Package ‘phenofit’

January 23, 2024

Type Package

Title Extract Remote Sensing Vegetation Phenology

Version 0.3.9

Description The merits of 'TIMESAT' and 'phenopix' are adopted. Besides, a simple and growing season dividing method and a practical snow elimination method based on Whittaker were proposed. 7 curve fitting methods and 4 phenology extraction methods were provided. Parameters boundary are considered for every curve fitting methods according to their ecological meaning.
And 'optimx' is used to select best optimization method for different curve fitting methods.
Reference:
Kong, D., (2020). R package: A state-of-the-art Vegetation Phenology extraction package, phenofit version 0.3.1, <doi:10.5281/zenodo.5150204>;
Kong, D., Zhang, Y., Wang, D., Chen, J., & Gu, X. (2020). Photoperiod Explains the Asynchronization Between Vegetation Carbon Phenology and Vegetation Greenness Phenology. Journal of Geophysical Research: Biogeosciences, 125(8), e2020JG005636. <doi:10.1029/2020JG005636>;
Kong, D., Zhang, Y., Gu, X., & Wang, D. (2019). A robust method for reconstructing global MODIS EVI time series on the Google Earth Engine.
ISPRS Journal of Photogrammetry and Remote Sensing, 155, 13–24;
Zhang, Q., Kong, D., Shi, P., Singh, V.P., Sun, P., 2018. Vegetation phenology on the Qinghai-Tibetan Plateau and its response to climate change (1982–2013).
Agric. For. Meteorol. 248, 408–417. <doi:10.1016/j.agrformet.2017.10.026>.

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.1)

Imports Rcpp, purrr, dplyr (>= 1.1.0), stringr, magrittr, lubridate, data.table, zoo, gridExtra, ggplot2, optimx, ucminf, numDeriv, methods, zeallot

Suggests knitr, rmarkdown, testthat (>= 2.1.0)

URL https://github.com/eco-hydro/phenofit

BugReports https://github.com/eco-hydro/phenofit/issues

VignetteBuilder knitr

NeedsCompilation yes

Author Dongdong Kong [aut, cre],
Mingzhong Xiao [aut],
Yongqiang Zhang [aut],
Xihui Gu [aut],
Jianjian Cui [aut]

Maintainer Dongdong Kong <kongdd.sysu@gmail.com>

Repository CRAN

Date/Publication 2024-01-23 06:50:02 UTC

R topics documented:

`CA_NS6` *MOD13A1 EVI observations at flux site CA-NS6*

Description

Variables in `CA-NS6`:

- `site`: site name
- `y`: EVI
- `date`: date of image
- `t`: date of compositing image
- `w`: weights of data point
- `QC_flag`: QC flag of y, in the range of `c("snow", "cloud", "shadow", "aerosol", "marginal", "good")`

Usage

```
data('CA_NS6')
```

Format

An object of class `data.table` (inherits from `data.frame`) with 161 rows and 6 columns.

`check_input` *check_input*

Description

Check input data, interpolate NA values in y, remove spike values, and set weights for NA in y and w.

Usage

```
check_input(
  t,
  y,
  w,
  QC_flag,
  nptperyear,
  south = FALSE,
  wmin = 0.2,
  wsnow = 0.8,
  ymin,
  missval,
  maxgap,
  alpha = 0.02,
  alpha_high = NULL,
  date_start = NULL,
  date_end = NULL,
  mask_spike = TRUE,
  na.rm = FALSE,
  ...
)
```

Arguments

`t`	Numeric vector, `Date` variable
`y`	Numeric vector, vegetation index time-series
`w`	(optional) Numeric vector, weights of `y`. If not specified, weights of all `NA` values will be `wmin`, the others will be 1.0.
`QC_flag`	Factor (optional) returned by `qcFUN`, levels should be in the range of `c("snow", "cloud", "shadow", "aerosol", "marginal", "good")`, others will be categoried into `others`. `QC_flag` is used for visualization in `get_pheno()` and `plot_curvefits()`.
`nptperyear`	Integer, number of images per year.
`south`	Boolean. In south hemisphere, growing year is 1 July to the following year 31 June; In north hemisphere, growing year is 1 Jan to 31 Dec.
`wmin`	Double, minimum weight of bad points, which could be smaller the weight of snow, ice and cloud.
`wsnow`	Doulbe. Reset the weight of snow points, after get `ylu`. Snow flag is an important flag of ending of growing season. Snow points is more valuable than marginal points. Hence, the weight of snow should be great than that of `marginal`.
`ymin`	If specified, `ylu[1]` is constrained greater than ymin. This value is critical for bare, snow/ice land, where vegetation amplitude is quite small. Generally, you can set ymin=0.08 for NDVI, ymin=0.05 for EVI, ymin=0.5 gC m-2 s-1 for GPP.
`missval`	Double, which is used to replace NA values in y. If missing, the default vlaue is `ylu[1]`.

`maxgap` Integer, nptperyear/4 will be a suitable value. If continuous missing value numbers less than maxgap, then interpolate those NA values by zoo::na.approx; If false, then replace those NA values with a constant value `ylu[1]`.
Replacing NA values with a constant missing value (e.g. background value ymin) is inappropriate for middle growing season points. Interpolating all values by na.approx, it is unsuitable for large number continous missing segments, e.g. in the start or end of growing season.

`alpha` Double, in `[0,1]`, quantile prob of ylu_min.

`alpha_high` Double, `[0,1]`, quantile prob of `ylu_max`. If not specified, `alpha_high=alpha`.

`date_start, date_end`
starting and ending date of the original vegetation time-sereis (before `add_HeadTail`)

`mask_spike` Boolean. Whether to remove spike values?

`na.rm` Boolean. If `TRUE`, NA and spike values will be removed; otherwise, NA and spike values will be interpolated by valid neighbours.

`...` Others will be ignored.

Value

A list object returned:

- `t` : Numeric vector
- `y0`: Numeric vector, original vegetation time-series.
- `y` : Numeric vector, checked vegetation time-series, `NA` values are interpolated.
- `w` : Numeric vector
- `Tn`: Numeric vector
- `ylu`: = `[ymin, ymax]`. `w_critical` is used to filter not too bad values.
  If the percentage good values (w=1) is greater than 30\
  The else, if the percentage of w >= 0.5 points is greater than 10\ `w_critical`=0.5. In boreal regions, even if the percentage of w >= 0.5 points is only 10\
  We can't rely on points with the wmin weights. Then,
  `y_good` = `y[w >= w_critical]`,
  `ymin` = `pmax( quantile(y_good, alpha/2), 0)`
  `ymax` = `max(y_good)`.

Examples

```
data("CA_NS6")
d = CA_NS6
# head(d)

nptperyear = 23
INPUT <- check_input(d$t, d$y, d$w, QC_flag = d$QC_flag,
     nptperyear = nptperyear, south = FALSE,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
plot_input(INPUT)
```

check_ylu *check_ylu*

Description

Curve fitting values are constrained in the range of `ylu`. Only constrain trough value for a stable background value. But not for peak value.

Usage

```
check_ylu(yfit, ylu)
```

Arguments

`yfit`	Numeric vector, curve fitting result
`ylu`	limits of y value, `[ymin, ymax]`

Value

yfit, the numeric vector in the range of `ylu`.

Examples

```
check_ylu(1:10, c(2, 8))
```

curvefit *Fine curve fitting*

Description

Curve fit vegetation index (VI) time-series of every growing season using fine curve fitting methods.

Usage

```
curvefit(
  y,
  t = index(y),
  tout = t,
  methods = c("AG", "Beck", "Elmore", "Gu", "Klos", "Zhang"),
  w = NULL,
  ...,
  type = 1L,
  use.cpp = FALSE
)
```

Arguments

`y`	Vegetation time-series index, numeric vector
`t`	The corresponding doy of x
`tout`	The output interpolated time.
`methods`	Fine curve fitting methods, can be one or more of `c('AG', 'Beck', 'Elmore', 'Gu', 'Klos', 'Zhang')`.
`w`	(optional) Numeric vector, weights of `y`. If not specified, weights of all `NA` values will be `wmin`, the others will be 1.0.
`...`	other parameters passed to curve fitting function.
`type`	integer, `1` or `-1` • `1`: trough-to-trough curve fitting • `-1`: peak-to-peak curve fitting
`use.cpp`	(unstable, not used) boolean, whether to use c++ defined fine fitting function? If `FALSE`, R version will be used.

Value

fFITs S3 object, see `fFITs()` for details.

Note

'Klos' have too many parameters. It will be slow and not stable.

See Also

`fFITs()`

Examples

```
library(phenofit)
# simulate vegetation time-series
FUN = doubleLog.Beck
par = c(mn  = 0.1, mx  = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- FUN(par, t)

methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout = tout, methods)
```

curvefits *Fine Curve fitting*

Description

Fine Curve fitting for INPUT time-series.

Usage

```
curvefits(INPUT, brks, options = list(), ...)
```

Arguments

INPUT	A list object with the elements of 't', 'y', 'w', 'Tn' (optional) and 'ylu', returned by `check_input`.
brks	A list object with the elements of 'fit' and 'dt', returned by `season` or `season_mov`, which contains the growing season division information.
options	see section: options for fitting for details.
...	other parameters to `curvefit()`

Value

List of phenofit fitting object.

options for fitting

- `methods` (default `c('AG', 'Beck', 'Elmore', 'Zhang')``): Fine curve fitting methods, can be one or more` 'Beck', 'Elmore', 'Zhang', 'Gu', 'Klos')`. Note that 'Gu' and 'Klos' are very slow.
- `iters` (default 2): max iterations of fine fitting.
- `wFUN` (default `wTSM`): Character or function, weights updating function of fine fitting function.
- `wmin` (default 0.1): min weights in the weights updating procedure.
- `use.rough` (default FALSE): Whether to use rough fitting smoothed time-series as input? If `false`, smoothed VI by rough fitting will be used for Phenological metrics extraction; If `true`, original input `y` will be used (rough fitting is used to divide growing seasons and update weights.
- `use.y0` (default TRUE): boolean. whether to use original `y0` as the input of `plot_input`, note that not for curve fitting. `y0` is the original value before the process of `check_input`.
- `nextend` (default 2): Extend curve fitting window, until `nextend` good or marginal points are found in the previous and subsequent growing season.
- `maxExtendMonth` (default 1): Search good or marginal good values in previous and subsequent `maxExtendMonth` period.
- `minExtendMonth` (default 0.5): Extend period defined by `nextend` and `maxExtendMonth`, should be no shorter than `minExtendMonth`. When all points of the input time-series are good value, then the extending period will be too short. In that situation, we can't make sure the connection between different growing seasons is smoothing.

