Package ‘phenofit’

June 7, 2019

Type     Package
Title    Extract Remote Sensing Vegetation Phenology
Version  0.2.5-2
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  
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>;  

License GPL-2 | file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 6.1.1
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
Suggests knitr, rmarkdown, testthat

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

BugReports https://github.com/kongdd/phenofit/issues
NeedsCompilation yes

1
Author  Dongdong Kong [aut, cre],
        Jianjian Cui [aut],
        Mingzhong Xiao [aut],
        Yongqiang Zhang [aut],
        Xihui Gu [aut]
Maintainer  Dongdong Kong <kongdd.sysu@gmail.com>
Repository  CRAN
Date/Publication  2019-06-07 08:00:08 UTC

R topics documented:

add_HeadTail .................................................. 3
backval .......................................................... 4
check_input ..................................................... 5
check_ylu ......................................................... 7
curvefit .......................................................... 8
curvefits ........................................................ 9
cv_coef ........................................................... 10
D ................................................................. 11
fFIT ............................................................... 13
fFITs .............................................................. 13
findpeaks .......................................................... 14
FitDL .............................................................. 15
f_goal ............................................................. 17
getBits ............................................................. 18
getRealDate ...................................................... 19
get_fitting ....................................................... 20
get_GOF ........................................................... 21
get_param ......................................................... 22
get_pheno ........................................................ 23
GOF ............................................................... 24
init_lambda ....................................................... 25
init_param ........................................................ 26
I_optim ............................................................ 27
kurtosis ............................................................ 29
Logistic ............................................................ 29
MOD13A1 ........................................................... 30
movmean ........................................................... 31
optim_pheno ........................................................ 32
opt_FUN ........................................................... 33
PhenoExtractMeth .................................................. 35
phenofit_loaddata .................................................. 36
phenofit_plot ..................................................... 37
phenofit_shiny .................................................... 39
plot_input ........................................................ 38
plot_phenofit ..................................................... 39
plot_season ....................................................... 40
**Description**

Add one year data in the head and tail

**Usage**

```r
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

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

dt <- tidy_MOD13.kee(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
```

Description

Calculate background values for vegetation index.

Usage

`backval(y, t, w, Tn, minT = 5, nptperyear, ...)`

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.
- `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()`).
- `minT`: min temperature for growing season.
- `nptperyear`: Integer, number of images per year.
- `...`: Others will be ignored.

Details

Night temperature Tn >= Tmin (default 5 degree) defined as raw growing season. Background value is determined from two neighboring vegetation in raw growing season by assuming that the background and vegetation abundance could remain the same during a consecutive two year period. Details can be seen in Zhang et al., (2015).
Value

back If back value is NA, it is impossible to extract phenology here.

Note

This function only works in every growing season.

References


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, ymin, missval, maxgap, alpha = 0.02, ...)

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 categorized 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 pheno::backval()).
check_input

wmin  Double, minimum weight of bad points, which could be smaller the weight of snow, ice and cloud.

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 value 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 continuous missing segments, e.g. in the start or end of growing season.

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

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%, then w_critical=1.
  The else, if the percentage of w >= 0.5 points is greater than 10%, then w_critical=0.5.
  In boreal regions, even if the percentage of w >= 0.5 points is only 10%, we still can’t set w_critical=wmin.
  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)
check_ylu

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$t, dnew>y, dnew>y, QC_flag = dnew$QC_flag,
nptperyear = nptperyear, south = south,
maxgap = nptperyear/4, alpha = 0.02, wmin = 0.2)

---

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, \([y_{min}, y_{max}]\)

Value

yfit, the numeric vector in the range of ylu.

Examples

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

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

---

**Description**

Fine Curve fitting for INPUT time-series.

**Usage**

```r
curvefits(INPUT, brks, wFUN = wTSM, iters = 2, wmin = 0.2,
  nextend = 2, maxExtendMonth = 3, minExtendMonth = 1, minT = 0,
  methods = c("AG", "Beck", "Elmore", "Gu", "Klos", "Zhang"),
  minPercValid = 0.2, print = TRUE, use.rough = FALSE, ...)
```

**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 weight (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 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.
- **minT**: Double, use night temperature Tn to define background 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 fitting result is set to NA.
- **print**: Whether to print progress information?
- **use.rough**: Whether to use rough fitting smoothed time-series as input?
- **...**: Other parameters will be ignore.
Value

fits Multiple phenofit object.

