Package 'phenofit'

April 2, 2020

Type Package

Title Extract Remote Sensing Vegetation Phenology

Version 0.2.7

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:
Dongdong Kong, R package: A state-of-the-art Vegetation Phenology extraction package, phenofit version 0.2.3, <https://github.com/kongdd/phenofit>;
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 7.1.0

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.1)

Imports Rcpp, tibble, dplyr, purrr, stringr, tidyr, ggplot2, lubridate, data.table, spam, grid, gridExtra, magrittr, plyr, reshape2, zoo, optimx, ucminf, numDeriv, grDevices, utils, stats, shiny, jsonlite, foreach, iterators, JuliaCall, methods

Suggests knitr, rmarkdown, testthat (>= 2.1.0)

URL `https://github.com/kongdd/phenofit`

BugReports `https://github.com/kongdd/phenofit/issues`

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 2020-04-02 11:40:03 UTC

R topics documented:

`add_HeadTail` *Add one year data in the head and tail*

Description

Add the data of the year of `year_start -1` to the head, add the data of the year of `year_end -1` to the tail.

Usage

```
add_HeadTail(d, south = FALSE, nptperyear, trs = 0.45)
```

Arguments

`d`	A data.table, should have `t` (compositing date) or `date` (image date) column which are (`Date` variable).
`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.
`nptperyear`	Integer, number of images per year.
`trs`	If nmissing < trs*nptperyear (little missing), this year is include to extract phenology; if `FALSE`, this year is excluded.

Value

data.table

Note

`date` is image date; `t` is compositing date.

Examples

```
library(phenofit)
data("MOD13A1")

dt <- tidy_MOD13.gee(MOD13A1$dt)
st <- MOD13A1$st

sitename <- dt$site[1]
d     <- dt[site == sitename, ] # get the first site data
sp    <- st[site == sitename, ] # station point

nptperyear = 23
dnew     <- add_HeadTail(d, nptperyear = nptperyear) # add one year in head and tail
```

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,
  Tn = NULL,
  wmin = 0.2,
  wsnow = 0.8,
  ymin,
  missval,
  maxgap,
  alpha = 0.02,
  alpha_high = NULL,
  date_start = NULL,
  date_end = NULL,
  mask_spike = TRUE,
  ...
)
```

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","aer` others will be categoried into `others`. `QC_flag` is used for visualization in `get_pheno()` and `plot_phenofit()`.
`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.
`Tn`	Numeric vector, night temperature, default is null. If provided, Tn is used to help divide ungrowing period, and then get background value in ungrowing season (see details in `phenofit::backval()`).

`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 value in [0,1], quantile prob of ylu_min.
`alpha_high`	Double value in [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?
`...`	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)`.

See Also

`phenofit::backval()`

Examples

```
library(phenofit)
data("MOD13A1")

df <- tidy_MOD13.gee(MOD13A1$dt)
st <- MOD13A1$st

date_start <- as.Date('2013-01-01')
date_end   <- as.Date('2016-12-31')

sitename <- 'CA-NS6' # df$site[1]
d     <- df[site == sitename & (date >= date_start & date <= date_end), ]
sp    <- st[site == sitename, ]
south <- sp$lat < 0
nptperyear <- 23

# global parameter
IsPlot = TRUE
print  = FALSE
ypeak_min  = 0.05
wFUN = wTSM

# add one year in head and tail
dnew     <- add_HeadTail(d, south = south, nptperyear = nptperyear)
INPUT    <- check_input(dnew$t, dnew$y, dnew$w, QC_flag = dnew$QC_flag,
     nptperyear = nptperyear, south = south,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
```

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"),
  ...
)
```

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')`.
`...`	other parameters passed to curve fitting function.