- `minPercValid`: (default 0, not use). If the percentage of good- and marginal- quality points is less than `minPercValid`, curve fiting result is set to `NA`.
- `minT`: (not use). If `Tn` not provided in `INPUT`, `minT` will not be used. `minT` use night temperature Tn to define backgroud value (days with `Tn` < `minT` treated as ungrowing season).

See Also

`FitDL()`

Examples

```
data("CA_NS6")
d = CA_NS6

nptperyear <- 23
INPUT <- check_input(d$t, d$y, d$w, QC_flag = d$QC_flag,
     nptperyear = nptperyear, south = FALSE,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
# plot_input(INPUT)

# Rough fitting and growing season dividing
wFUN <- "wTSM"
brks2 <- season_mov(INPUT,
    options = list(
        rFUN = "smooth_wWHIT", wFUN = wFUN,
        r_min = 0.05, ypeak_min = 0.05,
        lambda = 10,
        verbose = FALSE
    ))
# plot_season(INPUT, brks2, d)
# Fine fitting
fits <- curvefits(
    INPUT, brks2,
    options = list(
        methods = c("AG", "Beck", "Elmore", "Zhang"), #,"klos", "Gu"
        wFUN = wFUN,
        nextend = 2, maxExtendMonth = 2, minExtendMonth = 1, minPercValid = 0.2
    )
)

r_param = get_param(fits)
r_pheno = get_pheno(fits)
r_gof = get_GOF(fits)
d_fit = get_fitting(fits)

g <- plot_curvefits(d_fit, brks2)
grid::grid.newpage(); grid::grid.draw(g)
```

curvefits_LocalModel *curvefits by local model functions of TIMESAT*

Description

Local model functions `f_L(t)`, `f_C(t)` and `f_R(t)` describe the VI variation in intervals around the left minima, the central maxima and the right minima.

Local model function are merged into global model function via `merge_LocalModels()` and Per J\"onsson et al. (2004; their Eq. 12), where cut-off function sharply drop from 1 to 0 in small intervals around `(t_L + t_C)/2` and `(t_C + t_R)/2`.

$$F(t) = \begin{cases} \alpha(t) f_L(t) + [1 - \alpha(t)] f_C(t), t_L < t < t_C \\ \beta(t) f_C(t) + [1 - \beta(t)] f_R(t), t_C < t < t_R \end{cases}$$

Usage

```
curvefits_LocalModel(INPUT, brks, options = list(), ...)

merge_LocalModels(fits)
```

Arguments

`INPUT`	A list object with the elements of 't', 'y', 'w', 'Tn' (optional) and 'ylu', returned by `check_input`.
`brks`	A list object with the elements of 'fit' and 'dt', returned by `season` or `season_mov`, which contains the growing season division information.
`options`	see section: options for fitting for details.
`...`	other parameters to `curvefit()`
`fits`	List objects returned by `curvefits_LocalModel()` (not `curvefits()`).

options for fitting

- `methods` (default `c('AG', 'Beck', 'Elmore', 'Zhang')``): Fine curve fitting methods, can be one or more` 'Beck', 'Elmore', 'Zhang', 'Gu', 'Klos')`. Note that 'Gu' and 'Klos' are very slow.
- `iters` (default 2): max iterations of fine fitting.
- `wFUN` (default `wTSM`): Character or function, weights updating function of fine fitting function.
- `wmin` (default 0.1): min weights in the weights updating procedure.
- `use.rough` (default FALSE): Whether to use rough fitting smoothed time-series as input? If `false`, smoothed VI by rough fitting will be used for Phenological metrics extraction; If `true`, original input `y` will be used (rough fitting is used to divide growing seasons and update weights.
- `use.y0` (default TRUE): boolean. whether to use original `y0` as the input of `plot_input`, note that not for curve fitting. `y0` is the original value before the process of `check_input`.

- `nextend` (default 2): Extend curve fitting window, until `nextend` good or marginal points are found in the previous and subsequent growing season.
- `maxExtendMonth` (default 1): Search good or marginal good values in previous and subsequent `maxExtendMonth` period.
- `minExtendMonth` (default 0.5): Extend period defined by `nextend` and `maxExtendMonth`, should be no shorter than `minExtendMonth`. When all points of the input time-series are good value, then the extending period will be too short. In that situation, we can't make sure the connection between different growing seasons is smoothing.
- `minPercValid`: (default 0, not use). If the percentage of good- and marginal- quality points is less than `minPercValid`, curve fiting result is set to `NA`.
- `minT`: (not use). If `Tn` not provided in `INPUT`, `minT` will not be used. `minT` use night temperature Tn to define backgroud value (days with `Tn` < `minT` treated as ungrowing season).

References

1. Per J\"onsson, P., Eklundh, L., 2004. TIMESAT - A program for analyzing time-series of satellite sensor data. Comput. Geosci. 30, 833-845. doi:10.1016/j.cageo.2004.05.006.

See Also

`curvefits()`

Examples

```
## Not run:
library(phenofit)

data("CA_NS6")
d = CA_NS6

nptperyear <- 23
INPUT <- check_input(d$t, d$y, d$w, QC_flag = d$QC_flag,
     nptperyear = nptperyear, south = FALSE,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
# plot_input(INPUT)

# Rough fitting and growing season dividing
wFUN <- "wTSM"
brks2 <- season_mov(INPUT,
    options = list(
        rFUN = "smooth_wWHIT", wFUN = wFUN,
        r_min = 0.05, ypeak_min = 0.05,
        lambda = 10,
        verbose = FALSE
    ))
# plot_season(INPUT, brks2, d)

# Fine fitting
fits <- curvefits_LocalModel(
    INPUT, brks2,
```

```
    options = list(
        methods = c("AG", "Beck", "Elmore", "Zhang", "Gu"), #,"klos", "Gu"
        wFUN = wFUN,
        nextend = 2, maxExtendMonth = 2, minExtendMonth = 1, minPercValid = 0.2
    ),
    constrain = TRUE
)
# merge local model function into global model function
fits_merged = merge_LocalModels(fits)

## Visualization ---------------------------------------------------------------
l_fitting = map(fits %>% guess_names, get_fitting) #%>% melt_list("period")

d_merged = get_fitting(fits_merged[[2]]) %>% cbind(type = "Merged")
d_raw = l_fitting[2:4] %>% set_names(c("Left", "Central", "Right")) %>%
    melt_list("type")
d_obs = d_raw[, .(t, y, QC_flag)] %>% unique()
d_fit = rbind(d_merged, d_raw)[meth == "Zhang"]

levs = c("Left", "Central", "Right", "Merged")
levs_new = glue("({letters[1:4]}) {levs}") %>% as.character()
d_fit$type %<>% factor(levs, levs_new)

p = ggplot(d_obs, aes(t, y)) +
    geom_point() +
    geom_line(data = d_fit, aes(t, ziter2, color = type)) +
    facet_wrap(~type) +
    labs(x = "Date", y = "EVI") +
    scale_x_date(date_labels = "%b %Y", expand = c(1, 1)*0.08) +
    theme_bw(base_size = 13) +
    theme(legend.position = "none",
          strip.text = element_text(size = 14))
p

## End(Not run)
```

`findpeaks` *findpeaks*

Description

Find peaks (maxima) in a time series. This function is modified from `pracma::findpeaks`.

Usage

```
findpeaks(
  x,
  nups = 1,
  ndowns = nups,
  zero = "0",
```

```
    peakpat = NULL,
    minpeakheight = -Inf,
    minpeakdistance = 1,
    h_min = 0,
    h_max = 0,
    npeaks = 0,
    sortstr = FALSE,
    include_gregexpr = FALSE,
    IsPlot = F
)
```

Arguments

`x`	Numeric vector.
`nups`	minimum number of increasing steps before a peak is reached
`ndowns`	minimum number of decreasing steps after the peak
`zero`	can be `+`, `-`, or `0`; how to interprete succeeding steps of the same value: increasing, decreasing, or special
`peakpat`	define a peak as a regular pattern, such as the default pattern `[+]{1,}[-]{1,}`; if a pattern is provided, the parameters `nups` and `ndowns` are not taken into account
`minpeakheight`	The minimum (absolute) height a peak has to have to be recognized as such
`minpeakdistance`	The minimum distance (in indices) peaks have to have to be counted. If the distance of two maximum extreme value less than `minpeakdistance`, only the real maximum value will be left.
`h_min`	`h` is defined as the difference of peak value to the adjacent left and right trough value (`h_left` and `h_right` respectively). The real peaks should follow `min(h_left, h_right)` >= `h_min`.
`h_max`	Similar as `h_min`, the real peaks should follow `max(h_left, h_right)` >= `h_min`.
`npeaks`	the number of peaks to return. If `sortstr` = true, the largest npeaks maximum values will be returned; If `sortstr` = false, just the first npeaks are returned in the order of index.
`sortstr`	Boolean, Should the peaks be returned sorted in decreasing oreder of their maximum value?
`include_gregexpr`	Boolean (default `FALSE`), whether to include the matched `gregexpr`?
`IsPlot`	Boolean, whether to plot?

Note

In versions before v0.3.4, `findpeaks(c(1, 2, 3, 4, 4, 3, 1))` failed to detect peaks when a flat pattern exit in the middle.

From version v0.3.4, the peak pattern was changed from `[+]{%d,}[-]{%d,}` to `[+]{%d,}[0]{0,}[-]{%d,}`. The latter can escape the flat part successfully.

