phenopix R package vignettes 1/3: base vignette

Gianluca Filippa
Environmental Protection Agency of Aosta Valley
103 PUBLICATIONS **2,792** CITATIONS

Phenopix

G. Filippa, E. Cremonese, M. Migliavacca, A. Richardson, M. Galvagno, M. Forkel

December 24, 2020

This vignette aims at illustrating the main features of the package `phenopix`. This package was designed for processing digital images of the vegetation cover in order to compute vegetation indexes that can be in turn used to track the seasonal development of the vegetation. The analysis can be run on one or more portions of the image (so called regions of interest, ROIs). Regions of interest can be of any polygonal shape. For data processing, two approaches are available: ROI-averaged analysis or pixel based analysis. ROI-averaged analysis is based on the computation of vegetation indexes as the average of the entire ROI, whereas pixel based analysis allows to treat separately each pixel of the image. Data used to show phenopix package are from imagery archive of Torgnon Grassland site, belonging to the PHENOCAM network. The rationale and the objectives that motivate the processing chain that will be described here are established in the scientific literature of the last 20 years. See References for a sample of the most relevant publications. Many functions of the package are a partial modification of the package `greenbrown` (infos: http://greenbrown.r-forge.r-project.org/), for which we thank Matthias Forkel.

1 System requirements

`phenopix` requires R (>= 2.15.3) and imports one or more functions from the following packages:

`zoo, plyr, SDMTools, jpeg, stringr (>= 1.0.0), bcp, strucchange, parallel, foreach, doParallel, iterators, gtools, raster`

This vignette was run on:

```
> library(phenopix)
> sessionInfo()
```

```
R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.1 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=it_IT.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=it_IT.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=it_IT.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=it_IT.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] phenopix_2.4.2

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.5         lattice_0.20-41   codetools_0.2-16  gtools_3.8.2
 [5] zoo_1.8-8          foreach_1.5.1     grid_3.6.3        plyr_1.8.6
 [9] magrittr_1.5       stringi_1.5.3     sp_1.4-4          raster_3.3-13
[13] doParallel_1.0.16 strucchange_1.5-2 sandwich_3.0-0    iterators_1.0.13
[17] tools_3.6.3        stringr_1.4.0     jpeg_0.1-8.1      bcp_4.0.3
[21] parallel_3.6.3     compiler_3.6.3
```

From the output of `sessionInfo` you will also notice the phenopix version I am using.

2 Topics covered

This vignette covers the preliminary and main steps of the processing chain (see section Steps for details). A specific vignette is available for the pixel-based spatial analysis.

3 Install the package

The package `phenopix` is on CRAN and can be installed via the following command:

```
> install.packages("phenopix")
```

Package vignettes are no longer available within the package. Instead find them on my research gate page:

https://www.researchgate.net/profile/Gianluca$_F$*ilippa/publications*

4 The steps

The first step is to give a well defined structure to a folder with the function `structureFolder()`.

The second step of the analysis is to choose a region of interest in an image. The functions useful for this step include:

- `DrawROI()` to draw a region of interest in your pictures
- `PrintROI()` to plot your ROI into an image
- `updateROI()` to apply ROI coordinates to an image of different size

Once the ROI is chosen, drawn and the underlying coordinates properly saved, color digital numbers are extracted and vegetation indexes (VIs) are calculated, using one main function `extractVIs()`.

Afterwards, raw VIs must be filtered out to get a reliable seasonal trajectory. This is the job of the function `autoFilter()`.

Then, several options are available to process the resulting data, ranging from fitting a curve to extracting break points on a seasonal trajectory, including several methods to extract relevant moments in the season (aka phenophases). Functions useful for this step include:

- `greenProcess()` to fit a curve to the data (ROI-averaged approach)
- `greenExplore()` to fit all curves and phenophases with no uncertainty estimation, this function is coupled with
- `plotExplore()`, which plots all fittings and phenophases in the object in output from `greenExplore()`
- `spatialGreen()` to fit a curve to the data (pixel-based approach)
- `PhenoBP()` to extract break points on a seasonal trajectory of data

A number of facilities are then built to plot, summarize, post process and render the results. These include:

- generic `plot()`, `print()`, `update()` and `summary()` functions with dedicated methods

- `plotSpatial()` to plot results from the pixel-based analysis

- `extractParameters()` to extract phenophases and curve parameters after the pixel-based analysis.

In the following paragraphs each step will be discussed and illustrated in detail.

5 Structuring a folder tree useful for the analysis

Giving a good structure to your analysis can make all subsequent steps simple and straightforward. If you are running a site that records images you will be dealing with quite heavy folders (with likely multiple years of data, hence some thousand files of images) that you need to handle with care. We suggest separate folders for each site (of course) but also year of analysis. Each year folder should contain a sub-folder with all images to be processed (`/IMG`), one folder containing the reference image, i.e. the image you will use to draw your ROI (`/REF`), one folder containing data for the region of interest (`/ROI`) and one folder containing extracted vegetation indexes (`/VI`). The function `structureFolder()` provides a facility to create appropriate sub-folders:

```
> library(phenopix)
> my.path <- structureFolder(path = getwd(), showWarnings = FALSE)
```

```
Put all your images in /home/gian/sweave/IMG/
Put your reference image in /home/gian/sweave/REF/
Draw your ROI with DrawROI():
 set path_img_ref to  /home/gian/sweave/REF/
 set path_ROIs to /home/gian/sweave/ROI/
Then you can extractVIs():
 set img.path as /home/gian/sweave/IMG/
 set roi.path as /home/gian/sweave/ROI/
 set vi.path to /home/gian/sweave/VI/
------------------------
Alternatively, assign this function to an object and use named elements of the returned li
```

```
> str(my.path)
```

```
List of 4
 $ img: chr "/home/gian/sweave/IMG/"
 $ ref: chr "/home/gian/sweave/REF/"
 $ roi: chr "/home/gian/sweave/ROI/"
 $ VI : chr "/home/gian/sweave/VI/"
```

`structureFolder()` creates sub-folder at a given path (in this example, the working directory) and stores all path in a named list. You can easily access all needed paths by simply pointing to the right object in your path object. Note that if one folder already exists the function does not overwrite existing folders, but gives a warning. Note that the suggested structure is absolutely not mandatory. It is just a suggestion that can make easier the next steps. Once the folder structure is done, you have to:

- manually put your series of images to be processed into the `/IMG` folder
- manually put one of such images in the `/REF` folder, this is the image that will be printed on screen to draw your ROI.

6 Drawing a region of interest (ROI)

Apart from structuring folders, drawing a ROI is the first, hence most important step of the analysis.