Value

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

Note

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

See Also

`fFITs()`, `FitDL.AG()`, `FitDL.Beck()`, `FitDL.Elmore()`, `FitDL.Gu()`, `FitDL.Klos()`, `FitDL.Zhang()`

Examples

```
library(phenofit)
# 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)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)

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

---

curvefits *Fine Curve fitting*

Description

Fine Curve fitting for INPUT time-series.

Usage

```
curvefits(
  INPUT,
  brks,
  wFUN = wTSM,
  iters = 2,
  wmin = 0.1,
  nextend = 2,
  maxExtendMonth = 2,
  minExtendMonth = 1,
  minT = 0,
  methods = c("AG", "Beck", "Elmore", "Gu", "Klos", "Zhang"),
  minPercValid = 0.2,
  print = TRUE,
  use.rough = FALSE,
  use.y0 = TRUE,
  ...
)
```

Arguments

`INPUT`	A list object with the elements of 't', 'y', 'w', 'Tn' (option) 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 dividing information.
`wFUN`	weights updating function, can be one of `wTSM()`, `wChen()`, `wBisquare()` and `wSELF()`.
`iters`	How many times curve fitting is implemented.
`wmin`	Double, minimum weigth (i.e. weight of snow, ice and cloud).
`nextend`	Extend curve fitting window, until `nextend` good or marginal element are found in previous and subsequent growing season.
`maxExtendMonth`	Search good or marginal good values in previous and subsequent `maxExtendMonth` period.
`minExtendMonth`	Extending perid 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.
`minT`	Double, use night temperature Tn to define backgroud value. Tn < minT is treated as ungrowing season.
`methods`	Fine curve fitting methods, can be one or more of `c('AG','Beck','Elmore','Gu','Klos','Zhang')`.
`minPercValid`	If the percentage of good and marginal quality points is less than `minPercValid`, curve fiting result is set to `NA`.
`print`	Whether to print progress information?
`use.rough`	Whether to use rough fitting smoothed time-series as input?
`use.y0`	boolean. whether to use original `y0`, which is before the process of `check_input`.
`...`	Other parameters will be ignore.

Value

fits Multiple phenofit object.

Examples

```
library(phenofit)
data("MOD13A1")

df <- tidy_MOD13.gee(MOD13A1$dt)
st <- MOD13A1$st

date_start <- as.Date('2013-01-01')
date_end   <- as.Date('2016-12-31')

sitename <- 'CA-NS6' # df$site[1]
d     <- df[site == sitename & (date >= date_start & date <= date_end), ]
sp    <- st[site == sitename, ]
```

```
south <- sp$lat < 0
nptperyear <- 23

# global parameter
IsPlot = TRUE
print  = FALSE
ypeak_min  = 0.05
wFUN = wTSM

# add one year in head and tail
dnew     <- add_HeadTail(d, south = south, nptperyear = nptperyear)
INPUT    <- check_input(dnew$t, dnew$y, dnew$w, QC_flag = dnew$QC_flag,
     nptperyear = nptperyear, south = south,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
# Rough fitting and growing season dividing
brks2 <- season_mov(INPUT,
    rFUN = smooth_wWHIT, wFUN = wFUN,
    plotdat = d, IsPlot = IsPlot, print = FALSE, IsPlot.OnlyBad = FALSE)
# Fine fitting
fit <- curvefits(
    INPUT, brks2,
    methods = c("AG", "Beck", "Elmore", "Zhang"), #,"klos", "Gu"
    wFUN = wFUN,
    nextend = 2, maxExtendMonth = 2, minExtendMonth = 1, minPercValid = 0.2,
    print = TRUE, verbose = FALSE)
```

findpeaks *findpeaks*

Description

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

Usage

```
findpeaks(
  x,
  IsDiff = TRUE,
  nups = 1,
  ndowns = nups,
  zero = "0",
  peakpat = NULL,
  minpeakheight = -Inf,
  minpeakdistance = 1,
  y_min = 0,
  y_max = 0,
  npeaks = 0,
  sortstr = FALSE,
  IsPlot = F
)
```

Arguments

`x`	Numeric vector.
`IsDiff`	If want to find extreme values, `IsDiff` should be true; If just want to find the continue negative or positive values, just set `IsDiff` as false.
`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.
`y_min`	Threshold is defined as the difference of peak value with trough value. There are two threshold (left and right). The minimum threshold should be greater than `y_min`.
`y_max`	Similar as `y_min`, The maximum threshold should be greater than `y_max`.
`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?
`IsPlot`	Boolean.

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, y_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, ...)

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

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

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

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

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

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
`...`	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/kongdd/phenopix/blob/master/R/FitDoubleLogGu.R

Examples