Examples

library(phenofit)
data("MOD13A1")

df <- tidy_MOD13.ggee(MOD13A1$dt)
st <- MOD13A1$st
date_start <- as.Date('2013-01-01')
date_end <- as.Date('2016-12-31')
sitename <- 'CA-N56' # 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$, dnew$, dnew$, QC_flag = dnew$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 = 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)

---

**cv_coef**  

**weighted CV**

Description

weighted CV
Usage

\texttt{cv_coef(x, w)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1em} \text{Numeric vector}
  \item \texttt{w} \hspace{1em} \text{weights of different point}
\end{itemize}

Value

Named numeric vector, (mean, sd, cv).

Examples

\begin{verbatim}
library(phenofit)
x = rnorm(100)
coefs <- cv_coef(x)
\end{verbatim}

Description

Get derivative of \texttt{phenofit} object. D1 first order derivative, D2 second order derivative, curvature.

Usage

\begin{verbatim}
D1(fit, analytical = TRUE, smoothed.spline = FALSE, ...)
D2(fit, analytical = TRUE, smoothed.spline = FALSE, ...)
## S3 method for class 'ffIT'
D1(fit, analytical = TRUE, smoothed.spline = FALSE, ...)
## S3 method for class 'ffIT'
D2(fit, analytical = TRUE, smoothed.spline = FALSE, ...)
curvature(fit, analytical = TRUE, smoothed.spline = FALSE, ...)
## S3 method for class 'ffIT'
curvature(fit, analytical = TRUE,
  smoothed.spline = FALSE, ...)
\end{verbatim}
Arguments

fit A curve fitting object returned by `curvefit`.

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?

... Other parameters will be ignored.

details

If `fit$fun` has no gradient function or `smoothed.spline = TRUE`, time-series smoothed by spline first, and get derivatives at last. If `fit$fun` exists and `analytical = TRUE`, `smoothed.spline` will be ignored.

Value

- `der1` First order derivative
- `der2` Second order derivative
- `k` Curvature

Examples

```r
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
d1 <- D1(ffIT)
d2 <- D2(ffIT)
d_k <- curvature(ffIT)
```
**fFIT**

S3 class of fine curve fitting object.

---

**Description**

fFIT is returned by `optim_pheno()`.

**Format**

- `tout`: Corresponding doy of prediction
- `zs`: curve fitting values of every iteration
- `ws`: weight of every iteration
- `par`: Optimized parameter of fine curve fitting method
- `fun`: The name of fine curve fitting function.

---

**ffITs**

S3 class of multiple fine curve fittings object.

---

**Description**

plot curve fitting VI, gradient (first order difference D1), hessian (D2), curvature (k) and the change rate of curvature(der.k)

**Usage**

```r
## S3 method for class 'ffITs'
plot(x, method, ...)
```

**Arguments**

- `x`: Fine curve fitting object `fFITs()` returned by `curvefit()`.
- `method`: Which fine curve fitting method to be extracted?
- `...`: ignored.

**Examples**

```r
library(phenofit)
# simulate vegetation time-series
ffUN = doubleLog.Beck
par = c(
  mn = 0.1,
  mx = 0.7,
  sos = 50,
  rsp = 0.1,
```
findpeaks

Description

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

Usage

```r
findpeaks(x, IsDiff = TRUE, nups = 1, ndowns = nups, zero = "\0", peakpat = NULL, minpeakheight = -Inf, minpeakdistance = 1,
          r_min = 0, r_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.
- `r_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 r_min.
- `r_max`: Similar as r_min, The maximum threshold should be greater than r_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 order 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)
ht <- 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, r_min=4, sortstr=TRUE)
points(val=pos, x$x, pch=20, col="maroon")

FitDL

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, ...)
FitAG(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 parameters 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


Examples