The procedure is based on two steps: first, a reference image (chosen by the user) is read as a raster brick using the `raster::brick()` function, which is then plotted by a call to `plotRGB()`. In Fig. 1 is the reference image from one of our sites, Torgnon (NW Italy, 2100 m of elevation) and the code used to plot the image. We first define an easy plotting function to print on screen images.

```
> library(raster)
> img <- brick('REF/20130630T1000.jpg')
> plotRGB(img)
```

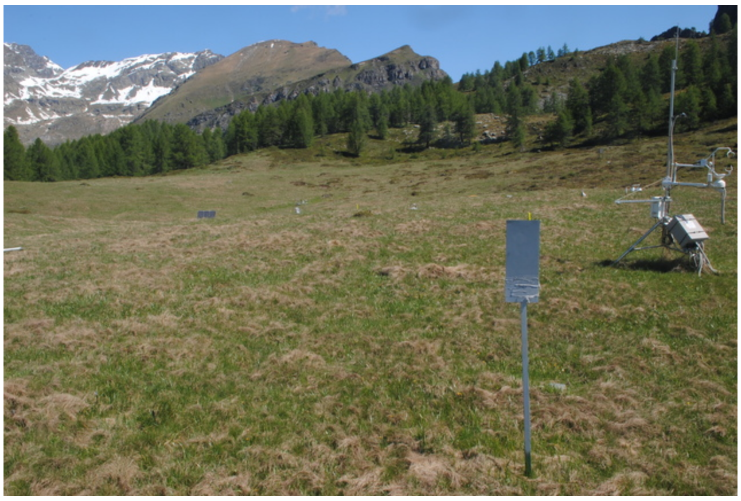

Figure 1: A jpeg image printed on a graphic device using `raster::brick()` and `raster::plotRGB()` functions

This chunk of code is automatically included in the `DrawMULTIROI()` function. The usage is:

```
> str(DrawMULTIROI)
```

```
function (path_img_ref, path_ROIs, nroi = 1, roi.names = NULL, file.type = ".jpg")
```

where `path_img_ref` is the folder of your reference image, path_ROIs is the path in your computer where to store the `RData` with ROI features, number of ROIs and their names. A call to the function opens a graphic device and allows the use of `locator()` to define your ROI(s). Note that the use of `locator` is somewhat system specific. Check out the help file `?locator` for more details. Locator allows to draw a polygon by left-clicking vertices and then right-clicking (or press ESC on MacOS) to close the polygon. Each ROI can be constituted by multiple polygons. When you close a polygon (right-click) you will be asked if you are done with current ROI If you have chosen more than one ROI, after closing your first polygon, the image will appear again unmodified to draw the second ROI, and so on. Note that the plot title recalls you which of your ROIs you are actually drawing. When you are done, in your `path_ROIs` an RData called `roi.data.RData` will be stored. This is actually a list with the following structure:

```
> load('ROI/roi.data.Rdata')
> names(roi.data)

[1] "fg" "bg"

> names(roi.data[[1]])

[1] "mask"     "polygons"

> class(roi.data[[1]]$mask)

[1] "RasterLayer"
attr(,"package")
[1] "raster"

> class(roi.data[[1]]$polygons)

[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

A two elements list (one for each ROI) with ROI names. Each element is again a list containing two elements. One is a binary raster mask, i.e. a raster layer with the same extent and resolution as your reference image; the second is a collection of spatialPolygons with coordinates of points as selected by `locator()`.

Additionally, in `path_ROIs` separate jpeg files for each of your regions of interest are stored. A call to the function `printROI()` allows to plot in the same graph all existing ROIs for a picture. In the example from Torgnon, two ROIs were drawn, one corresponding to the foreground of the image and one to the background (`fg` and `bg` respectively. Here is the code to generate the plot in fig. 2:

```
> PrintROI(path_img_ref = 'REF/20130630T1000.jpg',
+          path_ROIs = 'ROI/',
+          which = 'all',
+          col = palette())
```

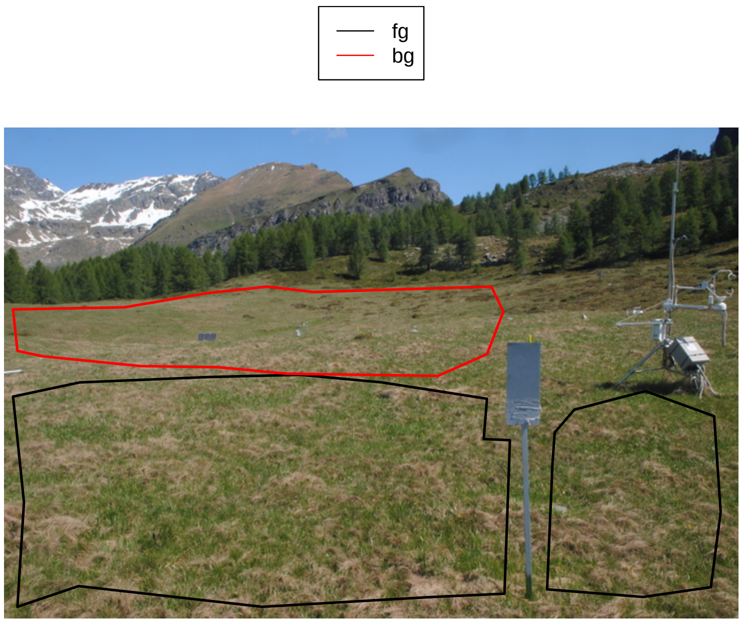

Figure 2: A plot of your regions of interest (ROIs), in output from `PrintROI()`. Note that ROI named fg (foreround) is actually composed by two separated polygons.

When you draw a ROI on your best quality image (in this case 640 x 428 pixels, as the REF image for Torgnon) you will probably need to identify the same ROI in smaller size images. This will be the case, for example, if you want to conduct a pixel-based analysis, illustrated later on. Pixel based analysis is computationally intense and therefore it is suggested to run it on rather small size images. The function `updateROI()` allows to recalculate pixels falling within a given ROI in images of different size compared to the one where the ROI was first drawn. Usage is:

```
> str(updateROI)

function (old.roi, new.img)
```

`old.roi` is the original `roi.data` object, `new.img` is the re-sized image. A new object with same structure as the original roi.data is returned.

7 Extraction of vegetation indexes

At this point, you have an R object stored as `roi.data.Rdata` in your ROI path that defines which pixels fall into one or more ROIs. The next step will be to extract information on those pixels from each of your images. The function that performs this task is `extractVIs()` and the usage is as follows:

```
> str(extractVIs)

function (img.path, roi.path, vi.path = NULL, roi.name = NULL, plot = TRUE,
    begin = NULL, spatial = FALSE, date.code, npixels = 1, file.type = ".jpg",
    bind = FALSE, shift.matrix = NULL, ncores = "all", log.file = NULL)
```

`img.path` is the path where a stack of one year of images are stored. It is not mandatory to have only one year of images in your folder. However it is suggested to structure your data into separate folders for each year because nearly all the functions we will see later are designed to work an a single season of data. `roi.path` is the path to your `roi.data.Rdata`, `vi.path` is the path where extracted vegetation indexes will be saved. Hence, this function can be assigned to an object to have your vegetation indexes returned as R object, or alternatively `load`ed later if not assigned. The argument `begin` allows to set a beginning date to update an existing time series without reprocessing the whole year of data. For example, if you run `extractVIs` in mid June to have a first look at your time series, once your season will be completed you do not want to re-run the analysis on the already processed images. Hence, you set the argument `begin` to the first unprocessed date. A new `VI.data.Rdata` will be saved in your path, with the beginning date incorporated in the filename if argument `bind` is set to FALSE. Conversely, the VI.data object already existing in your VI folder will be updated with new records and overwritten.

The argument `npixels` defines if a pixel aggregation is performed prior to the analysis (i.e. image degradation). Default 1 means no aggregation. If npixels == 2 than 4 pixels are aggregated in a 2x2 square. Similarly if npixels is 3, 9 pixels are aggregated in 3x3 squares and so on. The argument `file.type` is used to specify how the extension of your jpeg files are written (e.g. jpg, jpeg, JPG, JPEG). More than one argument is also allowed to account for different extensions in the same folder. However, remember that only jpeg files are allowed.

The argument spatial allows to perform pixel-based analysis. This is a topic discussed in a dedicated vignette.

The argument shift.matrix can be optionally used to shift the region of interest according to a two-dimentional matrix of shifts, with one row for each image in the stack. These shifts, in pixel units, will be used to adjust ROI masks to accommodate for field-of-view shifts. This subject will be soon implemented in the package. The argument ncores is used to specify how many cores will be used for parallel computation. The argument log.file can be used to store a log of the progress of processing.

The construction of the time series implies that R recognizes a time vector, typically retrieved from the file name of each picture. The function responsible for this conversion is extractDateFilename(). It is a rather internal function but it is worth to look how it works to properly set the filenames of your imagery archive. Arguments to the function are filename and date.code. Filename must be a character string with an underscore '_' that separates site name and date (e.g. 'torgnon_20140728.jpg'). The format of your date must be provided in date.code. In the example above, date.code will be: 'yyyymmdd'. Let's look at some examples, but before doing so, it is worth to remember that the file naming system is under your responsibility when you set up the storage process for your images, or by some renaming routines set up later.

```
> filename <- 'torgnon_20140728.jpg' ## correct, with no hour
> ## if hour is missing it is defaulted to 12 pm
> extractDateFilename(filename, date.code='yyyymmdd')

[1] "2014-07-28 12:00:00 CEST"

> filename <- 'torgnon_201407281100.jpg' ## correct, with hour
> ## hours and minutes to upper letters, in R POSIX style
> extractDateFilename(filename, date.code='yyyymmddHHMM')

[1] "2014-07-28 11:00:00 CEST"

> filename <- 'torgnon_1407281100.jpg' ## correct, with 2 numbers for the year
> extractDateFilename(filename, date.code='yymmddHHMM')

[1] "2014-07-28 11:00:00 CEST"

> ## any separator for date elements is allowed
> ## including underscore
> filename <- 'torgnon_2014.07_28-11.00.jpg'
> extractDateFilename(filename, date.code='yyyy.mm_dd-HH.MM')
```

```
[1] "2014-07-28 11:00:00 CEST"
```

```
> ## Since phenopix version 2.0.2 underscores are also allowed before the date
> filename <- 'torgnon_grassland_2014.07_28-11.00.jpg'
> extractDateFilename(filename, date.code='yyyy.mm_dd-HH.MM')
```

```
[1] "2014-07-28 11:00:00 CEST"
```

Now that arguments of extractVIs are enumerated, we can run few different examples of this extraction

Now let's look from closer at the structure of the object `VI.data` saved in your `/VI` directory.

```
> ## a basic version of extractVIs
> extractVIs(img.path = 'IMG/', roi.path = 'ROI/', vi.path = 'VI/',
+ date.code='yyyy_mm_dd_HHMM', log='/home/gian/', ncores=5)
```

The code above will extract for each image RGB values for the pixels belonging to the ROI(s), and compute some statistics. This will be done for the whole set of images in the 'IMG/' folder. Results will be saved (but can also be assigned, if needed) in the 'VI/' folder. Additionally a dignostic plot will be also saved in png format in the same 'VI/' folder. If the `date.code` is not properly specified the function will return an error before any other computation. A file of the processing progress will be saved in my home and I decided to use 5 processors for this computation. In the following example, I decide to process only the images collected after 2013-07-30, and append the results to an already existing `VI.data.RData` object.

```
> ## extractVIs applied to a subset of images, with
> ## results appended to an already existing VI.data object
> extractVIs(img.path = '/IMG/', roi.path = 'ROI/', vi.path = 'VI/',
+ date.code='yyyy_mm_dd_HHMM', log='/home/gian/', ncores=5, bind=TRUE,
+ begin='2018-07-30')
```

`extractVIs()` can be quite slow depending on the dimension of the images, apart from computer characteristics. Let's suppose I am not convinced of the position of my regions of interest and want to run a "quick and dirty" analysis. The argument `npixels` can be set to 2 or 4 to aggregate images and speed up computation in spite of a lower resolution. The aggregation is performed via `raster::aggregate()` function.

```
> load('VI/VI.data.Rdata')
> summary(VI.data) ## a list with two data.frames, one for each ROI

   Length Class      Mode
fg 18     data.frame list
bg 18     data.frame list

> names(VI.data[[1]]) ## check which vegetation indexes are extracted

 [1] "date"   "doy"    "r.av"   "g.av"   "b.av"   "r.sd"   "g.sd"   "b.sd"
 [9] "bri.av" "bri.sd" "gi.av"  "gi.sd"  "gei.av" "gei.sd" "ri.av"  "ri.sd"
[17] "bi.av"  "bi.sd"
```

The processing of each ROI produces a data.frame object with date in POSIXct format, numeric day of year (doy), and the vegetation indexes. Green, red and blue digital numbers (range [0,255]) averaged over the ROI (g.av, r.av and b.av, respectively), their standard deviations (g.sd, r.sd and b.sd). bri.av is the ROI averaged brightness, calculated as the sum of red green and blue digital numbers for each pixel and then averaged. From the digital numbers (dn) of each color, relative indexes (rel.i) are calculated as follows:

rel.i = dn color / (dn red + dn green + dn blue)

These values are calculated for each pixel and then averaged over the entire ROI (columns gi.av, ri.av, bi.av), and the standard deviation is calculated as well. In fig.3 you can see how a seasonal course of raw color digital numbers of a subalpine grassland site looks like:

```
> with(VI.data$fg, plot(date, r.av, pch=20, col='red',
+         ylim=c(0,255), ylab='DN [0,255]'))
> with(VI.data$fg, points(date, g.av, col='green', pch=20))
> with(VI.data$fg, points(date, b.av, col='blue', pch=20))
```

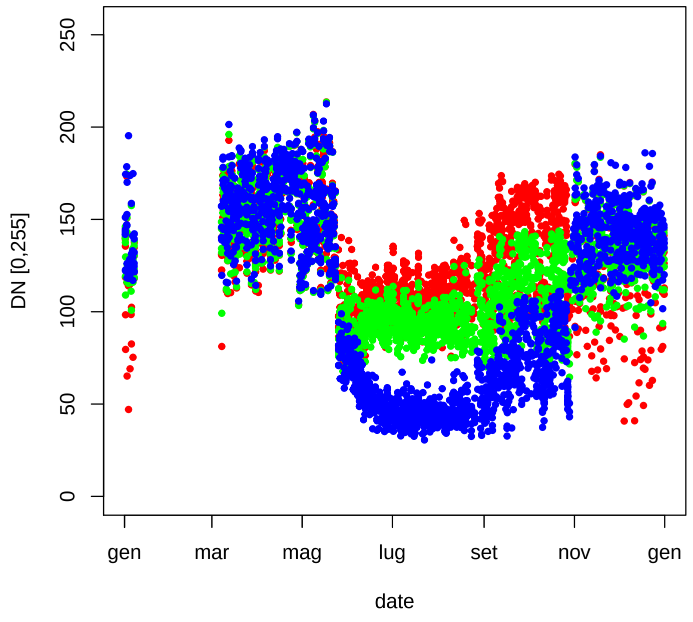

Figure 3: Seasonal course of raw digital numbers, Torgnon, year 2012

More interesting is the plot of relative indexes (fig. 4):

```
> with(VI.data$fg, plot(date, ri.av, pch=20, col='red',
+          ylim=c(0.1,0.6), ylab='Relative indexes'))
> with(VI.data$fg, points(date, gi.av, col='green', pch=20))
> with(VI.data$fg, points(date, bi.av, col='blue', pch=20))
```

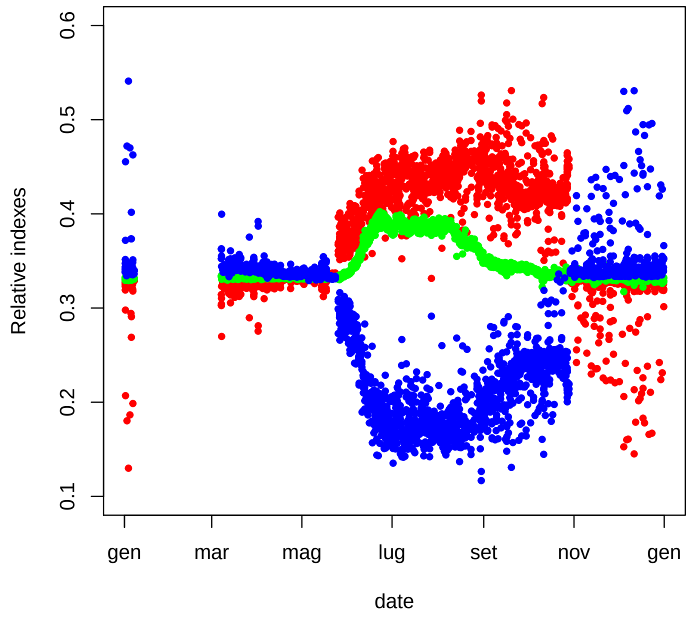

Figure 4: Seasonal course of relative green red and blue indexes, Torgnon grassland, year 2012

Several patterns are interesting in the seasonal course of fig.4:

- Snow disappearance (mid May) leads to an increase in relative red and a sharp decrease in relative blue

- The green signal follows a bell shaped pattern throughout the growing season, with a maximum in late July. This signal is somewhat mirrored by an inverse behavior of relative blue, whereas relative red gradually increases throughout the season.

The argument `spatial` can be set to true in order to perform a spatially explicit, pixel-based analysis. This will be covered in a dedicated vignette.

8 Filter out data

Data retrieved from images often need robust methods for polishing the time series. Bad weather conditions, low illumination, dirty lenses are among the most common issues that determine noise in the time series of vegetation indexes. Accordingly we designed a function `autoFilter()` based on 4 different approaches, see the examples in `?autoFilter` for details in the filtering procedure. The function is designed to receive in input a data.frame structured as in output from `extractVIs`, hence its default expression may appear rather complicate:

```
> str(autoFilter)
```

```
function (data, dn = c("ri.av", "gi.av", "bi.av"), raw.dn = FALSE, brt = "bri.av",
    na.fill = TRUE, filter = c("night", "spline", "max"), filter.options = NULL,
    plot = TRUE, ...)
```

But when applied to the `VI.data` object generated before it is quite straightforward as you see in the code below. Note also that `autoFilter()` returns by default a diagnostic plot shown in fig.5:

```
> filtered.data <- autoFilter(VI.data$fg)
> str(filtered.data)

'zoo' series from 2018-01-01 to 2018-12-31
  Data: num [1:297, 1:7] 0.324 0.329 0.329 0.328 0.325 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:7] "rcc" "gcc" "bcc" "brt" ...
  Index:  POSIXct[1:297], format: "2018-01-01" "2018-01-02" "2018-01-03" "2018-01-04" "201

> names(filtered.data)