Examples

```
x <- seq(0, 1, len = 1024)
pos <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.40, 0.44, 0.65, 0.76, 0.78, 0.81)
hgt <- c(4, 5, 3, 4, 5, 4.2, 2.1, 4.3, 3.1, 5.1, 4.2)
wdt <- c(0.005, 0.005, 0.006, 0.01, 0.01, 0.03, 0.01, 0.01, 0.005, 0.008, 0.005)
pSignal <- numeric(length(x))
for (i in seq(along=pos)) {
    pSignal <- pSignal + hgt[i]/(1 + abs((x - pos[i])/wdt[i]))^4
}

plot(pSignal, type="l", col="navy"); grid()
x <- findpeaks(pSignal, npeaks=3, h_min=4, sortstr=TRUE)
points(val~pos, x$X, pch=20, col="maroon")
```

`FitDL` *Fine fitting*

Description

Fine curve fitting function is used to fit vegetation time-series in every growing season.

Usage

```
FitDL.Zhang(y, t = index(y), tout = t, method = "nlm", w, type = 1L, ...)

FitDL.AG(y, t = index(y), tout = t, method = "nlminb", w, type = 1L, ...)

FitDL.AG2(y, t = index(y), tout = t, method = "nlminb", w, type = 1L, ...)

FitDL.Beck(y, t = index(y), tout = t, method = "nlminb", w, type = 1L, ...)

FitDL.Elmore(y, t = index(y), tout = t, method = "nlminb", w, type = 1L, ...)

FitDL.Gu(y, t = index(y), tout = t, method = "nlminb", w, type = 1L, ...)

FitDL.Klos(y, t = index(y), tout = t, method = "BFGS", w, type = 1L, ...)
```

Arguments

`y`	input vegetation index time-series.
`t`	the corresponding doy(day of year) of y.
`tout`	the time of output curve fitting time-series.
`method`	method passed to `optimx` or `optim` function.
`w`	weights
`type`	integer, 1 or -1

- `1`: trough-to-trough curve fitting
- `-1`: peak-to-peak curve fitting

`...` other paraters passed to `optim_pheno()`.

Value

- `tout`: The time of output curve fitting time-series.
- `zs` : Smoothed vegetation time-series of every iteration.
- `ws` : Weights of every iteration.
- `par` : Final optimized parameter of fine fitting.
- `fun` : The name of fine fitting.

References

1. Beck, P.S.A., Atzberger, C., Hogda, K.A., Johansen, B., Skidmore, A.K., 2006. Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. Remote Sens. Environ. https://doi.org/10.1016/j.rse.2005.10.021.
2. Elmore, A.J., Guinn, S.M., Minsley, B.J., Richardson, A.D., 2012. Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. Glob. Chang. Biol. 18, 656-674. https://doi.org/10.1111/j.1365-2486.2011.02521.x.
3. Gu, L., Post, W.M., Baldocchi, D.D., Black, TRUE.A., Suyker, A.E., Verma, S.B., Vesala, TRUE., Wofsy, S.C., 2009. Characterizing the Seasonal Dynamics of Plant Community Photosynthesis Across a Range of Vegetation Types, in: Noormets, A. (Ed.), Phenology of Ecosystem Processes: Applications in Global Change Research. Springer New York, New York, NY, pp. 35-58. https://doi.org/10.1007/978-1-4419-0026-5_2.
4. https://github.com/cran/phenopix/blob/master/R/FitDoubleLogGu.R

Examples

```
# simulate vegetation time-series
t    <- seq(1, 365, 8)
par  <- c(mn = 0.1, mx = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
y    <- doubleLog.Beck(par, t)
data <- data.frame(t, y)
# methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang")
tout <- seq(1, 365, 1)
r <- FitDL.Elmore(y, t, tout)

plot(r, data)
get_GOF(r, data)
get_param(r)
```

f_goal — *Goal function of fine curve fitting methods*

Description

Goal function of fine curve fitting methods

Usage

```
f_goal(par, fun, y, t, pred, w, ylu, ...)
```

Arguments

par	A vector of parameters
fun	A curve fitting function, can be one of doubleAG, doubleLog.Beck, doubleLog.Elmore, doubleLog.Gu, doubleLog.Klos, doubleLog.Zhang, see Logistic() for details.
y	Numeric vector, vegetation index time-series
t	Numeric vector, Date variable
pred	Numeric Vector, predicted values
w	(optional) Numeric vector, weights of y. If not specified, weights of all NA values will be wmin, the others will be 1.0.
ylu	[ymin, ymax], which is used to force ypred in the range of ylu.
...	others will be ignored.

Value

RMSE Root Mean Square Error of curve fitting values.

Examples

```
library(phenofit)

par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
par0 = c( mn  = 0.15, mx  = 0.65, sos = 100, rsp = 0.12, eos = 200, rau = 0.12)

# simulate vegetation time-series
fFUN = doubleLog_Beck
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y    <- fFUN(par, t)

f_goal(par0, fFUN, y, t)
```

get_fitting *getFittings*

Description

Get curve fitting data.frame

Usage

```
get_fitting(x)

## S3 method for class 'list'
get_fitting(x)

## S3 method for class 'fFITs'
get_fitting(x)
```

Arguments

x	`fFITs` object returned by `curvefit()`, or list of `fFITs` objects

Examples

```
library(phenofit)
# simulate vegetation time-series
FUN = doubleLog.Beck
par  = c( mn  = 0.1, mx  = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- FUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods) # `fFITs` (fine-fitting) object
fits <- list(`2001` = fit, `2002` = fit) # multiple years

l_param   <- get_param(fits)
d_GOF     <- get_GOF(fits)
d_fitting <- get_fitting(fits)
l_pheno   <- get_pheno(fits, "AG", IsPlot=TRUE)
```

get_GOF *get_GOF*

Description

Goodness-of-fitting (GOF) of fine curve fitting results.

Usage

```
get_GOF(x, ...)

## S3 method for class 'list'
get_GOF(x, ...)

## S3 method for class 'fFITs'
get_GOF(x, ...)

## S3 method for class 'fFIT'
get_GOF(x, data, ...)
```

Arguments

`x`	`fFITs` object returned by `curvefit()`, or list of `fFITs` objects
`...`	ignored.
`data`	A data.frame with the columns of `c('t', 'y')`

Value

- `meth`: The name of fine curve fitting method
- `RMSE`: Root Mean Square Error
- `NSE` : Nash-Sutcliffe model efficiency coefficient
- `R` : Pearson-Correlation
- `R2` : determined coefficient
- `pvalue`: pvalue of `R`
- `n` : The number of observations

References

1. https://en.wikipedia.org/wiki/Nash-Sutcliffe_model_efficiency_coefficient

2. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

See Also

`curvefit()`

Examples

```
library(phenofit)
# simulate vegetation time-series
FUN = doubleLog.Beck
par  = c( mn  = 0.1, mx  = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- FUN(par, t)
```

```
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods) # `fFITs` (fine-fitting) object
fits <- list(`2001` = fit, `2002` = fit) # multiple years

l_param   <- get_param(fits)
d_GOF     <- get_GOF(fits)
d_fitting <- get_fitting(fits)
l_pheno   <- get_pheno(fits, "AG", IsPlot=TRUE)
```

`get_param` *Get parameters from curve fitting result*

Description

Get parameters from curve fitting result

Usage

```
get_param(x)

## S3 method for class 'list'
get_param(x)

## S3 method for class 'fFITs'
get_param(x)

## S3 method for class 'fFIT'
get_param(x)
```

Arguments

x	`fFITs` object returned by `curvefit()`, or list of `fFITs` objects

Value

A list of `tibble` with the length being equal to the number of methods. Each line of `tibble` cotains the corresponding parameters of each growing season.

Examples

```
library(phenofit)
# simulate vegetation time-series
FUN = doubleLog.Beck
par  = c( mn  = 0.1, mx  = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- FUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods) # `fFITs` (fine-fitting) object
```

```
fits <- list(`2001` = fit, `2002` = fit) # multiple years

l_param   <- get_param(fits)
d_GOF     <- get_GOF(fits)
d_fitting <- get_fitting(fits)
l_pheno   <- get_pheno(fits, "AG", IsPlot=TRUE)
```

---

get_pheno *get_pheno*

Description

Get yearly vegetation phenological metrics of a curve fitting method

Usage

```
get_pheno(x, ...)

## S3 method for class 'rfit'
get_pheno(x, TRS = c(0.2, 0.5), asymmetric = TRUE, ...)

## S3 method for class 'list'
get_pheno(
  x,
  method,
  TRS = c(0.2, 0.5, 0.6),
  analytical = FALSE,
  smoothed.spline = FALSE,
  IsPlot = FALSE,
  show.title = TRUE,
  ...
)

## S3 method for class 'fFITs'
get_pheno(
  x,
  method,
  TRS = c(0.2, 0.5),
  analytical = FALSE,
  smoothed.spline = FALSE,
  IsPlot = FALSE,
  title.left = "",
  show.PhenoName = TRUE,
  ...
)
```

Arguments

`x`	One of: • `rfit` (rought fitting object), returned by `brks2rfit()`. • `fFITs` (fine fitting object), return by multiple curve fitting methods by `curvefit()` for a growing season. • list of `fFITs()` object, for multiple growing seasons.
`...`	ignored.
`TRS`	Threshold for `PhenoTrs`.
`asymmetric`	If true, background value in spring season and autumn season is regarded as different.
`method`	Which fine curve fitting method to be extracted?
`analytical`	If true, `numDeriv` package `grad` and `hess` will be used; if false, `D1` and `D2` will be used.
`smoothed.spline`	Whether apply `smooth.spline` first?
`IsPlot`	Boolean. Whether to plot figure?
`show.title`	Whether to show the name of fine curve fitting method in top title?
`title.left`	String of growing season flag.
`show.PhenoName`	Whether to show phenological methods names in the top panel?
`fFITs`	`fFITs` object returned by `curvefits()`

Value

List of every year phenology metrics

Examples

```
library(phenofit)
# simulate vegetation time-series
FUN = doubleLog.Beck
par  = c( mn  = 0.1, mx  = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- FUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods) # `fFITs` (fine-fitting) object
fits <- list(`2001` = fit, `2002` = fit) # multiple years

l_param   <- get_param(fits)
d_GOF     <- get_GOF(fits)
d_fitting <- get_fitting(fits)
l_pheno   <- get_pheno(fits, "AG", IsPlot=TRUE)
```

GOF *GOF*

Description

Good of fitting

Usage

```
GOF(Y_obs, Y_sim, w, include.r = TRUE, include.cv = FALSE)
```

Arguments

Y_obs	Numeric vector, observations
Y_sim	Numeric vector, corresponding simulated values
w	Numeric vector, weights of every points. If w included, when calculating mean, Bias, MAE, RMSE and NSE, w will be taken into considered.
include.r	If true, r and R2 will be included.
include.cv	If true, cv will be included.