```r
library(phenofit)
# simulate vegetation time-series
fFUN = doubleLogBeck
par = c(
    mn = 0.1,
    mx = 0.7,
    sos = 50,
    rsp = 0.1,
    eos = 250,
    rau = 0.1)
```
f_goal

Goal function of fine curve fitting methods

Usage

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

Arguments

par
A vector of parameters

y
Numeric vector, vegetation index time-series

t
Numeric vector, date variable

fun

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)
# simulate vegetation time-series
fFUN = doubleLog.Beck
par = c(
  mn = 0.1,
  mx = 0.7,
  ...
### Description

- **getBits**: Extract bitcoded QA information from bin value
- **qc_summary**: Initial weights based on Quality reliability of VI pixel, suit for MOD13A1, MOD13A2 and MOD13Q1 (SummaryQA band).
- **qc_51**: 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_NDVI3g**: For NDVI3g
- **qc_NDVI4**: For NDVIv4
- **qc_StateQA**: Initial weights based on StateQA, suit for MOD09A1, MYD09A1.

### Usage

```r
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_51(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
qc_NDVI3g(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
qc_NDVI4(QA, wmin = 0.2, wmid = 0.5, wmax = 1)
```
getRealDate

Arguments

- **x**: Binary value
- **start**: Bit starting position, count from zero
- **end**: Bit ending position
- **QA**: Quality control variable
- **wmin**: Double, minimum weight (i.e. weight of snow, ice and cloud).
- **wmid**: Double, middle weight, i.e. marginal,
- **wmax**: Double, maximum weight, i.e. good,

Value

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

Examples

```r
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_51 <- qc_51(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)
```

---

Description

Convert MODIS DayOfYear to the exact compositing date.

Usage

```r
getRealDate(date, DayOfYear)
```

Arguments

- **date**: Date vector, the first day of the 16-day composite period.
- **DayOfYear**: Numeric vector, exact composite day of year.

Value

- A data.table with a new column `t`, which is the exact compositing date.
get_fitting

Examples

library(phenofit)
data("MOD13A1")

df <- MOD13A1$dt
df$t <- getRealDate(df$date, df$DayOfYear)

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**

---

**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**


**See Also**

`curvefit()`

**Examples**

```r
library(phenofit)
# simulate vegetation time-series
fFUN = doubleLog.Beck
par = c(
  mn = 0.1,
  mx = 0.7,
  sos = 50,
```

```r
```
get_param

Description
Get parameters from curve fitting result

Usage
get_param(fits)

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)
methods <- c("AG", "Beck", "Elmore", "Gu", "Zhang") # Klos too slow
fFItS <- curvefit(y, t, tout, methods)
# multiple years
fits <- list('2001' = fFItSs, '2002' = fFItSs)

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), analytical = TRUE,
smoothed.spline = FALSE, IsPlot = FALSE, showName_fitting = TRUE,
...

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

Arguments
fits A list of fFItSs() 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.
fFItSs fFItSs 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

```r
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**

**Description**

Good of fitting

**Usage**

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

include.cv  If true, cv will be included.
include.r   If true, r and R2 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
gofHy_obsL y_simI
```

Description

This function is only suitable for 16-day EVI time-series.

Usage

```
init_lambda(y)
```

Arguments

- `y`  Numeric vector
Examples

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

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

sitename <- dt$site[1]
d <- dt[site == sitename, ] # get the first site data
lambda <- init_lambda(d$y)
```

---

init_param

---

Description

Initialize parameters of double logistic function

Usage

```r
init_param(y, t, w)
```

Arguments

- `y` input vegetation index time-series.
- `t` the corresponding doy(day of year) of `y`.
- `w` weights

Examples

```r
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)

l_param <- init_param(y, t)
```
I_optim  Interface of unified optimization functions.

Description

Caution that optimx speed is not so satisfied. So I_optim is present.

Usage

I_optim(prior, FUN, y, t, tout, method = "BFGS", ...)

I_optimx(prior, FUN, y, t, tout, method, 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.
FUN           Fine curve fitting function for goal function f_goal().
y            Numeric vector, vegetation index time-series
t            Numeric vector, Date variable
tout            Corresponding doy of prediction.
method            method can be one of 'BFGS', 'CG', 'Nelder-Mead', 'L-BFGS-B', 'nlm', 'nlminb', 'ucminf'. For I_optimx, other methods are also supported, e.g. 'spg', 'Rcgmin', 'Rvmmin', 'newuoa', 'bobyqa'...
verbose            If TRUE, all optimization methods in optimx::optimx() are used, and print optimization information of all methods.