[1] "rcc"              "gcc"              "bcc"              "brt"
[5] "night.filtered"   "spline.filtered"  "max.filtered"
```

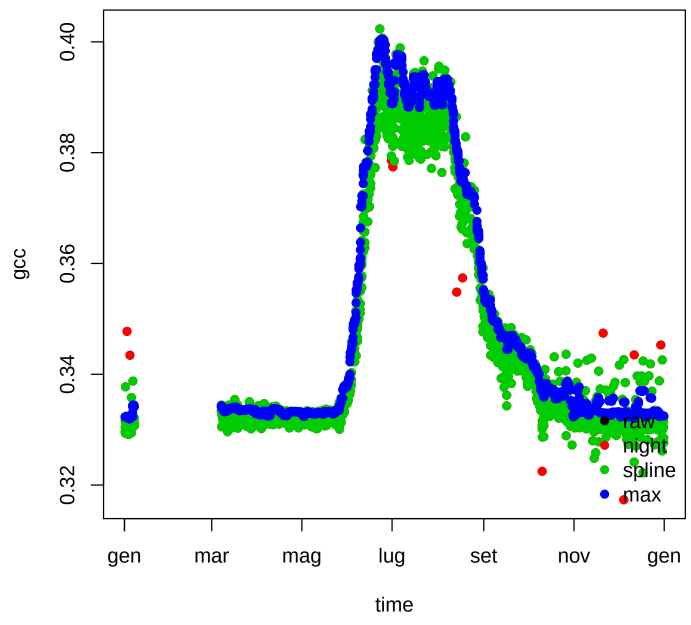

Figure 5: Raw and filtered relative greenness index, default plot of function `autoFilter()`

In the structure of the output data.frame there are three important points:

- We introduce here a new class of R objects (`zoo`). From here on all further analyses are based on `zoo` (or, to a lesser extent `ts`) time series. The time index of the data is numeric day of year (doy). As a consequence, the attribute year is lost at this step of the analysis (i.e. we suggest to include it in the object name);

- The function `autoFilter` aggregates data to a daily time step by default. The returned data.frame contains unfiltered (but still daily aggregated) color indexes (here called gcc, rcc and bcc, cc standing for chromatic coordinate) and a column of data for each filtering step. The name of the filter applied is reported in the column name.

- The argument `na.fill` defaults to TRUE, meaning that NA already existing in the VI.data (unlikely) or data discarded by the filtering procedure (much more likely) are filled by linear approximation (using `na.approx` from `zoo` package. This is done because the subsequent fitting step requires no NA appearing in the time series. If a user wants to have control on the discarded data and e.g. customize the gap-filling we recommend setting `na.fill` to FALSE.

For those unfamiliar with the `zoo` structure we created a function `convert` to convert from zoo to a normal data.frame

```
> dataframed <- convert(filtered.data, year='2012')
> str(dataframed)
```

```
'data.frame':        297 obs. of  9 variables:
 $ rcc            : num  0.324 0.329 0.329 0.328 0.325 ...
 $ gcc            : num  0.331 0.332 0.331 0.331 0.332 ...
 $ bcc            : num  0.345 0.339 0.34 0.341 0.343 ...
 $ brt            : num  419 432 355 356 362 ...
 $ night.filtered : num  0.331 0.332 0.331 0.331 0.332 ...
 $ spline.filtered: num  0.331 0.332 0.331 0.331 0.332 ...
 $ max.filtered   : num  0.332 0.332 0.332 0.332 0.332 ...
 $ doy            : POSIXct, format: "2018-01-01" "2018-01-02" ...
 $ time           : POSIXct, format: "2012-07-19" "2012-07-19" ...
```

However, we strongly recommend to get familiar with the `zoo` package since it has wonderful facilities for plotting, aggregating and filling time series.

Filters are based on methods relying on different parameters that can be tuned by the user (called filter options). A function allows to return default filter options that can be in turn changed.

```
> my.options <- get.options()
> names(my.options) # a named list, one element for each filter

[1] "night.filter"  "blue.filter"   "mad.filter"    "max.filter"
[5] "spline.filter"

> ## see help file for th meaning
> my.options$max.filter$qt <- 0.95 ## use 95th percentile instead
> ## of 90th for max.filter
> filtered.data2 <- autoFilter(VI.data$fg, filter.options=my.options, plot=FALSE)
> plot(filtered.data$max.filtered) ## default options
> lines(filtered.data2$max.filtered, col='red') ## customized options
> legend('topleft', col=palette()[1:2], lty=1, legend=c('90th', '95th'), bty='n')
```

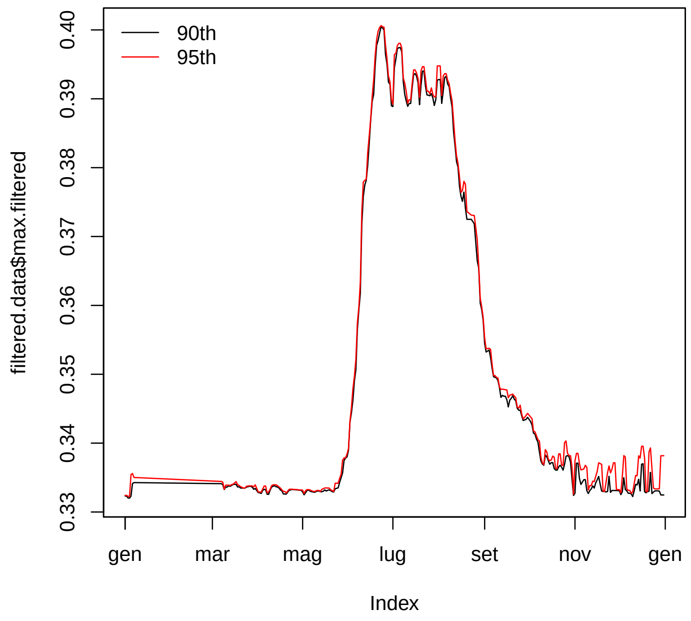

Figure 6: Effect (not that large indeed) of changing filter options with function `autoFilter()`

9 Fit a curve to the data

The seasonal trajectory of greenness index of a vegetation canopy provides per se important information, but to turn qualitative information into quantitative data we need to make some more computation. Traditionally, data similar to these (e.g. satellite-based NDVI trajectories) are processed in two main ways:

- extract time thresholds based on a percentage of development (e.g. the day when half of the maximum value of the index is reached);

- fit a curve and extract relevant thresholds based on curve properties.

In the package `phenopix` both possibilities are available. The core function for data fitting and phenophase extraction is `greenProcess()`. This function calls and is related to several rather internal functions that perform the different fittings. Available fittings include:

- the fit of a cubic spline

- the fit of an equation proposed by Beck et al. (2006)

- the fit of an equation proposed by Elmore et al. (2012)

- the fit of an equation proposed by Klosterman et al. (2014) with two implementations

- the fit of an equation proposed by Gu et al. (2009)

All fits are based on a double - logistic function with a different number of parameters.

After curve fitting, relevant dates in the seasonal trajectory (aka phenophases) are extracted with different methods:

- A method called `trs` which splits the seasonal course into increasing and decreasing trajectory based on the sign of the first derivative and then identifies a given threshold (by default the 50%) of both the increasing and decreasing trajectory. It allows to determine start of season (sos), end of season (eos) and length of season (los) as the difference between the two.

- A method called `derivatives` which extends `trs` in that it also calculates maximum growing and decreasing rates

- A method based on Klosterman approach which individuates 4 moments in the seasonal trajectory. Greenup represents the beginning of growth, maturity represents the reaching of some summer plateau, senescence represents the beginning of green decrease (or yellowing increase) and dormancy represents the end of the growing season.

- A method based on Gu approach which individuates 4 moments and some other curve parameters. The 4 relevant moments do not differ in their meaning compared to Klosterman phases, and are called upturn date (UD), stabilization date (SD), downturn date (DD) and recession date (RD).

Detail on curve fitting and phenophase extraction is provided in the help function of `?greenProcess` as well as in the help files of other more internal functions such as `?KlostermanFit`, `?GuFit`, `?PhenoExtract`. In fig.6 we show

4 different fitting methods applied to the same data (Torgnon grassland). But let's first have a look at the arguments of `greenProcess`:

```
> str(greenProcess)

function (ts, fit, threshold = NULL, plot = TRUE, which = "light", uncert = FALSE,
    nrep = 100, envelope = "quantiles", quantiles = c(0.1, 0.9), hydro = FALSE,
    sf = quantile(ts, na.rm = TRUE, prob = c(0.05, 0.95)), ncores = "all",
    ...)
```

`ts` is the `zoo` time series in input. It must be a time series with no NA. Arguments `fit` and `threshold` allows to choose the fitting and phenopahse methods, respectively. `plot` is a logical determining if a plot is returnoed or not, `which` is pertinent only if fit = 'klosterman', `uncert` is a logical for uncertainty computation, for which number of replicates is controlled by `nrep`. `envelope` and `quantiles` will be detailed later. `hydro` is a logical indicating wheter days must be converted to hydrodays before the analysis, where october 1t will be doy 1 and so on (designed for southern emisphere or for winter-growing plants). Since phenopix version > 2.0 the uncertainty estimation benefits from parallelization, for which arguments `ncores` controls the number of cores used in parallel computation, default is 'all' and the actual number of cores you want to use can be set with an integer. Parallelization is performed by calling function `foreach` in the `foreach` package.

```
> ## spline curve + trs phenophases
> fit1 <- greenProcess(filtered.data$max.filtered,
+         'spline',
+         'trs',
+         plot=FALSE
+         )
> summary(fit1)

Data
     Index          observed
 Min.   :  1.0   Min.   :0.3320
 1st Qu.:135.0   1st Qu.:0.3331
 Median :210.0   Median :0.3365
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3665
 Max.   :365.0   Max.   :0.4004
```

```
Predicted
     Index         predicted     
 Min.   :  1.0   Min.   :0.3315  
 1st Qu.:135.0   1st Qu.:0.3334  
 Median :210.0   Median :0.3361  
 Mean   :210.4   Mean   :0.3502  
 3rd Qu.:289.0   3rd Qu.:0.3666  
 Max.   :365.0   Max.   :0.3956  

Formula
NULL

Thresholds
        sos         eos         los         pop         mgs         rsp 
160.0000000 241.0000000  81.0000000 184.0000000   0.3865253          NA 
        rau        peak         msp         mau 
         NA   0.3956133   0.3650437   0.3614309 
```

```
> ## check the plot
> plot(fit1, type='p', pch=20, col='grey')
> ## Beck fitting + derivatives
> fit2 <- greenProcess(filtered.data$max.filtered,
+         'beck',
+         'derivatives',
+         plot=FALSE)
> summary(fit2)
```

```
Data
     Index          observed     
 Min.   :  1.0   Min.   :0.3320  
 1st Qu.:135.0   1st Qu.:0.3331  
 Median :210.0   Median :0.3365  
 Mean   :210.4   Mean   :0.3502  
 3rd Qu.:289.0   3rd Qu.:0.3665  
 Max.   :365.0   Max.   :0.4004  

Predicted
     Index         predicted     
 Min.   :  1.0   Min.   :0.3342  
 1st Qu.:135.0   1st Qu.:0.3342  
 Median :210.0   Median :0.3356
 Mean   :210.4   Mean   :0.3517
 3rd Qu.:289.0   3rd Qu.:0.3697
 Max.   :365.0   Max.   :0.3987

Formula
expression(mn + (mx - mn) * (1/(1 + exp(-rsp * (t - sos))) +
    1/(1 + exp(rau * (t - eos)))))

Thresholds
          sos           eos           los           pop           mgs
161.000000000 235.000000000  74.000000000 180.000000000   0.391555995
          rsp           rau          peak           msp           mau
  0.004508726  -0.003031381   0.398749314   0.368700727   0.373697346
```

```
> plot(fit2, type='p', pch=20, col='grey')
> ## klosterman fitting + klosterman phenophases
> fit3 <- greenProcess(filtered.data$max.filtered,
+         'klosterman',
+         'klosterman',
+         plot=FALSE)
> summary(fit3)
```

```
Data
     Index          observed
 Min.   :  1.0   Min.   :0.3320
 1st Qu.:135.0   1st Qu.:0.3331
 Median :210.0   Median :0.3365
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3665
 Max.   :365.0   Max.   :0.4004

Predicted
     Index         predicted
 Min.   :  1.0   Min.   :0.3331
 1st Qu.:135.0   1st Qu.:0.3335
 Median :210.0   Median :0.3350
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3655
 Max.   :365.0   Max.   :0.3968
```

```
Formula
expression((a1 * t + b1) + (a2 * t^2 + b2 * t + c) * (1/(1 +
    q1 * exp(-B1 * (t - m1)))^v1 - 1/(1 + q2 * exp(-B2 * (t -
    m2)))^v2))

Thresholds
   Greenup   Maturity Senescence   Dormancy
       147        173        212        268
```

```
> ## plot(fit3, type='p', pch=20, col='grey')
>
> ## gu fitting and phenophases
> fit4 <- greenProcess(filtered.data$max.filtered,
+         'gu',
+         'gu',
+         plot=FALSE)
> summary(fit4)
```

```
Data
     Index          observed     
 Min.   :  1.0   Min.   :0.3320  
 1st Qu.:135.0   1st Qu.:0.3331  
 Median :210.0   Median :0.3365  
 Mean   :210.4   Mean   :0.3502  
 3rd Qu.:289.0   3rd Qu.:0.3665  
 Max.   :365.0   Max.   :0.4004  

Predicted
     Index         predicted     
 Min.   :  1.0   Min.   :0.3331  
 1st Qu.:135.0   1st Qu.:0.3333  
 Median :210.0   Median :0.3353  
 Mean   :210.4   Mean   :0.3502  
 3rd Qu.:289.0   3rd Qu.:0.3649  
 Max.   :365.0   Max.   :0.3940  

Formula
expression(y0 + (a1/(1 + exp(-(t - t01)/b1))^c1) - (a2/(1 + exp(-(t -
    t02)/b2))^c2))
```

```
Thresholds
           UD            SD            DD            RD       maxline
 1.533105e+02  1.665884e+02  2.163170e+02  2.645647e+02  9.986943e-01
     baseline           prr           psr plateau.slope
 8.491011e-03  7.457540e-02 -2.018062e-02 -1.163522e-04
```