Value

- `RMSE` root mean square error
- `NSE` NASH coefficient
- `MAE` mean absolute error
- `AI` Agreement index (only good points (w == 1)) participate to calculate. See details in Zhang et al., (2015).
- `Bias` bias
- `Bias_perc` bias percentage
- `n_sim` number of valid obs
- `cv` Coefficient of variation
- `R2` correlation of determination
- `R` pearson correlation
- `pvalue` pvalue of `R`

References

Zhang Xiaoyang (2015), http://dx.doi.org/10.1016/j.rse.2014.10.012

Examples

```
Y_obs = rnorm(100)
Y_sim = Y_obs + rnorm(100)/4
GOF(Y_obs, Y_sim)
```

input_single — *input object with one growing season per year*

Description

Variables in `input_single`:

- `t`: date of compositing image
- `y`: EVI
- `w`: weights of data point
- `ylu`: lower and upper boundary
- `nptperyear`: points per year
- `south`: boolean, whether in south Hemisphere?

Usage

```
data('input_single')
```

Format

An object of class `list` of length 6.

Logistic — *Fine fitting functions*

Description

double logistics, piecewise logistics and many other functions to curve fit VI time-series.

Usage

```
Logistic(par, t)

doubleLog.Zhang(par, t)

doubleLog.AG(par, t)

doubleLog.AG2(par, t)

doubleLog.Beck(par, t)

doubleLog.Elmore(par, t)

doubleLog.Gu(par, t)

doubleLog.Klos(par, t)
```

Arguments

par	A vector of parameters
t	A Date or numeric vector

Details

- Logistic The traditional simplest logistic function. It can be only used in half growing season, i.e. vegetation green-up or senescence period.
- doubleLog.Zhang Piecewise logistics, (Zhang Xiaoyang, RSE, 2003).
- doubleAG Asymmetric Gaussian.
- doubleLog.Beck Beck logistics.
- doubleLog.Gu Gu logistics.
- doubleLog.Elmore Elmore logistics.
- doubleLog.Klos Klos logistics.

All of those function have par and formula attributes for the convenience for analytical D1 and D2

References

1. Beck, P.S.A., Atzberger, C., Hogda, K.A., Johansen, B., Skidmore, A.K., 2006. Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. Remote Sens. Environ. https://doi.org/10.1016/j.rse.2005.10.021.
2. Elmore, A.J., Guinn, S.M., Minsley, B.J., Richardson, A.D., 2012. Landscape controls on the timing of spring, autumn, and growing season length in mid-Atlantic forests. Glob. Chang. Biol. 18, 656-674. https://doi.org/10.1111/j.1365-2486.2011.02521.x.
3. Gu, L., Post, W.M., Baldocchi, D.D., Black, TRUE.A., Suyker, A.E., Verma, S.B., Vesala, TRUE., Wofsy, S.C., 2009. Characterizing the Seasonal Dynamics of Plant Community Photosynthesis Across a Range of Vegetation Types, in: Noormets, A. (Ed.), Phenology of Ecosystem Processes: Applications in Global Change Research. Springer New York, New York, NY, pp. 35-58. https://doi.org/10.1007/978-1-4419-0026-5_2.
4. Peter M. Atkinson, et al., 2012, RSE, 123:400-417
5. https://github.com/cran/phenopix/blob/master/R/FitDoubleLogGu.R

Examples

```
# simulate vegetation time-series
t    <- seq(1, 365, 8)
par  <- c(mn = 0.1, mx = 0.7, sos = 50, rsp = 0.1, eos = 250, rau = 0.1)
y    <- doubleLog.Beck(par, t)
data <- data.frame(t, y)
# methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang")
tout <- seq(1, 365, 1)
r <- FitDL.Elmore(y, t, tout)

plot(r, data)
```

```
get_GOF(r, data)
get_param(r)
```

MOD13A1 *MOD13A1*

Description

A data.table dataset, raw data of MOD13A1 data, clipped in 10 representative points ('DE-Obe', 'IT-Col', 'CN-Cha', 'AT-Neu', 'ZA-Kru', 'AU-How', 'CA-NS6', 'US-KS2', 'CH-Oe2', 'CZ-wet').

Usage

```
data('MOD13A1')
```

Format

An object of class `list` of length 2.

Details

Variables in MOD13A1:

- `dt`: vegetation index data
  - `system:index`: image index
  - `DayOfYear`: Numeric, Julian day of year
  - `DayOfYear`: corresponding doy of compositing NDVI and EVI
  - `DetailedQA`: VI quality indicators
  - `SummaryQA`: Quality reliability of VI pixel
  - `EVI`: Enhanced Vegetation Index
  - `NDVI`: Normalized Difference Vegetation Index
  - `date`: Date, corresponding date
  - `site`: String, site name
  - `sur_refl_b01`: Red surface reflectance
  - `sur_refl_b02`: NIR surface reflectance
  - `sur_refl_b03`: Blue surface reflectance
  - `sur_refl_b07`: MIR surface reflectance
  - `.geo`: geometry
- `st`: station info
  - `ID`: site ID
  - `site`: site name
  - `lat`: latitude
  - `lon`: longitude
  - `IGBPname`: IGBP land cover type

References

1. https://code.earthengine.google.com/dataset/MODIS/006/MOD13A1

---

movmean *movmean*

Description

NA and Inf values in the yy will be ignored automatically.

Usage

```
movmean(y, halfwin = 1L, SG_style = FALSE, w = NULL)
```

Arguments

y	A numeric vector.
halfwin	Integer, half of moving window size
SG_style	If true, head and tail values will be in the style of SG (more weights on the center point), else traditional moving mean style.
w	Corresponding weights of yy, same long as yy.

Examples

```
x <- 1:100
x[50] <- NA; x[80] <- Inf
s1 <- movmean(x, 2, SG_style = TRUE)
s2 <- movmean(x, 2, SG_style = FALSE)
```

---

optim_pheno *optim_pheno*

Description

Interface of optimization functions for double logistics and other parametric curve fitting functions.

Usage

```
optim_pheno(
  prior,
  sFUN,
  y,
  t,
  tout,
  method,
  w,
  nptperyear,
  ylu,
  iters = 2,
  wFUN = wTSM,
  lower = -Inf,
  upper = Inf,
  constrain = TRUE,
  verbose = FALSE,
  ...,
  use.cpp = FALSE
)
```

Arguments

`prior`	A vector of initial values for the parameters for which optimal values are to be found. `prior` is suggested giving a column name.
`sFUN`	The name of fine curve fitting functions, can be one of `'FitAG'`, `'FitDL.Beck'`, `'FitDL.Elmore'`, `'Fi`
`y`	Numeric vector, vegetation index time-series
`t`	Numeric vector, `Date` variable
`tout`	Corresponding doy of prediction.
`method`	The name of optimization method to solve fine fitting, one of `'BFGS'`,`'CG'`,`'Nelder-Mead'`, `'L-BFGS-B` and `'spg'`,`'Rcgmin'`,`'Rvmmin'`, `'newuoa'`,`'bobyqa'`,`'nmkb'`,`'hjkb'`.
`w`	(optional) Numeric vector, weights of `y`. If not specified, weights of all `NA` values will be `wmin`, the others will be 1.0.
`nptperyear`	Integer, number of images per year, passed to `wFUN`. Only `wTSM()` needs `nptperyear`. If not specified, `nptperyear` will be calculated based on `t`.
`ylu`	`[ymin, ymax]`, which is used to force `ypred` in the range of `ylu`.
`iters`	How many times curve fitting is implemented.
`wFUN`	weights updating function, can be one of 'wTSM', 'wChen' and 'wBisquare'.
`lower, upper`	vectors of lower and upper bounds, replicated to be as long as `start`. If unspecified, all parameters are assumed to be unconstrained.
`constrain`	boolean, whether to use parameter constrain
`verbose`	Whether to display intermediate variables?
`...`	other parameters passed to `I_optim()` or `I_optimx()`.
`use.cpp`	(unstable, not used) boolean, whether to use c++ defined fine fitting function? If `FALSE`, R version will be used.

Value

A `fFIT()` object, with the element of:

- `tout`: The time of output curve fitting time-series.
- `zs` : Smoothed vegetation time-series of every iteration.
- `ws` : Weights of every iteration.
- `par` : Final optimized parameter of fine fitting.
- `fun` : The name of fine fitting.

See Also

`fFIT()`, `stats::nlminb()`

Examples

```
# library(magrittr)
# library(purrr)

# simulate vegetation time-series
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)

FUN = doubleLog_Beck
par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
par0 = c( mn  = 0.15, mx  = 0.65, sos = 100, rsp = 0.12, eos = 200, rau = 0.12)

y <- FUN(par, t)

methods = c("BFGS", "ucminf", "nlm", "nlminb")
opt1 <- I_optim(par0, doubleLog_Beck, y, t, methods) # "BFGS", "ucminf", "nlm",
# opt2 <- I_optimx(prior, fFUN, y, t, tout, )

sFUN   = "doubleLog.Beck" # doubleLog.Beck
r <- optim_pheno(par0, sFUN, y, t, tout, method = methods[4],
              nptperyear = 46, iters = 2, wFUN = wTSM, verbose = FALSE, use.julia = FALSE)
```

`opt_FUN` *Unified optimization function*

Description

`I_optimx` is rich of functionality, but with a low computing performance. Some basic optimization functions are unified here, with some input and output format.

- `opt_ncminf` General-Purpose Unconstrained Non-Linear Optimization, see `ucminf::ucminf()`.
- `opt_nlminb` Optimization using PORT routines, see `stats::nlminb()`.
- `opt_nlm` Non-Linear Minimization, `stats::nlm()`.
- `opt_optim` General-purpose Optimization, see `stats::optim()`.