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

stats::optim(), stats::nlminb(), stats::nlm(), optimx::optimx(), ucminf::ucminf()  

Examples

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

# initial parameter
par0 <- c(
  mn = 0.15,
  mx = 0.65,
  sos = 100,
  resp = 0.12,
  eos = 200,
  rau = 0.12)

objective <- f_goal # goal function
optFUNs <- c("opt_ucminf", "opt_nlminb", "opt_nlm", "opt_optim") %>% set_names(., .)
prior <- as.matrix(par0) %>% t() %>% rbind(., .)

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

# microbenchmark::microbenchmark(
#   I_optim(prior, fFUN, y, t, tout, c("BFGS", "ucminf", "nlm", "nlminb")),
#   I_optimx(prior, fFUN, y, t, tout, c("BFGS", "ucminf", "nlm", "nlminb")),
#   times = 2
# )
kurtosis

Description

Inherit from package e1071

Usage

kurtosis(x, na.rm = FALSE, type = 3)
skewness(x, na.rm = FALSE, type = 3)

Arguments

x a numeric vector containing the values whose skewness is to be computed.
na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
type an integer between 1 and 3 selecting one of the algorithms for computing skewness.

Examples

x = rnorm(100)
coef_kurtosis <- kurtosis(x)
coef_skewness <- skewness(x)

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.Ag Asymmetric Gaussian.


Usage

logistic(par, t)
doubleLog.Zhang(par, t)
doubleAg(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 functions 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

Description


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


Description

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

Usage

```r
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

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

Description

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

Usage

```r
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.Nbeck', 'FitDL.Nelu', 'FitDL.Ngu', 'FitDL.Nklos', 'FitDL.Zhang'.
- **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, passed to `I_optim()` or `I_optimx()`. `I_optim` supports 'BFGS', 'CG', 'Nelder-Mead', 'L-BFGS-B', 'nlm', 'nlminb', 'ucminf'. `I_optimx` supports 'spg', 'Rcgmin', 'Rvmmin', 'newuoa', 'bobyqa', 'nmkb', 'hjkbb'.
- **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()`.
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.

- \texttt{opt\_ncminf} General-Purpose Unconstrained Non-Linear Optimization, see \texttt{ucminf::ucminf()}.
- \texttt{opt\_nlminb} Optimization using PORT routines, see \texttt{stats::nlminb()}.
- \texttt{opt\_nlm} Non-Linear Minimization, \texttt{stats::nlm()}.
- \texttt{opt\_optim} General-purpose Optimization, see \texttt{stats::optim()}.

Usage

\begin{verbatim}
opt\_ucminf(par\_\theta, objective, ...)
opt\_nlminb(par\_\theta, objective, ...)
opt\_nlm(par\_\theta, objective, ...)
opt\_optim(par\_\theta, objective, method = "BFGS", ...)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{par\_\theta} Initial values for the parameters to be optimized over.
  \item \texttt{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.
  \item \texttt{...} other parameters passed to \texttt{objective}.
  \item \texttt{method} optimization method to be used in \texttt{p\_optim}. See \texttt{stats::optim()}.
\end{itemize}

Value

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

# initial parameter
par0 <- c(
  mn = 0.15,
  mx = 0.65,
  sos = 100,
  rsp = 0.12,
  eos = 200,
  rau = 0.12)

objective <- f_goal # goal function
optFUNs <- c("opt_ucminf", "opt_nlminb", "opt_nlm", "opt_optim") %>% set_names(. , .)

opts <- lapply(optFUNs, function(optFUN){
  optFUN <- get(optFUN)
  opt <- optFUN(par0, objective, y = y, t = t, fun = fFUN)
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 legend?

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)
FITT <- fFITs$FITT$AG
par(mfrow = c(2, 2))
PhenoTrs(FITT)
PhenoDeriv(FITT)
PhenoGu(FITT)
PhenoKl(FITT)

phenofit_loaddata update all INPUT data according to input file.