```
> plot(fit4, type='p', pch=20, col='grey')
> ## show all together
> library(zoo)
> t <- as.numeric(format(index(filtered.data$max.filtered), '%j'))
> par(lwd=3)
> plot(t, dataframed$max.filtered, type='p', pch=20,
+         ylab='Green chromatic coordinate', xlab='DOYs')
> lines(fitted(fit1), col='blue')
> lines(fitted(fit2), col='red')
> lines(fitted(fit3), col='green')
> lines(fitted(fit4), col='violet')
> legend('topleft', col=c('blue', 'red', 'green', 'violet'),
+         lty=1, legend=c('Spline', 'Beck', 'Klosterman', 'Gu'),
+         bty='n')
```

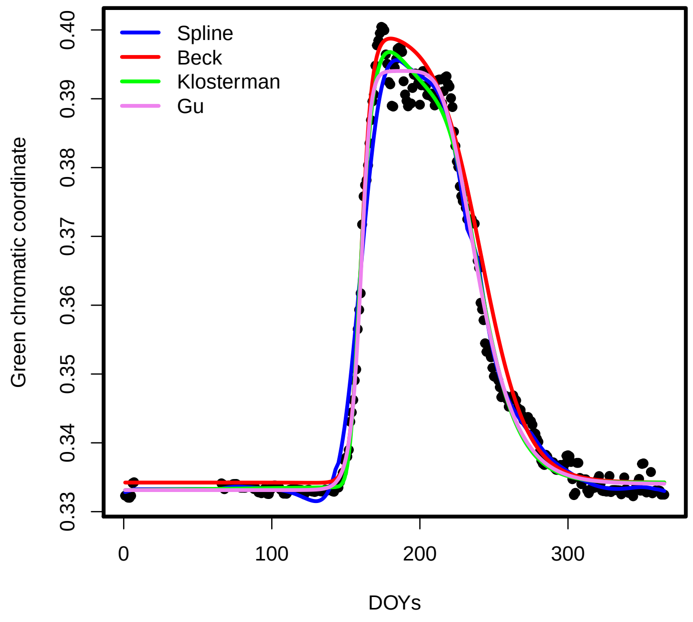

Figure 7: Comparison of 4 different fittings from `phenopix` package

The function `greenProcess` creates an object of class `phenopix` with dedicated methods. The `summary` function displays a summary of the input data and of the predicted points. It then reports the formula of the fitting equation, if pertinent, see e.g. summary of `fit1` which is not based on an equation.

Phenophases are printed as well. Note also the `fitted` function applied to `phenopix` object that returns a `zoo` time series of fitted values that can be directly `line`d to the plot.

To complete the overview on display generic methods applied to the objects of class `phenopix` here is the application of generic `plot` (fig.8) and `print` functions:

```
> plot(fit4, pch=20, col='grey', type='p',
+          xlab='DOYs', ylab='Green chromatic coordinates')
```

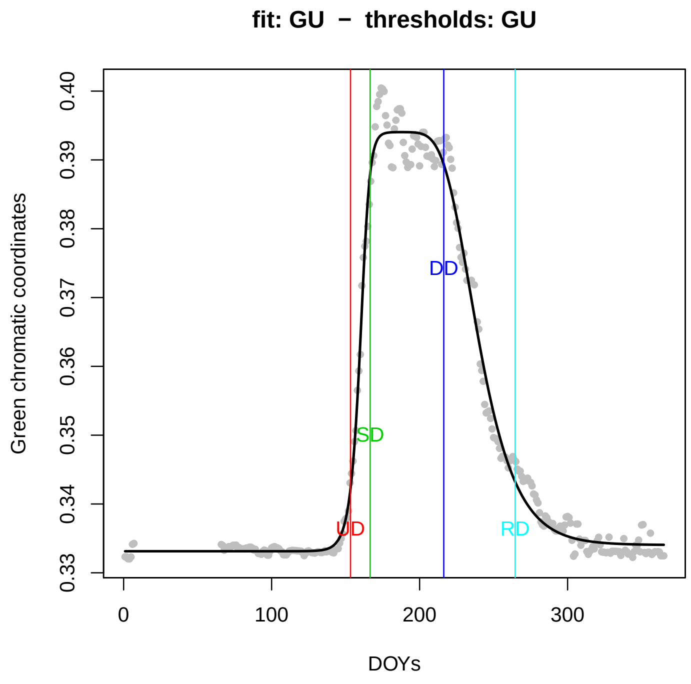

Figure 8: Generic `plot` function applied to `phenopix` objects

```
> print(fit4)

 #### phenopix time series processing ####

FITTING: GU

PREDICTED VALUES:
     Index          predicted
 Min.   :  1.0   Min.   :0.3331
 1st Qu.:135.0   1st Qu.:0.3333
 Median :210.0   Median :0.3353
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3649
 Max.   :365.0   Max.   :0.3940

FITTING EQUATION:
expression(y0 + (a1/(1 + exp(-(t - t01)/b1))^c1) - (a2/(1 + exp(-(t -
    t02)/b2))^c2))

FITTING PARAMETERS:
          y0           a1           a2          t01          t02           b1
8.491011e-03 9.902810e-01 9.755980e-01 1.623428e+02 1.870489e+02 2.684337e+00
          b2           c1           c2
1.718645e+01 5.607741e-01 1.522668e+01

THRESHOLDS: GU
           UD             SD             DD             RD        maxline
 1.533105e+02  1.665884e+02  2.163170e+02  2.645647e+02  9.986943e-01
     baseline            prr            psr plateau.slope
 8.491011e-03  7.457540e-02 -2.018062e-02 -1.163522e-04

UNCERTAINTY: FALSE
 N of replications = 0

HYDROLOGICAL DAY OF YEAR: FALSE
```

The `print` function returns information similar to `summary` but it also reports which fitting and phenophase methods were used, and if the uncertainty was estimated. The `plot` function returns a plot similar to the one constructed above, except that extracted phenophases are also shown the as vertical colored lines. Fig.5 shows that different fitting equation lead to very similar fitted values on the example from Torgnon data. For the sake of robustness, in such situation it is preferable to choose a fitted equation rather than a spline fit. Let's decide to choose the fitting from Gu. Now let's look from closer how do the different phenophase extraction methods impact when applied to the same fitted curve in fig.9 (and note the use of `update` generic function with method `phenopix`):