Usage

```
opt_ucminf(par0, objective, ...)

opt_nlm(par0, objective, ...)

opt_optim(par0, objective, method = "BFGS", ...)

opt_nlminb(par0, objective, ...)
```

Arguments

`par0`	Initial values for the parameters to be optimized over.
`objective`	A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
`...`	other parameters passed to `objective`.
`method`	optimization method to be used in `p_optim`. See `stats::optim()`.

Value

- `convcode`: An integer code. 0 indicates successful convergence. Various methods may or may not return sufficient information to allow all the codes to be specified. An incomplete list of codes includes
  - `1`: indicates that the iteration limit `maxit` had been reached.
  - `20`: indicates that the initial set of parameters is inadmissible, that is, that the function cannot be computed or returns an infinite, NULL, or NA value.
  - `21`: indicates that an intermediate set of parameters is inadmissible.
  - `10`: indicates degeneracy of the Nelder–Mead simplex.
  - `51`: indicates a warning from the `"L-BFGS-B"` method; see component `message` for further details.
  - `52`: indicates an error from the `"L-BFGS-B"` method; see component `message` for further details.
  - `9999`: error
- `value`: The value of fn corresponding to par
- `par`: The best parameter found
- `nitns`: the number of iterations
- `fevals`: The number of calls to `objective`.

See Also

`optim_pheno()`, `I_optim()`

Examples

```
library(phenofit)
library(ggplot2)
library(magrittr)
library(purrr)
library(data.table)

# simulate vegetation time-series
fFUN = doubleLog_Beck
par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
par0 = c( mn  = 0.15, mx  = 0.65, sos = 100, rsp = 0.12, eos = 200, rau = 0.12)

t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)

optFUNs <- c("opt_ucminf", "opt_nlminb", "opt_nlm", "opt_optim") %>% set_names(., .)
opts <- lapply(optFUNs, function(optFUN){
    optFUN <- get(optFUN)
    opt    <- optFUN(par0, f_goal, y = y, t = t, fun = fFUN)
    opt$ysim <- fFUN(opt$par, t)
    opt
})

# visualization
df   <- map(opts, "ysim") %>% as.data.table() %>% cbind(t, y, .)
pdat <- data.table::melt(df, c("t", "y"), variable.name = "optFUN")

ggplot(pdat) +
    geom_point(data = data.frame(t, y), aes(t, y), size = 2) +
    geom_line(aes(t, value, color = optFUN), linewidth = 0.9)
```

PhenoDeriv *Phenology extraction in Derivative method (DER)*

Description

Phenology extraction in Derivative method (DER)

Usage

```
PhenoDeriv(x, t, ...)

## S3 method for class 'fFIT'
PhenoDeriv(x, t = NULL, analytical = FALSE, smoothed.spline = FALSE, ...)

## Default S3 method:
PhenoDeriv(x, t, der1, IsPlot = TRUE, show.legend = TRUE, ...)
```

Arguments

`x`	numeric vector, or `fFIT` object returned by `curvefit()`.
`t`	doy vector, corresponding doy of vegetation index.
`...`	Other parameters will be ignored.
`analytical`	If true, `numDeriv` package `grad` and `hess` will be used; if false, `D1` and `D2` will be used.
`smoothed.spline`	Whether apply `smooth.spline` first?
`der1`	the first order difference
`IsPlot`	whether to plot?
`show.legend`	whether show figure lelend?

References

1. Filippa, G., Cremonese, E., Migliavacca, M., Galvagno, M., Forkel, M., Wingate, L., … Richardson, A. D. (2016). Phenopix: A R package for image-based vegetation phenology. Agricultural and Forest Meteorology, 220, 141–150. doi:10.1016/j.agrformet.2016.01.006

See Also

`PhenoTrs()`, `PhenoGu()`, `PhenoKl()`

PhenoGu — *Phenology extraction in GU method (GU)*

Description

Phenology extraction in GU method (GU)

Usage

```
PhenoGu(x, t, ...)

## S3 method for class 'fFIT'
PhenoGu(x, t = NULL, analytical = FALSE, smoothed.spline = FALSE, ...)

## Default S3 method:
PhenoGu(x, t, der1, IsPlot = TRUE, ...)
```

Arguments

`x`	numeric vector, or `fFIT` object returned by `curvefit()`.
`t`	doy vector, corresponding doy of vegetation index.
`...`	other parameters to `PhenoGu.default()` or `PhenoGu.fFIT()`
`analytical`	If true, `numDeriv` package `grad` and `hess` will be used; if false, `D1` and `D2` will be used.
`smoothed.spline`	Whether apply `smooth.spline` first?
`der1`	the first order difference
`IsPlot`	whether to plot?

Value

A numeric vector, with the elements of:

- `UD`: upturn date
- `SD`: stabilisation date
- `DD`: downturn date
- `RD`: recession date

References

1. Gu, L., Post, W. M., Baldocchi, D. D., Black, T. A., Suyker, A. E., Verma, S. B., … Wofsy, S. C. (2009). Characterizing the Seasonal Dynamics of Plant Community Photosynthesis Across a Range of Vegetation Types. In A. Noormets (Ed.), Phenology of Ecosystem Processes: Applications in Global Change Research (pp. 35–58). New York, NY: Springer New York. doi:10.1007/9781441900265_2
2. Filippa, G., Cremonese, E., Migliavacca, M., Galvagno, M., Forkel, M., Wingate, L., … Richardson, A. D. (2016). Phenopix: A R package for image-based vegetation phenology. Agricultural and Forest Meteorology, 220, 141–150. doi:10.1016/j.agrformet.2016.01.006

Examples

```
# `doubleLog.Beck` simulate vegetation time-series
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
y <- doubleLog.Beck(par, t)

methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods)
x <- fit$model$AG # one model

par(mfrow = c(2, 2))
PhenoTrs(x)
PhenoDeriv(x)
PhenoGu(x)
PhenoKl(x)
```

PhenoKl *Phenology extraction in Inflection method (Zhang)*

Description

Phenology extraction in Inflection method (Zhang)

Usage

```
PhenoKl(
  fFIT,
  t = NULL,
  analytical = FALSE,
  smoothed.spline = FALSE,
  IsPlot = TRUE,
  show.legend = TRUE,
  ...
)
```

Arguments

`fFIT`	object return by `curvefit()`
`t`	doy vector, corresponding doy of vegetation index.
`analytical`	If true, `numDeriv` package `grad` and `hess` will be used; if false, `D1` and `D2` will be used.
`smoothed.spline`	Whether apply `smooth.spline` first?
`IsPlot`	whether to plot?
`show.legend`	whether show figure lelend?
`...`	Other parameters will be ignored.

Value

A numeric vector, with the elements of: `Greenup`, `Maturity`, `Senescence`, `Dormancy`.

References

1. Zhang, X., Friedl, M. A., Schaaf, C. B., Strahler, A. H., Hodges, J. C. F. F., Gao, F., ... Huete, A. (2003). Monitoring vegetation phenology using MODIS. Remote Sensing of Environment, 84(3), 471–475. doi:10.1016/S00344257(02)001359

Examples

```
# `doubleLog.Beck` simulate vegetation time-series
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
y <- doubleLog.Beck(par, t)

methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods)
x <- fit$model$AG # one model

par(mfrow = c(2, 2))
PhenoTrs(x)
PhenoDeriv(x)
PhenoGu(x)
PhenoKl(x)
```

PhenoTrs *Phenology extraction in Threshold method (TRS)*

Description

Phenology extraction in Threshold method (TRS)

Usage

```
PhenoTrs(
  x,
  t = NULL,
  approach = c("White", "Trs"),
  trs = 0.5,
  asymmetric = TRUE,
  IsPlot = TRUE,
  ...
)

## S3 method for class 'fFIT'
PhenoTrs(x, t = NULL, ...)

## Default S3 method:
PhenoTrs(
  x,
  t = NULL,
  approach = c("White", "Trs"),
  trs = 0.5,
  asymmetric = TRUE,
  IsPlot = TRUE,
  ...
)
```

Arguments

`x`	numeric vector, or `fFIT` object returned by `curvefit()`.
`t`	doy vector, corresponding doy of vegetation index.
`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", in (0, 1).
`asymmetric`	If true, background value in spring season and autumn season is regarded as different.
`IsPlot`	whether to plot?
`...`	other parameters to PhenoPlot

See Also

`PhenoDeriv()`, `PhenoGu()`, `PhenoKl()`

Examples

```
# `doubleLog.Beck` simulate vegetation time-series
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
par  = c( mn  = 0.1 , mx  = 0.7 , sos = 50 , rsp = 0.1 , eos = 250, rau = 0.1)
y <- doubleLog.Beck(par, t)

methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fit <- curvefit(y, t, tout, methods)
x <- fit$model$AG # one model

par(mfrow = c(2, 2))
PhenoTrs(x)
PhenoDeriv(x)
PhenoGu(x)
PhenoKl(x)
```

---

plot_curvefits *plot_curvefits*

Description

plot_curvefits

Usage

```
plot_curvefits(
  d_fit,
  seasons,
  d_obs = NULL,
```

```
  title = NULL,
  xlab = "Time",
  ylab = "Vegetation Index",
  yticks = NULL,
  font.size = 14,
  theme = NULL,
  cex = 2,
  shape = "point",
  angle = 30,
  show.legend = TRUE,
  layer_extra = NULL,
  ...
)
```

Arguments

d_fit	data.frame of curve fittings returned by `get_fitting()`.
seasons	growing season division object returned by `season()` and `season_mov()`.
d_obs	data.frame of original vegetation time series, with the columns of `t`, `y` and `QC_flag`. If not specified, it will be determined from `d_fit`.
title	String, title of figure.
xlab, ylab	String, title of `xlab` and `ylab`.
yticks	ticks of y axis
font.size	Font size of axis.text
theme	ggplot theme
cex	point size for VI observation.
shape	the shape of input VI observation? `line` or `point`
angle	`text.x` angle
show.legend	Boolean
layer_extra	(not used) extra ggplot layers
...	ignored

Examples

```
data("CA_NS6")
d = CA_NS6

nptperyear <- 23
INPUT <- check_input(d$t, d$y, d$w, QC_flag = d$QC_flag,
     nptperyear = nptperyear, south = FALSE,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
# plot_input(INPUT)

# Rough fitting and growing season dividing
wFUN <- "wTSM"
brks2 <- season_mov(INPUT,
```

```
    options = list(
        rFUN = "smooth_wWHIT", wFUN = wFUN,
        r_min = 0.05, ypeak_min = 0.05,
        lambda = 10,
        verbose = FALSE
    ))
# plot_season(INPUT, brks2, d)
# Fine fitting
fits <- curvefits(
    INPUT, brks2,
    options = list(
        methods = c("AG", "Beck", "Elmore", "Zhang"), #,"klos", "Gu"
        wFUN = wFUN,
        nextend = 2, maxExtendMonth = 2, minExtendMonth = 1, minPercValid = 0.2
    )
)

r_param = get_param(fits)
r_pheno = get_pheno(fits)
r_gof = get_GOF(fits)
d_fit = get_fitting(fits)

g <- plot_curvefits(d_fit, brks2)
grid::grid.newpage(); grid::grid.draw(g)
```

---

plot_input *Plot INPUT returned by check_input*

Description

Plot INPUT returned by check_input

Usage

```
plot_input(INPUT, wmin = 0.2, show.y0 = TRUE, ylab = "VI", ...)
```

Arguments

INPUT	A list object with the elements of `t`, `y`, `w`, `Tn` (optional) and `ylu`, returned by `check_input()`.
wmin	double, minimum weigth (i.e. weight of snow, ice and cloud).
show.y0	boolean. Whether to show original time-series `y0` or processed time-series `y` by `check_input()`?
ylab	y axis title
...	other parameter will be ignored.