```
library(phenofit)
# 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)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang")

r <- FitDL.AG(y, t, tout)
plot(t, y)
lines(tout, r$zs$iter2, col = "red")
legend('topright', c('Original time-series', 'AG smoothed'),
    lty = c(0, 1), pch = c(16, NA), col = c("black", "red"))
```

`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)
```

`getBits` *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`
- `qc_NDVI3g`: For NDVI3g
- `qc_NDVIv4`: For 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_5l(QA, wmin = 0.2, wmid = 0.5, wmax = 1)

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

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

qc_NDVIv4(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")`

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

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)
```

---

get_fitting *getFittings*

Description

Get curve fitting data.frame

Usage

```
get_fitting(fit)

get_fitting.fFITs(fFITs)
```

Arguments

fit	Object returned by curvefits.
fFITs	fFITs object returned by curvefit().

Examples

```
library(phenofit)
# 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)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fFITs <- curvefit(y, t, tout, methods)
# multiple years
fits  <- list(`2001` = fFITs, `2002` = fFITs)

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(fit)

get_GOF.fFITs(fFITs)
```

Arguments

`fit`	Object returned by `curvefits`.
`fFITs`	`fFITs` object returned by `curvefit()`.

Value

- `meth`: The name of fine curve fitting method
- `RMSE`: Root Mean Square Error
- `NSE` : Nash-Sutcliffe model efficiency coefficient
- `R` : Pearson-Correlation
- `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
fFUN = 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 <- fFUN(par, t)
```

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

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(fits)

get_param.fFITs(fFITs)
```

Arguments

fits	Multiple methods curve fitting results by curvefits result.
fFITs	fFITs object returned by curvefit().

Examples

```
library(phenofit)
# 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)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fFITs <- curvefit(y, t, tout, methods)
# multiple years
fits  <- list(`2001` = fFITs, `2002` = fFITs)

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(
  fits,
  method,
  TRS = c(0.2, 0.5, 0.6),
  analytical = TRUE,
  smoothed.spline = FALSE,
  IsPlot = FALSE,
  showName_fitting = TRUE,
  ...
)

get_pheno.fFITs(
  fFITs,
  method,
  TRS = c(0.2, 0.5),
  analytical = TRUE,
  smoothed.spline = FALSE,
  IsPlot = FALSE,
  title_left = "",
  showName_pheno = TRUE
)
```

Arguments

`fits`	A list of `fFITs()` object, for a single curve fitting method.
`method`	Which fine curve fitting method to be extracted?
`TRS`	Threshold for `PhenoTrs`.
`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?
`showName_fitting`	Whether to show the name of fine curve fitting method in top title?
`...`	ignored.
`fFITs`	`fFITs` object returned by `curvefit()`.

`title_left`	String of growing season flag.
`showName_pheno`	Whether to show names of phenological methods in top title? Generally, only show top title in the first row.

Value

List of every year phenology metrics

Note

Please note that only a single fine curve fitting method allowed here!

Examples

```
library(phenofit)
# 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)
t    <- seq(1, 365, 8)
tout <- seq(1, 365, 1)
y <- fFUN(par, t)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fFITs <- curvefit(y, t, tout, methods)
# multiple years
fits  <- list(`2001` = fFITs, `2002` = fFITs)

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)
```

`Logistic` *Double logistics functions*

Description

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

- `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.

Usage

```
Logistic(par, t)

doubleLog.Zhang(par, t)

doubleLog.AG(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

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

References

Peter M. Atkinson, et al., 2012, RSE, 123:400-417

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,
  verbose = 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', 'FitDL.Gu' a
`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', 'nlm', 'nlminb', 'ucminf' 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'.
`verbose`	Whether to display intermediate variables?
`...`	other parameters passed to `I_optim()` or `I_optimx()`.

Value

fFIT object, see `fFIT()` for details.