Description
update all INPUT data according to input file.

Usage
phenofit_loaddata(options, rv, ...)

Arguments

options should have children of file_site, and one of file_veg_rda or file_veg_text.
rv return values to reactiveValues object.
... ignored.
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: fF!T

- 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 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?

Description

The GUI allows you to interactively visualize curve fitting time series and phenological metrics.

Usage

phenofit_shiny()

Examples

```r
## Not run:
phenofit_shiny

## End(Not run)```
plot_input  

Plot INPUT returned by check_input

Description

Plot INPUT returned by check_input

Usage

plot_input(INPUT, wmin = 0.2, ...)

Arguments

INPUT  
A list object with the elements of t, y, w, Tn (optional) and ylu, returned by 
check_input().

wmin  
Double, minimum weight (i.e. weight of snow, ice and cloud).

...  
other parameter will be ignored.

Examples

library(phenofit)
data("MOD13A1")

dt <- tidy_MOD13.ggee(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

Description

plot_phenofit

Usage

plot_phenofit(d_fit, seasons, title = NULL, title.ylab = "Vegetation Index", font.size = 14, 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().
title String, title of figure.
title.ylab String, title of ylab.
font.size Font size of axis.text
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-N56' # 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 = wTS

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

Description

Plot growing season dividing result.

Usage

plot_season(input, brks, plotdat, ylu, IsPlot.OnlyBad = FALSE, show.legend = TRUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>A list object with the elements of t, y, w, Tn (optional) and ylu, returned by check_input().</td>
</tr>
<tr>
<td>brks</td>
<td>A list object returned by season or season_mov.</td>
</tr>
<tr>
<td>plotdat</td>
<td>(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.</td>
</tr>
<tr>
<td>ylu</td>
<td>[low, high] of time-series y (curve fitting values are constrained in the range of ylu.</td>
</tr>
<tr>
<td>IsPlot.OnlyBad</td>
<td>If true, only plot partial figures whose NSE &lt; 0.3.</td>
</tr>
<tr>
<td>show.legend</td>
<td>Whether to show legend?</td>
</tr>
</tbody>
</table>
**R2_sign**

*Critical value of determined correlation*

**Description**

Critical value of determined correlation

**Usage**

\[
\text{R2\_sign}(n, \text{NumberOfPredictor} = 2, \text{alpha} = 0.05)
\]

**Arguments**

- **n** length of observation.
- **NumberOfPredictor** Number of predictor, including constant.
- **alpha** significant level.

**Value**

F statistic and \(R^2\) at significant level.

**References**

Chen Yanguang (2012), Geographical Data analysis with MATLAB.

**Examples**

\[
\text{R2\_critical} \gets \text{R2\_sign}(30, \text{NumberOfPredictor} = 2, \text{alpha} = 0.05)
\]

**season**

*Growing season dividing*

**Description**

Divide growing seasons according to rough fitting (rFUN) result.

For **season**, rough fitting is applied for whole. For **season\_mov** rough fitting is applied in every year, during which \text{maxExtendMonth} is extended.
Usage

```r
season(INPUT, rFUN = wWHIT, wFUN = wTSM, iters = 2, wmin = 0.1,
lambda, nf = 3, frame = floor(INPUT$nptperyear/5) * 2 + 1,
minpeakdistance, r_max = 0.2, r_min = 0.05, ypeak_min = 0.1,
rtrough_max = 0.6, MaxPeaksPerYear = 2, MaxTroughsPerYear = 3,
calendarYear = FALSE, IsPlot = FALSE, plotdat = INPUT,
print = FALSE, adj.param = TRUE, ...)
```

```r
season_mov(INPUT, rFUN = wWHIT, wFUN = wTSM, iters = 2, wmin = 0.1,
IsOptim_lambda = FALSE, lambda = NULL, nf = 3,
frame = floor(INPUT$nptperyear/5) * 2 + 1, maxExtendMonth = 12,
calendarYear = FALSE, ..., IsPlot = TRUE, IsPlot.vc = FALSE,
IsPlot.OnlyBad = FALSE, plotdat = INPUT, print = TRUE,
titlestr = "")
```