Examples

```
library(phenofit)
data("CA_NS6"); d = CA_NS6
# global parameter
IsPlot = TRUE
nptperyear = 23
ypeak_min  = 0.05

INPUT    <- check_input(d$t, d$y, d$w, d$QC_flag, nptperyear,
                        maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
plot_input(INPUT)
```

---

plot_season *plot_season*

Description

Plot growing season divding result.

Usage

```
plot_season(
  INPUT,
  brks,
  plotdat,
  IsPlot.OnlyBad = FALSE,
  show.legend = TRUE,
  ylab = "VI",
  title = NULL,
  show.shade = TRUE,
  margin = 0.35
)
```

Arguments

INPUT	A list object with the elements of t, y, w, Tn (optional) and ylu, returned by check_input().
brks	A list object returned by season or season_mov.
plotdat	(optional) A list or data.table, with t, y and w. Only if IsPlot=TRUE, plot_input() will be used to plot. Known that y and w in INPUT have been changed, we suggest using the original data.table.
IsPlot.OnlyBad	If true, only plot partial figures whose NSE < 0.3.
show.legend	Whether to show legend?
ylab	y axis title
title	The main title (on top)
show.shade	Boolean, period inside growing cycle colored as shade?
margin	ylim = c(ymin, ymax + margin * A); A = ymax - ymin.

qcFUN *Initial weights according to qc*

Description

- `getBits`: Extract bitcoded QA information from bin value
- `qc_summary`: Initial weigths based on Quality reliability of VI pixel, suit for MOD13A1, MOD13A2 and MOD13Q1 (SummaryQA band).
- `qc_5l`: Initial weights based on Quality control of five-level confidence score, suit for MCD15A3H(LAI, FparLai_QC), MOD17A2H(GPP, Psn_QC) and MOD16A2(ET, ET_QC).
- `qc_StateQA`: Initial weights based on `StateQA`, suit for MOD09A1, MYD09A1.
- `qc_FparLai`: For MODIS LAI
- `qc_NDVI3g`: For AVHRR NDVI3g
- `qc_NDVIv4`: For AVHRR NDVIv4

Usage

```
getBits(x, start, end = start)

qc_summary(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_StateQA(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_FparLai(QA, FparLai_QC = NULL, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_5l(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_NDVIv4(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_NDVI3g(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

qc_SPOT(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
```

Arguments

`x`	Binary value
`start`	Bit starting position, count from zero
`end`	Bit ending position
`QA`	quality control variable
`wmin`	Double, minimum weigth (i.e. weight of snow, ice and cloud).
`wmid`	Dougle, middle weight, i.e. marginal
`wmax`	Double, maximum weight, i.e. good
`FparLai_QC`	Another QC flag of `MCD15A3H`

Details

If FparLai_QC specified, I_margin = SCF_QC >= 2 & SCF_QC <= 3.

Value

A list object with

- weigths: Double vector, initial weights.
- QC_flag: Factor vector, with the level of c("snow", "cloud", "shadow", "aerosol", "marginal", "good")

Note

qc_5l and qc_NDVIv4 only returns weight, without QC_flag.

References

https://developers.google.com/earth-engine/datasets/catalog/MODIS_006_MOD13A1

https://developers.google.com/earth-engine/datasets/catalog/MODIS_006_MCD15A3H

Erwin Wolters, Else Swinnen, Carolien Toté, Sindy Sterckx. SPOT-VGT COLLECTION 3 PRODUCTS USER MANUAL V1.2, 2018, P47

See Also

qc_sentinel2()

Examples

```
set.seed(100)
QA <- as.integer(runif(100, 0, 2^7))

r1 <- qc_summary(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
r2 <- qc_StateQA(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
r_5l <- qc_5l(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
r_NDVI3g <- qc_NDVI3g(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
r_NDVIv4 <- qc_NDVIv4(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
```

---

qc_sentinel2 *Initial weights for sentinel2 according to SCL band*

Description

SCL Value	Description	Quality	weight
1	Saturated or defective	Bad	w_{min}
2	Dark Area Pixels	Bad	w_{min}
3	Cloud Shadows	Bad	w_{min}
4	Vegetation	Good	w_{max}
5	Bare Soils	Good	w_{max}
6	Water	Good	w_{max}
7	Clouds Low Probability / Unclassified	Good	w_{max}
8	Clouds Medium Probability	Marginal	w_{mid}
9	Clouds High Probability	Bad	w_{mid}
10	Cirrus	Good	w_{mid}
11	Snow / Ice	Bad	w_{mid}

Usage

```
qc_sentinel2(SCL, wmin = 0.2, wmid = 0.5, wmax = 1)
```

Arguments

`SCL`	quality control variable for sentinel2
`wmin`	Double, minimum weigth (i.e. weight of snow, ice and cloud).
`wmid`	Dougle, middle weight, i.e. marginal
`wmax`	Double, maximum weight, i.e. good

References

https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR

Examples

```
qc_sentinel2(1:11)
```

`rcpp_wSG` *Weighted Savitzky-Golay written in RcppArmadillo*

Description

NA and Inf values in the yy has been ignored automatically.

Usage

```
rcpp_wSG(y, halfwin = 1L, d = 1L, w = NULL)

rcpp_SG(y, halfwin = 1L, d = 1L)
```

Arguments

y	colvec
halfwin	halfwin of Savitzky-Golay
d	polynomial of degree. When d = 1, it becomes moving average.
w	colvec of weight

Examples

```
y <- 1:15
w <- seq_along(y)/length(y)

frame = 5
d = 2
s1 <- rcpp_wSG(y, frame, d, w)
s2 <- rcpp_SG(y, frame, d)
```

---

season_mov *Moving growing season division*

Description

Moving growing season division

Usage

```
season_mov(INPUT, options = list(), ..., years.run = NULL)
```

Arguments

INPUT	A list object with the elements of `t`, `y`, `w`, `Tn` (optional) and `ylu`, returned by `check_input()`.
options	see the following section `options for season` for details.
...	others parameter to `set_options()`
years.run	Numeric vector. Which years to run? If not specified, it is all years.

options for season

(a) Parameters for rough fitting:

- `rFUN` : character (default `smooth_wWHIT`), the name of rough curve fitting function, can be one of `c("smooth_wSG", "smooth_wWHIT", "smooth_wHANTS")`, which are corresponding to `smooth_wSG()`, `smooth_wWHIT()` and `smooth_wHANTS()`.
- `wFUN` : character (default `wTSM`), the name of weights updating functions, can be one of c("wTSM", "wChen", "wBisquare", "wSELF"). See `wTSM()`, `wChen()`, `wBisquare()` and `wSELF()` for details.
- `iters` : integer (default 2), the number of rough fitting iterations.
- `wmin` : double, the minimum weight of bad points (i.e. snow, ice and cloud).
- `verbose` : logical (default `FALSE`). If `TRUE`, `options$season` will be printed on the console.
- `lambda` : double (default NULL), the smoothing parameter of `smooth_wWHIT()`.
  - If `lambda` = `NULL`, V-curve theory will be employed to find the optimal `lambda`. See `lambda_vcurve()` for details.
- `frame` : integer (default NULL), the parameter of `smooth_wSG()`, moving window size.
  - If `frame` = `NULL`, `frame` will be reset as `floor(nptperyear/5)*2 + 1` (refered by TIMESAT).
- `nf` : integer (default 4), the number of frequencies in `smooth_wHANTS()`.
- `maxExtendMonth`: integer (default 12), previous and subsequent `maxExtendMonth` (in month) data were added to the current year for rough fitting.
- `nextend` : integer (default NULL), same as `maxExtendMonth`, but in points.
  - If `nextend` provided, `maxExtendMonth` will be ignored.
  - If `nextend` = `NULL`, `nextend` will be reset as `ceiling(maxExtendMonth/12*nptperyear)`

(b) Parameters for growing season division:

- `minpeakdistance` : double (default NULL), the minimum distance of two peaks (in points). If the distance of two maximum extreme value less than `minpeakdistance`, only the maximum one will be kept.
  - If `minpeakdistance` = `NULL`, it will be reset as `nptperyear/6`.
- `r_max` : double (default 0.2; in (0, 1)). `r_max` and `r_min` are used to eliminate fake peaks and troughs.
  - The real peaks should satisfy:
    1. $max(h_{peak,L}, h_{peak,R}) > r_{max}A$
    2. $min(h_{peak,L}, h_{peak,R}) > r_{min}A$, where $h_{peak,L}, h_{peak,R}$ are height difference from the peak to the left- and right-hand troughs.
  - The troughs should satisfy:
    1. $max(h_{trough,L}, h_{trough,R}) > r_{max}A$, where $h_{trough,L}, h_{trough,R}$ are height difference from the trough to the left- and right-hand peaks.
- `r_min` : double (default 0.05; in (0, 1)), see above `r_max` for details. `r_min` < `r_max`.
- `rtrough_max` : double (default 0.6, in (0, 1)), $y_{peak} <= rtrough_max * A + ylu[1]$.
- `ypeak_min` : double 0.1 (in VI unit), $y_{peak} >= ypeak_min$.
- `.check_season` : logical (default `TRUE`). check the growing season length according to `len_min` and `len_max`. If `FALSE`, `len_min` and `len_max` will lose their effect.
- `len_min` : integer (default 45), the minimum length (in days) of growing season