See Also

`FitDL()`

Examples

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

# 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)
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 <- 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)

# 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.frame() %>% cbind(t, y, .)
pdat <- reshape2::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), size = 0.9)
```

opt_nlminb_julia *Optimization using PORT routines*

Description

Unconstrained and box-constrained optimization using PORT routines.

Usage

```
opt_nlminb_julia(
  par0,
  fitMeth = "doubleLog_Beck",
  y,
  t,
  w = NULL,
  ylu = NULL,
  lower = NULL,
  upper = NULL,
  ...
)
```

Arguments

`par0`	Initial values for the parameters to be optimized over.
`fitMeth`	Curve fitting methods, one of `c("doubleLog_Beck","doubleLog_Elmore","doubleLog_AG","doubleI`
`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.
`ylu`	ymin, ymax, which is used to force `ypred` in the range of `ylu`.
`lower`	vectors of lower and upper bounds, replicated to be as long as `start`. If unspecified, all parameters are assumed to be unconstrained.
`upper`	vectors of lower and upper bounds, replicated to be as long as `start`. If unspecified, all parameters are assumed to be unconstrained.
`...`	ignored parameters

Value

A list object of

- `par`: The optimal parameters
- `convergence`:
  - 0: convergent;
  - 1: Non-convergent

- iterations
- evaluations: list(function, gradient)
- objective

See Also

`stats::nlminb()`

Examples

```
## Not run:

t    = seq(1.0, 366, 8)
fun  = doubleLog_Beck
par  = c(0.1 , 0.7, 50, 0.1, 250, 0.1)
par0 = c(0.05, 0.6 , 45, 0.1, 200, 0.2)

ypred = t*0
y     = fun(par, t)

julia_init()
r_julia  <- opt_nlminb_julia(par0, "doubleLog_Beck", y, t)
r_R <- opt_nlminb(par0, f_goal, fun = fun, y = y, t = t, pred = ypred)

list(julia = r_julia, R = r_R) %>%
    map(~c(.$par, .$objective, .$value)) %>%
    do.call(rbind, .)# %>%

n <- length(t)
w <- rep(0.2, n)
# julia is 5 times faster
{
    # microbenchmark::microbenchmark : 18.939826 ms in R
    info <- rbenchmark::benchmark(
        r1 <- opt_nlminb_julia(par0, "doubleLog_Beck", y, t, w),
        r2 <- opt_nlminb(par0, f_goal, fun = fun, y = y, t = t, pred = ypred),
        replications = 500
    )
    print(info)
}


## End(Not run)
```

`PhenoExtractMeth` *Phenology Extraction methods*

Description

- `PhenoTrs` Threshold method
- `PhenoDeriv` Derivative method
- `PhenoGu` Gu method
- `PhenoKl` Inflection method

Usage

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

PhenoDeriv(
  fFIT,
  analytical = TRUE,
  smoothed.spline = FALSE,
  IsPlot = TRUE,
  show.lgd = TRUE,
  ...
)

PhenoGu(fFIT, analytical = TRUE, smoothed.spline = FALSE, IsPlot = TRUE, ...)

PhenoKl(
  fFIT,
  analytical = TRUE,
  smoothed.spline = FALSE,
  IsPlot = TRUE,
  show.lgd = TRUE,
  ...
)
```

Arguments

`fFIT`	`fFIT` object returned by `optim_pheno()`.
`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
`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?
`show.lgd`	whether show figure lelend?

Examples

```
library(phenofit)
# 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)

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

methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # "Klos" too slow
fFITs <- curvefit(y, t, tout, methods)
fFIT  <- fFITs$fFIT$AG

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

`phenofit_plot` *phenofit_plot*

Description

phenofit_plot

Usage

```
phenofit_plot(
  obj,
  type = "all",
  methods,
  title = NULL,
  title.ylab = "Vegetation Index",
  IsPlot = TRUE,
  show.legend = TRUE,
  newpage = TRUE
)
```

Arguments

`obj`	`fFIT`
`type`	one of c("season", "fitting", "pheno", "all")
`methods`	Fine curve fitting methods, can be one or more of `c('AG','Beck','Elmore','Gu','Klos','Zhang')`.
`title`	String, title of figure.
`title.ylab`	String, title of `xlab` and `ylab`.
`IsPlot`	boolean. If false, a ggplot object will be returned.
`show.legend`	If now show legend, ggplot object will be returned, else grid object will be returned.
`newpage`	boolean, whether draw figure in a new page?

`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, ...)
```

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()`?
`...`	other parameter will be ignored.

