[runif(50, 1, length(ndvi))] <- NA
gaps[cycle(ndvi) == 1 | cycle(ndvi) == 2 | cycle(ndvi) == 12] <- NA
plot(gaps)

# perform pre-processing of time series using different methods
pp.lin <- TsPP(gaps, tsgf=TSGFlinear) # linear interpolation + running median
pp.spl <- TsPP(gaps, tsgf=TSGFspline) # smoothing splines
pp.beck <- TsPP(gaps, tsgf=TSGFdoublelog, method="Beck") # Beck et al. (2006)
pp.elmore <- TsPP(gaps, tsgf=TSGFdoublelog, method="Elmore") # Elmore et al. (2012)

plot(gaps)
cols <- rainbow(5)
lines(pp.lin, col=cols[1])
lines(pp.spl, col=cols[2])
lines(pp.beck, col=cols[3])
lines(pp.elmore, col=cols[4])

data.df <- ts.union(time(gaps), orig=ndvi, pp.lin, pp.spl, pp.beck, pp.elmore)
plot(data.df)
cor(na.omit(data.df[is.na(gaps),]))
```

WriteNCDF *Write raster objects to NetCDF files*

## Description

This function writes raster layers to NetCDF files including meta information as variable names and units and time axes.

## Usage

```
WriteNCDF(data = NULL, var.name, var.unit, var.longname = "",
    file = NULL, time = NULL, layernames = NULL, data.name = NA,
    region.name = NA, file.title = var.longname, file.description = NULL,
    reference = "", provider = "", creator = "greenbrown R package",
    naflag = -9999, scale = 1, offset = 0, overwrite = FALSE)
```

## Arguments

| | |
|---|---|
| `data` | raster layer or raster brick |
| `var.name` | variable name |
| `var.unit` | variable unit |
| `var.longname` | variable long name |
| `file` | file name. If NULL the file name will be created from the variable name and the dimensions of the data. |
| `time` | vector of time steps for each layer. |
| `layernames` | layer names if layers are not time steps. |
| `data.name` | name of the dataset |
| `region.name` | name of the region |
| `file.title` | title of the file |
| `file.description` | description of the file |
| `reference` | reference for the dataset |
| `provider` | dataset provider |
| `creator` | dataset creator |
| `naflag` | flag for NA values |
| `scale` | sclaing values for the data |
| `offset` | offset value |
| `overwrite` | overwrite existing file? |

## Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]

`WriteNCDF4` *Write NetCDF files*

### Description

Writes NetCDF files from rasters and makes sure that meta-information is properly defined.

### Usage

```
WriteNCDF4(data.l, var.name, var.unit, time = as.Date("2000-01-01"),
    var.description = var.name, file = NULL, data.name = NA,
    region.name = NA, file.title = var.name, file.description = var.name,
    reference = "", provider = "", creator = "", missval = -9999,
    scale = 1, offset = 0, compression = 9, overwrite = FALSE)
```

### Arguments

| | |
|---|---|
| `data.l` | a single Raster* object or a list of Raster* objects |
| `var.name` | vector of variable names |
| `var.unit` | vector of variable units |
| `time` | vector of time steps for each layer. |
| `var.description` | vector of variable descriptions |
| `file` | file name. If NULL the file name will be created from the variable name and the dimensions of the data. |
| `data.name` | name of the dataset |
| `region.name` | name of the region |
| `file.title` | title of the file |
| `file.description` | description of the file |
| `reference` | reference for the dataset |
| `provider` | dataset provider |
| `creator` | dataset creator |
| `missval` | flag for missing/NA values |
| `scale` | scaling values for the data |
| `offset` | offset value |
| `compression` | If set to an integer between 1 (least compression) and 9 (most compression), this enables compression for the variable as it is written to the file. Turning compression on forces the created file to be in netcdf version 4 format, which will not be compatible with older software that only reads netcdf version 3 files. |
| `overwrite` | overwrite existing file? |

### Author(s)

Matthias Forkel <matthias.forkel@geo.tuwien.ac.at> [aut, cre]