- len_max : integer (default 650), the minimum length (in days) of growing season
- adj.param : logical. If TRUE (default), if there are too many or too less peaks and troughs, phenofit will automatically adjust rough curve fitting function parameters. See MaxPeaksPerYear and MaxTroughsPerYear for details.
- MaxPeaksPerYear (optional) : integer (default 2), the max number of peaks per year. If PeaksPerYear > MaxPeaksPerYear, then lambda = lambda*2.
- MaxTroughsPerYear (optional) : integer (default 3), the max number of troughs per year. If TroughsPerYear > MaxTroughsPerYear, then lambda = lambda*2.
- calendarYear : logical (default FALSE). If TRUE, the start and end of a calendar year will be regarded as growing season division (North Hemisphere is from 01 Jan to 31 Dec; South Hemisphere is from 01 Jul to 30 Jun).
- rm.closed : logical (default TRUE). If TRUE, closed peaks (or troughs) will be further tidied. Only the maximum
- is.continuous (not used): logical (default TRUE). This parameter is for fluxnet2015 fluxsite data, where the input might be not continuous.

References

1. Kong, D., Zhang, Y., Wang, D., Chen, J., & Gu, X. (2020). Photoperiod Explains the Asynchronization Between Vegetation Carbon Phenology and Vegetation Greenness Phenology. Journal of Geophysical Research: Biogeosciences, 125(8), e2020JG005636. https://doi.org/10.1029/2020JG005636
2. Kong, D., Zhang, Y., Gu, X., & Wang, D. (2019). A robust method for reconstructing global MODIS EVI time series on the Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 155, 13-24.

See Also

season()

Examples

```
data("CA_NS6")
d <- CA_NS6

nptperyear <- 23
INPUT <- check_input(d$t, d$y, d$w,
    QC_flag = d$QC_flag,
    nptperyear = nptperyear, south = FALSE,
    maxgap = nptperyear / 4, alpha = 0.02, wmin = 0.2
)

# curve fitting by year
brks_mov <- season_mov(INPUT,
    options = list(
        rFUN = "smooth_wWHIT", wFUN = "wTSM",
        lambda = 10,
        r_min = 0.05, ypeak_min = 0.05,
        verbose = TRUE
    )
)
```

```
plot_season(INPUT, brks_mov)

rfit <- brks2rfit(brks_mov)
# Phenological Metrics from rough fitting
r <- get_pheno(rfit)
```

`set_options` *set and get phenofit option*

Description

set and get phenofit option

Usage

```
set_options(..., options = NULL)

get_options(names = NULL)
```

Arguments

`...`	list of phenofit options FUN_season: character, `season_mov` or `season` rFUN: character, rough fitting function. `smooth_wWHIT`, `smooth_wSG` or `smooth_wHANTs`.

`options`	If not NULL, `options` will overwrite the default parameters (`get_options()`).
	• `qcFUN` : function to process qc flag, see `qcFUN()` for details.
	• `nptperyear` : Integer, number of images per year.
	• `wFUN` : character (default `wTSM`), the name of weights updating functions, can be one of c("wTSM", "wChen", "wBisquare", "wSELF"). See `wTSM()`, `wChen()`, `wBisquare()` and `wSELF()` for details.
	  – If `options$season$wFUN` or `options$season$wFUN` is `NULL`, the `options$wFUN` will overwrite it.
	• `wmin` : double, the minimum weigth of bads points (i.e. snow, ice and cloud).
	  – If `options$season$wmin` or `options$season$wmin` is `NULL`, the `options$wmin` will overwrite it.
	• `season` : See the following part: options for season for details.
	• `fitting` : See the following part: options for fitting for details.

`names`	vector of character, names of options

options for season

(a) Parameters for rough fitting:

- `rFUN` : character (default `smooth_wWHIT`), the name of rough curve fitting function, can be one of `c("smooth_wSG", "smooth_wWHIT", "smooth_wHANTS")`, which are corresponding to `smooth_wSG()`, `smooth_wWHIT()` and `smooth_wHANTS()`.

- wFUN : character (default wTSM), the name of weights updating functions, can be one of c("wTSM", "wChen", "wBisquare", "wSELF"). See wTSM(), wChen(), wBisquare() and wSELF() for details.
- iters : integer (default 2), the number of rough fitting iterations.
- wmin : double, the minimum weight of bad points (i.e. snow, ice and cloud).
- verbose : logical (default FALSE). If TRUE, options$season will be printed on the console.
- lambda : double (default NULL), the smoothing parameter of smooth_wWHIT().
  - If lambda = NULL, V-curve theory will be employed to find the optimal lambda. See lambda_vcurve() for details.
- frame : integer (default NULL), the parameter of smooth_wSG(), moving window size.
  - If frame = NULL, frame will be reset as floor(nptperyear/5)*2 + 1 (refered by TIME-SAT).
- nf : integer (default 4), the number of frequencies in smooth_wHANTS().
- maxExtendMonth: integer (default 12), previous and subsequent maxExtendMonth (in month) data were added to the current year for rough fitting.
- nextend : integer (default NULL), same as maxExtendMonth, but in points.
  - If nextend provided, maxExtendMonth will be ignored.
  - If nextend = NULL, nextend will be reset as ceiling(maxExtendMonth/12*nptperyear)

(b) Parameters for growing season division:

- minpeakdistance : double (default NULL), the minimum distance of two peaks (in points). If the distance of two maximum extreme value less than minpeakdistance, only the maximum one will be kept.
  - If minpeakdistance = NULL, it will be reset as nptperyear/6.
- r_max : double (default 0.2; in (0, 1)). r_max and r_min are used to eliminate fake peaks and troughs.
  - The real peaks should satisfy:
    1. $max(h_{peak,L}, h_{peak,R}) > r_{max}A$
    2. $min(h_{peak,L}, h_{peak,R}) > r_{min}A$, where $h_{peak,L}, h_{peak,R}$ are height difference from the peak to the left- and right-hand troughs.
  - The troughs should satisfy:
    1. $max(h_{trough,L}, h_{trough,R}) > r_{max}A$, where $h_{trough,L}, h_{trough,R}$ are height difference from the trough to the left- and right-hand peaks.
- r_min : double (default 0.05; in (0, 1)), see above r_max for details. r_min < r_max.
- rtrough_max : double (default 0.6, in (0, 1)), $y_{peak} <= rtrough_max * A + ylu[1]$.
- ypeak_min : double 0.1 (in VI unit), $y_{peak} >= ypeak_min$.
- .check_season : logical (default TRUE). check the growing season length according to len_min and len_max. If FALSE, len_min and len_max will lose their effect.
- len_min : integer (default 45), the minimum length (in days) of growing season
- len_max : integer (default 650), the minimum length (in days) of growing season
- adj.param : logical. If TRUE (default), if there are too many or too less peaks and troughs, phenofit will automatically adjust rough curve fitting function parameters. See MaxPeaksPerYear and MaxTroughsPerYear for details.
- MaxPeaksPerYear (optional) : integer (default 2), the max number of peaks per year. If PeaksPerYear > MaxPeaksPerYear, then lambda = lambda*2.

- `MaxTroughsPerYear` (optional) : integer (default 3), the max number of troughs per year. If `TroughsPerYear` > `MaxTroughsPerYear`, then `lambda` = `lambda*2`.
- `calendarYear` : logical (default `FALSE`). If `TRUE`, the start and end of a calendar year will be regarded as growing season division (North Hemisphere is from 01 Jan to 31 Dec; South Hemisphere is from 01 Jul to 30 Jun).
- `rm.closed` : logical (default `TRUE`). If `TRUE`, closed peaks (or troughs) will be further tidied. Only the maximum
- `is.continuous` (not used): logical (default `TRUE`). This parameter is for `fluxnet2015` fluxsite data, where the input might be not continuous.

options for fitting

- `methods` (default `c('AG', 'Beck', 'Elmore', 'Zhang')``): Fine curve fitting methods, can be one or more` 'Beck', 'Elmore', 'Zhang', 'Gu', 'Klos')`. Note that 'Gu' and 'Klos' are very slow.
- `iters` (default 2): max iterations of fine fitting.
- `wFUN` (default `wTSM`): Character or function, weights updating function of fine fitting function.
- `wmin` (default 0.1): min weights in the weights updating procedure.
- `use.rough` (default FALSE): Whether to use rough fitting smoothed time-series as input? If `false`, smoothed VI by rough fitting will be used for Phenological metrics extraction; If `true`, original input `y` will be used (rough fitting is used to divide growing seasons and update weights.
- `use.y0` (default TRUE): boolean. whether to use original `y0` as the input of `plot_input`, note that not for curve fitting. `y0` is the original value before the process of `check_input`.
- `nextend` (default 2): Extend curve fitting window, until `nextend` good or marginal points are found in the previous and subsequent growing season.
- `maxExtendMonth` (default 1): Search good or marginal good values in previous and subsequent `maxExtendMonth` period.
- `minExtendMonth` (default 0.5): Extend period defined by `nextend` and `maxExtendMonth`, should be no shorter than `minExtendMonth`. When all points of the input time-series are good value, then the extending period will be too short. In that situation, we can't make sure the connection between different growing seasons is smoothing.
- `minPercValid`: (default 0, not use). If the percentage of good- and marginal- quality points is less than `minPercValid`, curve fiting result is set to `NA`.
- `minT`: (not use). If `Tn` not provided in `INPUT`, `minT` will not be used. `minT` use night temperature Tn to define backgroud value (days with `Tn` < `minT` treated as ungrowing season).