```r
stat_season(INPUT, brks)
```

Arguments

- **INPUT** A list object with the elements of t, y, w, Tn (optional) and ylu, returned by `check_input()`.  
- **rFUN** Rough curve fitting function, can be one of `wSG()`, `wWHIT()` and `wHANTS()`.  
- **wFUN** weights updating function, can be one of `wTSM()`, `wChen()`, `wBisquare()` and `wSELF()`.  
- **iters** How many times curve fitting is implemented.  
- **wmin** Double, minimum weight (i.e. weight of snow, ice and cloud).  
- **lambda** The smoothing parameter of `wWHIT()`. For `season_mov()`, if lambda is NULL, `init_lambda()` will be used. Generally, it was set as 10000, 15, and 5 for daily, 8-day and 16-day inputs respectively.  
- **nf** The parameter of `wHANTS()`, number of frequencies to be considered above the zero frequency.  
- **frame** The parameter of `wSG()`, moving window size. Suggested by TIMESAT, default `frame = floor(nptperyear/7)+2 + 1`.  
- **minpeakdistance** Numeric, in the unit of points (default as `nptperyear/6`). The minimum distance of two peaks. If the distance of two maximum extreme value less than `minpeakdistance`, only the real maximum value will be left.  
- **r_max** Similar as `r_min`, The maximum threshold should be greater than `r_max`.  
- **r_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 `r_min`.  
- **ypeak_min** `ypeak >= ypeak_min`  
- **rtrough_max** `ytrough <= rtrough_max*A`, A is the amplitude of y.
MaxPeaksPerYear
This parameter is used to adjust lambda in iterations. If PeaksPerYear > MaxPeaksPerYear, then lambda = lambda*2.

MaxTroughsPerYear
This parameter is used to adjust lambda in iterations. If TroughsPerYear > MaxTroughsPerYear, then lambda = lambda*2.

calendarYear
If true, only one static calendar growing season will be returned.

IsPlot
Boolean

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.

print
Whether to print progress information

adj.param
Adjust rough curve fitting function parameters automatically, if too many or to less peak and trough values.

... For season_mov(). Other parameters passed to season(); For season(), other parameters passed to findpeaks().

IsOptim_lambda
Whether to optimize Whittaker's parameter lambda by V-curve theory?

maxExtendMonth
Previous and subsequent maxExtendMonth data were added for every year curve fitting.

IsPlot.vc
Whether to plot V-curve optimized time-series.

IsPlot.OnlyBad
If true, only plot partial figures whose NSE < 0.3.

titlestr
string for title

brks
A list object returned by season or season_mov.

Details
Before dividing growing season, INPUT should be added a year in head and tail first by add_HeadTail. Finally, use findpeaks() to get local maximum and local minimum values. If two local minimum define a growing season. If two local minimum(maximum) are too closed, then only the smaller(biger) is left.

Value
- whit: Rough fitting result.
- dt: Growing season dividing information.

List object, list(whit, dt)

See Also

findpeaks().
Examples

```r
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)

# all year as a whole
brks <- season(INPUT,
rFUN = wWHIT, wFUN = wFUN,
lambda = 10,
plotdat = d, IsPlot = IsPlot, print = FALSE, IsPlot.OnlyBad = FALSE)

# curve fitting by year
brks2 <- season_mov(INPUT,
rFUN = wWHIT, wFUN = wFUN,
lambda = 10,
plotdat = d, IsPlot = IsPlot, print = FALSE, IsPlot.OnlyBad = FALSE)
```

smooth_wSG  
*Weighted Savitzky-Golay*

**Description**

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

**Usage**

smooth_wSG(y, halfwin = 1L, d = 1L, w = NULL)

smooth_SG(y, halfwin = 1L, d = 1L)
**tidyFitPheno**

**Arguments**

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

**Examples**

```r
y <- 1:15
w <- seq_along(y)/length(y)
frame = 5
d = 2
s1 <- smooth_wSG(y, frame, d, w)
s2 <- smooth_SG(y, frame, d)
```

**Description**

Tidy for every method with multiple years phenology data

**Usage**

```r
tidyFitPheno(phenodata)
```

**Arguments**

- `phenodata`: Phenology metrics extracted from `get_pheno`

**Examples**