Examples

```
library(phenofit)
data("MOD13A1")

dt <- tidy_MOD13.gee(MOD13A1$dt)
st <- MOD13A1$st

sitename <- dt$site[1]
d     <- dt[site == sitename, ] # get the first site data
sp    <- st[site == sitename, ] # station point
# global parameter
IsPlot = TRUE
print  = FALSE
```

```
nptperyear = 23
ypeak_min  = 0.05

dnew     <- add_HeadTail(d, nptperyear = nptperyear) # add one year in head and tail
INPUT    <- check_input(dnew$t, dnew$y, dnew$w, d$QC_flag, nptperyear,
                        maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
plot_input(INPUT)
```

`plot_phenofit` *plot_phenofit*

Description

plot_phenofit

Usage

```
plot_phenofit(
  d_fit,
  seasons,
  d_obs = NULL,
  title = NULL,
  title.xlab = "Time",
  title.ylab = "Vegetation Index",
  font.size = 14,
  theme = NULL,
  cex = 2,
  shape = "point",
  angle = 30,
  show.legend = TRUE
)
```

Arguments

`d_fit`	data.frame of curve fittings returned by `get_fitting()`.
`seasons`	Growing season dividing 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.
`title.xlab, title.ylab`	String, title of `xlab` and `ylab`.
`font.size`	Font size of axis.text
`theme`	ggplot theme to be applied
`cex`	point size for VI observation.
`shape`	the shape of input VI observation? `line` or `point`
`angle`	`text.x` angle
`show.legend`	Boolean

Examples

```
library(phenofit)
data("MOD13A1")

df <- tidy_MOD13.gee(MOD13A1$dt)
st <- MOD13A1$st

date_start <- as.Date('2013-01-01')
date_end   <- as.Date('2016-12-31')

sitename <- 'CA-NS6' # df$site[1]
d     <- df[site == sitename & (date >= date_start & date <= date_end), ]
sp    <- st[site == sitename, ]
south <- sp$lat < 0
nptperyear <- 23

# global parameter
IsPlot = TRUE
print  = FALSE
ypeak_min  = 0.05
wFUN = wTSM

# add one year in head and tail
dnew     <- add_HeadTail(d, south = south, nptperyear = nptperyear)
INPUT    <- check_input(dnew$t, dnew$y, dnew$w, QC_flag = dnew$QC_flag,
     nptperyear = nptperyear, south = south,
     maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)
# Rough fitting and growing season dividing
brks2 <- season_mov(INPUT,
    rFUN = smooth_wWHIT, wFUN = wFUN,
    plotdat = d, IsPlot = IsPlot, print = FALSE, IsPlot.OnlyBad = FALSE)
# Fine fitting
fit <- curvefits(
    INPUT, brks2,
    methods = c("AG", "Beck", "Elmore", "Zhang"), #,"klos", "Gu"
    wFUN = wFUN,
    nextend = 2, maxExtendMonth = 2, minExtendMonth = 1, minPercValid = 0.2,
    print = TRUE, verbose = FALSE)
## visualization
df_fit <- get_fitting(fit)
g <- plot_phenofit(df_fit, brks2)
grid::grid.newpage(); grid::grid.draw(g)
```

`plot_season` *plot_season*

Description

Plot growing season divding result.

Usage

```
plot_season(
  INPUT,
  brks,
  plotdat,
  ylu,
  IsPlot.OnlyBad = FALSE,
  show.legend = TRUE,
  title = NULL
)
```

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.
`ylu`	[low, high] of time-series y (curve fitting values are constrained in the range of `ylu`.
`IsPlot.OnlyBad`	If true, only plot partial figures whose NSE < 0.3.
`show.legend`	Whether to show legend?
`title`	The main title (on top)

`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)
```

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.gee(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/7) * 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.gee(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`	whittaker parameter (2-15 is suitable for 16-day VI). Multiple lambda values also are accept, then a list object return.
`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

References

1. Eilers, P.H.C., 2003. A perfect smoother. Anal. Chem. https://doi.org/10.1021/ac034173t

2. Frasso, G., Eilers, P.H.C., 2015. L- and V-curves for optimal smoothing. Stat. Modelling 15, 91–111. https://doi.org/10.1177/1471082X14549288

Examples

```
library(phenofit)
data("MOD13A1")
dt <- tidy_MOD13.gee(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)
```

`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.gee(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