Examples

```
set_options(verbose = FALSE)
get_options("season") %>% str()
```

smooth_wHANTS *Weighted HANTS SMOOTH*

Description

Weighted HANTS smoother

Usage

```
smooth_wHANTS(
  y,
  t,
  w,
  nf = 3,
  ylu,
  periodlen = 365,
  nptperyear,
  wFUN = wTSM,
  iters = 2,
  wmin = 0.1,
  ...
)
```

Arguments

y	Numeric vector, vegetation index time-series
t	Numeric vector, Date variable
w	(optional) Numeric vector, weights of y. If not specified, weights of all NA values will be wmin, the others will be 1.0.
nf	number of frequencies to be considered above the zero frequency
ylu	[low, high] of time-series y (curve fitting values are constrained in the range of ylu.
periodlen	length of the base period, measured in virtual samples (days, dekads, months, etc.). nptperyear in timesat.
nptperyear	Integer, number of images per year.
wFUN	weights updating function, can be one of 'wTSM', 'wChen' and 'wBisquare'.
iters	How many times curve fitting is implemented.
wmin	Double, minimum weigth (i.e. weight of snow, ice and cloud).
...	Additional parameters are passed to wFUN.

Value

- ws: weights of every iteration
- zs: curve fittings of every iteration

Author(s)

Wout Verhoef, NLR, Remote Sensing Dept. June 1998 Mohammad Abouali (2011), Converted to MATLAB Dongdong Kong (2018), introduced to R and modified into weighted model.

Examples

```
library(phenofit)
data("MOD13A1")
dt <- tidy_MOD13(MOD13A1$dt)
d <- dt[site == "AT-Neu", ]

l <- check_input(d$t, d$y, d$w, nptperyear=23)
r_wHANTS <- smooth_wHANTS(l$y, l$t, l$w, ylu = l$ylu, nptperyear = 23, iters = 2)
```

---

smooth_wSG *Weighted Savitzky-Golay*

Description

Weighted Savitzky-Golay

Usage

```
smooth_wSG(
  y,
  w,
  nptperyear,
  ylu,
  wFUN = wTSM,
  iters = 2,
  frame = floor(nptperyear/5) * 2 + 1,
  d = 2,
  ...
)
```

Arguments

y	Numeric vector, vegetation index time-series
w	(optional) Numeric vector, weights of y. If not specified, weights of all NA values will be wmin, the others will be 1.0.
nptperyear	Integer, number of images per year.
ylu	(optional) [low, high] value of time-series y (curve fitting values are constrained in the range of ylu.
wFUN	weights updating function, can be one of 'wTSM', 'wChen' and 'wBisquare'.
iters	How many times curve fitting is implemented.
frame	Savitzky-Golay windows size

d	polynomial of degree. When d = 1, it becomes moving average.
...	Additional parameters are passed to wFUN.

Value

- ws: weights of every iteration
- zs: curve fittings of every iteration

References

1. Chen, J., J\"onsson, P., Tamura, M., Gu, Z., Matsushita, B., Eklundh, L., 2004. A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky-Golay filter. Remote Sens. Environ. 91, 332-344. https://doi.org/10.1016/j.rse.2004.03.014.

2. https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter

Examples

```
library(phenofit)
data("MOD13A1")
dt <- tidy_MOD13(MOD13A1$dt)
d <- dt[site == "AT-Neu", ]

l <- check_input(d$t, d$y, d$w, nptperyear=23)
r_wSG <- smooth_wSG(l$y, l$w, l$ylu, nptperyear = 23, iters = 2)
```

---

smooth_wWHIT *Weigthed Whittaker Smoother*

Description

Weigthed Whittaker Smoother

Usage

```
smooth_wWHIT(
  y,
  w,
  ylu,
  nptperyear,
  wFUN = wTSM,
  iters = 1,
  lambda = 15,
  second = FALSE,
  ...
)
```

Arguments

`y`	Numeric vector, vegetation index time-series
`w`	(optional) Numeric vector, weights of `y`. If not specified, weights of all `NA` values will be `wmin`, the others will be 1.0.
`ylu`	`[low, high]` of time-series y (curve fitting values are constrained in the range of `ylu`.
`nptperyear`	Integer, number of images per year.
`wFUN`	weights updating function, can be one of 'wTSM', 'wChen' and 'wBisquare'.
`iters`	How many times curve fitting is implemented.
`lambda`	scaler or numeric vector, whittaker parameter. • If `lambda` = `NULL`, `V-curve` theory will be applied to retrieve the optimal `lambda`. • If multiple `lambda` provided (numeric vector), a list of the smoothing results with the same length of `lambda` will be returned.
`second`	If true, in every iteration, Whittaker will be implemented twice to make sure curve fitting is smooth. If curve has been smoothed enough, it will not care about the second smooth. If no, the second one is just prepared for this situation. If lambda value has been optimized, second smoothing is unnecessary.
`...`	Additional parameters are passed to `wFUN`.

Value

- `ws`: weights of every iteration
- `zs`: curve fittings of every iteration

Note

Whittaker smoother of the second order difference is used!

References

1. Eilers, P.H.C., 2003. A perfect smoother. Anal. Chem. doi:10.1021/ac034173t
2. Frasso, G., Eilers, P.H.C., 2015. L- and V-curves for optimal smoothing. Stat. Modelling 15, 91-111. doi:10.1177/1471082X14549288.

See Also

`lambda_vcurve()`

Examples

```
data("MOD13A1")
dt <- tidy_MOD13(MOD13A1$dt)
d <- dt[site == "AT-Neu", ]

l <- check_input(d$t, d$y, d$w, nptperyear=23)
```

```
r_wWHIT <- smooth_wWHIT(l$y, l$w, l$ylu, nptperyear = 23, iters = 2)

## Optimize `lambda` by V-curve theory
# (a) optimize manually
lambda_vcurve(l$y, l$w, plot = TRUE)

# (b) optimize automatically by setting `lambda = NULL` in smooth_wWHIT
r_wWHIT2 <- smooth_wWHIT(l$y, l$w, l$ylu, nptperyear = 23, iters = 2, lambda = NULL) #
```

---

whit2 *Weighted Whittaker smoothing with a second order finite difference penalty*

Description

This function smoothes signals with a finite difference penalty of order 2. This function is modified from `ptw` package.

Usage

```
whit2(y, lambda, w = rep(1, ny))
```

Arguments

`y`	signal to be smoothed: a vector
`lambda`	smoothing parameter: larger values lead to more smoothing
`w`	weights: a vector of same length as y. Default weights are equal to one

Value

A numeric vector, smoothed signal.

Author(s)

Paul Eilers, Jan Gerretzen

References

1. Eilers, P.H.C. (2004) "Parametric Time Warping", Analytical Chemistry, **76** (2), 404 – 411.

2. Eilers, P.H.C. (2003) "A perfect smoother", Analytical Chemistry, **75**, 3631 – 3636.

Examples

```
library(phenofit)
data("MOD13A1")
dt <- tidy_MOD13(MOD13A1$dt)
y <- dt[site == "AT-Neu", ][1:120, y]

plot(y, type = "b")
lines(whit2(y, lambda = 2), col = 2)
lines(whit2(y, lambda = 10), col = 3)
lines(whit2(y, lambda = 100), col = 4)
legend("bottomleft", paste("lambda = ", c(2, 10, 15)), col = 2:4, lty = rep(1, 3))
```

`wSELF` *Weight updating functions*

Description

- `wSELF` weigth are not changed and return the original.
- `wTSM` weight updating method in TIMESAT.
- `wBisquare` Bisquare weight update method. wBisquare has been modified to emphasis on upper envelope.
- `wBisquare0` Traditional Bisquare weight update method.
- `wChen` Chen et al., (2004) weight updating method.
- `wBeck` Beck et al., (2006) weigth updating method. wBeck need sos and eos input. The function parameter is different from others. It is still not finished.

Usage

```
wSELF(y, yfit, w, ...)

wTSM(y, yfit, w, iter = 2, nptperyear, wfact = 0.5, ...)

wBisquare0(y, yfit, w, ..., wmin = 0.2)

wBisquare(y, yfit, w, ..., wmin = 0.2, .toUpper = TRUE)

wChen(y, yfit, w, ..., wmin = 0.2)

wKong(y, yfit, w, ..., wmin = 0.2)
```

Arguments

`y`	Numeric vector, vegetation index time-series
`yfit`	Numeric vector curve fitting values.
`w`	(optional) Numeric vector, weights of `y`. If not specified, weights of all `NA` values will be `wmin`, the others will be 1.0.

`...`	other parameters are ignored.
`iter`	iteration of curve fitting.
`nptperyear`	Integer, number of images per year.
`wfact`	weight adaptation factor (0-1), equal to the reciprocal of 'Adaptation strength' in TIMESAT.
`wmin`	Double, minimum weight of bad points, which could be smaller the weight of snow, ice and cloud.
`.toUpper`	Boolean. Whether to approach the upper envelope?

Value

wnew Numeric Vector, adjusted weights.

Author(s)

wTSM is implemented by Per J\"onsson, Malm\"o University, Sweden <per.jonsson@ts.mah.se> and Lars Eklundh, Lund University, Sweden <lars.eklundh@nateko.lu.se>. And Translated into Rcpp by Dongdong Kong, 01 May 2018.

References

1. Per J\"onsson, P., Eklundh, L., 2004. TIMESAT - A program for analyzing time-series of satellite sensor data. Comput. Geosci. 30, 833-845. https://doi.org/10.1016/j.cageo.2004.05.006.

2. https://au.mathworks.com/help/curvefit/smoothing-data.html#bq_6ys3-3

3. Garcia, D., 2010. Robust smoothing of gridded data in one and higher dimensions with missing values. Computational statistics & data analysis, 54(4), pp.1167-1178.

4. Chen, J., J\"onsson, P., Tamura, M., Gu, Z., Matsushita, B., Eklundh, L., 2004. A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky-Golay filter. Remote Sens. Environ. 91, 332-344. https://doi.org/10.1016/j.rse.2004.03.014.

5. Beck, P.S.A., Atzberger, C., Hogda, K.A., Johansen, B., Skidmore, A.K., 2006. Improved monitoring of vegetation dynamics at very high latitudes: A new method using MODIS NDVI. Remote Sens. Environ. https://doi.org/10.1016/j.rse.2005.10.021

6. https://github.com/kongdd/phenopix/blob/master/R/FitDoubleLogBeck.R

Index