```r
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
```
tidy_MOD13.gee
fItS <- curvefit(y, t, tout, methods)

# multiple years
fits <- list('2001' = fFITs, '2002' = fFITs)
pheno <- get_pheno(fits, "AG", IsPlot=FALSE)

p <- tidyFitPheno(pheno)

### Description

Tidy MODIS 'MOD13' VI products (e.g. MOD13A1, MOD13A2, ...) raw data exported from Google Earth Engine. Tidy contents include:

1. add exact compositing date, see `getRealDate()`.

2. Init weights according SummaryQA, see `qc_summary()`.

### Usage

tidy_MOD13.gee(infile, outfile, wmin = 0.2)

### Arguments

- **infile**: A character csv file path or a data.table
- **outfile**: Output file name. If missing, will not be written to file.
- **wmin**: Double, minimum weight of bad points, which could be smaller the weight of snow, ice and cloud.

### Value

A tidied data.table, with columns of 'site', 'y', 't', 'w', 'date' and 'SummaryQA'.

- **site**: site name
- **y**: real value of EVI, [-1, 1]
- **date**: image date
- **t**: exact compositing date constructed from DayOfYear
- **w**: weights
- **SummaryQA**: A factor, QA types, one of "good", "margin", "snow/ice" or "cloud".

### Examples

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

**Description**

V-curve is used to optimize Whittaker parameter lambda. Update 20180605 add weights updating to whittaker lambda selecting

**Usage**

\[
v\_curve(INPUT, \text{lg\_lambdas}, d = 2, \text{IsPlot} = \text{FALSE}, \text{wFUN} = \text{wTSM}, \text{iters} = 2)\]

**Arguments**

- **INPUT** A list object with the elements of \( t, y, w, Tn \) (optional) and \( ylu \), returned by `check_input()`. 
- **lg\_lambdas** \( \lambda \) vectors of Whittaker parameter. 
- **d** Difference order. 
- **IsPlot** Boolean. Whether to plot figure? 
- **wFUN** weights updating function, can be one of \( \text{wTSM(), wChen(), wBisquare(), wSELF()} \). 
- **iters** How many times curve fitting is implemented.

**Examples**

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

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

sitename <- dt$site[1] 
sp <- st[Site == sitename,] # station point

D <- check_input(dt, st, d, wmin = 0.02)

# INPUT$y0 <- dnew$y  # raw time-series, for visualization

lg_lambdas <- seq(0, 3, 0.1)
r <- v_curve(INPUT, lg_lambdas, d = 2, IsPlot = TRUE)
```

**Description**

Weighted HANTS smoother

**Usage**

```r
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 weight (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)**

whit2

Examples

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

l <- check_input(d$t, d$y, d$w, nptperyear=23)
r_whants <- wHANTS(l$y, l$t, l$w, ylu = 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

```r
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

Examples

```r
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 weight 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.
- wBeck Beck et al., (2006) weight updating method. wBeck need sos and eos input. The function parameter is different from others. It is still not finished.

**Usage**

```r
wSELF(y, yfit, w, ...)
wTSM(y, yfit, w, iter = 2, nptperyear, wfact = 0.5, ...)
wBisquare(y, yfit, w, ..., wmin = 0.2)
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.
wSG

Description

Weighted Savitzky-Golay

nptperyear Integer, number of images per year.

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.

Value

wnew Numeric Vector, adjusted weights.

Author(s)

TSM 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


wSG Weighted Savitzky-Golay
Usage

\begin{verbatim}
wsG(y, w, nptperyear, ylu, wFUN = wTSM, iters = 2,
    frame = floor(nptperyear/7) * 2 + 1, d = 2, ...)
\end{verbatim}

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**: [low, high] 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


Examples

\begin{verbatim}
library(phenofit)
data("MOD13A2")
dt <- tidy_MOD13.gee(MOD13A2$dt)
d <- dt[site == "AT-Neu", ]

l <- check_input(d$dt, d$y, d$w, nptperyear=23)
r_wsG <- wsG(l$y, l$w, l$ylu, nptperyear = 23, iters = 2)
\end{verbatim}
**Description**

Weigthed Whittaker Smoother

**Usage**

\[ \text{wWHIT}(y, w, ylu, nptperyear, wFUN = \text{wTSM}, \text{iters} = 1, \lambda = 15, \text{second} = \text{FALSE}, \ldots) \]

**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 \( w_{\text{min}} \), the others will be 1.0.

- **ylu**
  
  \([\text{low}, \text{high}]\) 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**


Examples

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

l <- check_input(d$t, d$y, d$w, nptperyear=23)
r_whit <- wWHIT(l$y, l$w, l$sylu, nptperyear = 23, iters = 2)
```
Index

*Topic **datasets**
  MOD13A1, 30
  add_HeadTail, 3
  backval, 4
  check_input, 5
  check_input(), 38, 40, 42, 47
  check_ylu, 7
  curvature (D), 11
  curvefit, 8
  curvefit(), 13, 20–23
  curvefits, 9
  cv_coef, 10
  D, 11
  D1 (D), 11
  D2 (D), 11
  doubleAG (Logistic), 29
  doubleLog.Beck (Logistic), 29
  doubleLog.Elmore (Logistic), 29
  doubleLog.Gu (Logistic), 29
  doubleLog.Klos (Logistic), 29
  doubleLog.Zhang (Logistic), 29
  f_goal, 17
  f_goal(), 27
  fFIT, 13
  fFIT(), 33
  fFITs, 13
  fFITs(), 8, 13, 23
  findpeaks, 14
  findpeaks(), 43
  FitAG(FitDL), 15
  FitAG(), 8
  FitDL, 15
  FitDL.Beck(), 8
  FitDL.Elmore(), 8
  FitDL.Gu(), 8
  FitDL.Klos(), 8
  FitDL.Zhang(), 8
  get_fitting, 20
  get_fitting(), 39
  get_GOF, 21
  get_param, 22
  get_pheno, 23
  get_pheno(), 5
  getBits, 18
  getRealDate, 19
  getRealDate(), 46
  GOF, 24
  I_optim, 27
  I_optim(), 27, 32, 34
  I_optimx (I_optim), 27
  I_optimx(), 27, 32
  init_lambda, 25
  init_lambda(), 42
  init_param, 26
  kurtosis, 29
  Logistic, 29
  Logistic(), 17
  MOD13A1, 30
  movmean, 31
  opt_FUN, 33
  opt_nlm (opt_FUN), 33
  opt_nlmnb (opt_FUN), 33
  opt_optim (opt_FUN), 33
  opt_ucminf (opt_FUN), 33
  optim_pheno, 32
  optim_pheno(), 13, 16, 34, 35
  optimx::optimx(), 27, 28
  PhenoDeriv (PhenoExtractMeth), 35
  PhenoExtractMeth, 35
  phenofit::backval(), 4–6
phenofit_loaddata, 36
phenofit_plot, 37
phenofit_shiny, 37
PhenoGu (PhenoExtractMeth), 35
PhenoK1 (PhenoExtractMeth), 35
PhenoTrs (PhenoExtractMeth), 35
plot.fffits (fffITs), 13
plot_input, 38
plot_input(), 40, 43
plot_phenofit, 39
plot_phenofit(), 5
plot_season, 40

qc_5l (getBits), 18
qc_NDVI3g (getBits), 18
qc_NDVIv4 (getBits), 18
qc_StateQA (getBits), 18
qc_summary (getBits), 18
qc_summary(), 46

R2_sign, 41
season, 41
season(), 39, 43
season_mov (season), 41
season_mov(), 39, 42, 43
skewness (kurtosis), 29
smooth_SG (smooth_wSG), 44
smooth_wSG, 44
stat_season (season), 41
stats::nlm(), 28, 33
stats::nlminb(), 28, 33
stats::optim(), 28, 33

tidy_MOD13.gee, 46
tidyFitPheno, 45

ucminf::ucminf(), 28, 33

v_curve, 47
wBisquare (wSELF), 50
wBisquare(), 9, 42, 47
wChen (wSELF), 50
wChen(), 9, 42, 47
wHANTS, 48
wHANTS(), 42
whit2, 49
wKong (wSELF), 50
wSELF, 50