```
> fit4.trs <- update(fit4, 'trs', plot=FALSE)
> fit4.klosterman <- update(fit4, 'klosterman', plot=FALSE)
> fit4.gu <- update(fit4, 'gu', plot=FALSE)
> par(mfrow=c(2,2), oma=rep(5,4,4,2), mar=rep(0,4))
> plot(fit4.trs, type='n', main='', xaxt='n')
> mtext('trs', 3, adj=0.1, line=-2)
> plot(fit4.klosterman, type='n', main='', xaxt='n', yaxt='n')
> mtext('klosterman', 3, adj=0.1, line=-2)
> plot(0, type='n', axes=FALSE, xlab='', ylab='')
> plot(fit4.gu, type='n', main='', yaxt='n')
> axis(4)
> mtext('gu', 3, adj=0.1, line=-2)
```

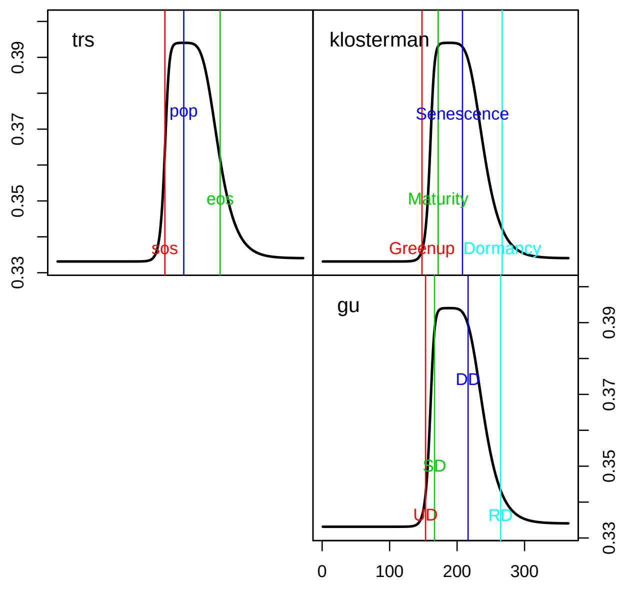

Figure 9: Three phenophase methods applied to the Gu fitting

The `trs` thresholds (50% of increasing and decreasing trajectory) hold a different meaning compared to Klosterman and Gu phenophases. The latter two show good correspondence except that the Klosterman s beginning of senescence occurs later compared to correspondent phase in Gu thresholds (i.e DD, downturn date).

In this paragraph we have shown 4 different approaches to matematically describe the seasonal trajectory of greenness, with additionally 5 methods to extract phenophases on the obtained curves. The combination of curves and phenophase methods leads to as many as 20 possible approaches to describe a seasonal trajectory. Sometimes it could be useful to make a decision on which curves and phenophases to use, without computing the uncertainty on all of them. To do so we have designed two functions that provide a quick overview on what would be the best fit and phenophase method for your actual trajectory. Here is how to compute the 20 combinations of fit and uncertainty in a single function:

```
> explored <- greenExplore(filtered.data$max.filtered)
```

```
[1] "Fitting spline 1/5"
[1] "Fitting Beck 2/5"
[1] "Fitting Elmore 3/5"
[1] "Fitting Klosterman 4/5"
[1] "Fitting Gu 5/5"
```

explored is a list with 20 + 1 elements, i.e. the 20 combinations + a vector containing the RMSEs from each of the 4 fittings. This object will only be used as argument of the `plotExplore()` function (fig.10):

```
> plotExplore(explored)
```

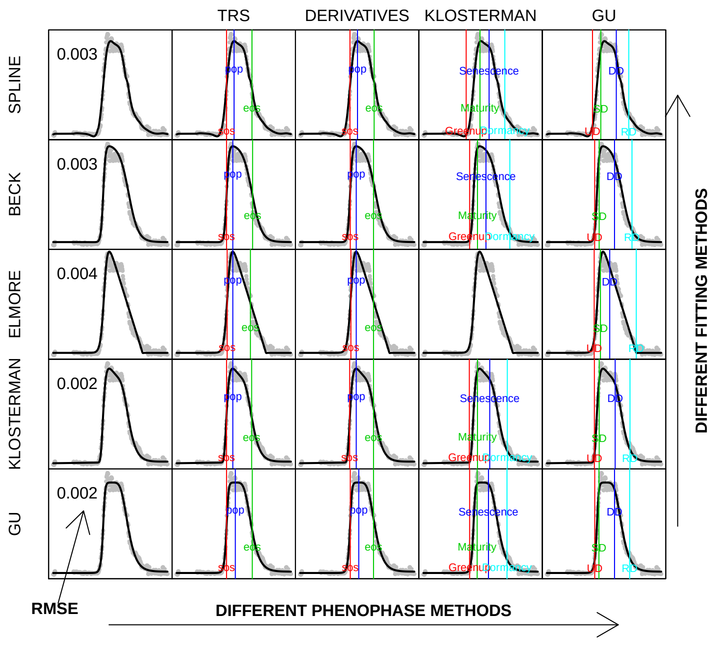

Figure 10: Overview of all combinations of curves and fits as obtained by the `plotExplore` function

The plot in fig.10 shows the impact of different fittings (moving up-downwards) and different phenophases (from left to right) on the same data (Torgnon grassland). The RMSE for each of the four fitting methods is also annotated in the first column. This plot might be useful to choose the most appropriate fitting and thresholding methods on your data.

`greenProcess` is a wrapper function that allows to define the fitting and phenophase methods as arguments. The "primitive" functions that actually perform the fits are the following:

`BeckFit`, `ElmoreFit`, `KlostermanFit` and so on. Their usage is generally:

```
> str(ElmoreFit)

function (ts, uncert = FALSE, nrep = 100, ncores = "all", sf = quantile(ts, 
    probs = c(0.05, 0.95), na.rm = TRUE))
```

with the most important argument beeing `ts`, the time series. Compared to using `greenProcess`, the single fitting functions have the advantage to allow more flexibility but in general the user won't need to use them.

The phenophase extraction methods also have a dedicated wrapper function already embedded in the `greenProcess()` function, `PhenoExtract()` which usage is:

```
> str(PhenoExtract)

function (data, method = "trs", uncert = FALSE, breaks = 3, envelope = "quantiles",
    quantiles = c(0.1, 0.9), plot = TRUE, sf, ...)
```

where the argument `method` allows to choose the phenophase method. Note that input data in this case should be a fitted time series in output from e.g. `FitDoubleLogElmore` and not a `phenopix` object in output from `greenProcess`. Here is an example:

```
> fit.elmore <- greenProcess(filtered.data$max.filtered,
+         'elmore',
+         'trs',
+         plot=FALSE
+         )
> phenopix::extract(fit.elmore, 'metrics')

        sos         eos         los         pop         mgs         rsp
162.0000000 236.0000000  74.0000000 180.0000000   0.3878086          NA
        rau        peak         msp         mau
         NA   0.4016648   0.3696331   0.3671837

> fit.elmore.2 <- ElmoreFit(filtered.data$max.filtered)
> PhenoExtract(fit.elmore.2, 'trs', plot=FALSE)

$metrics
        sos         eos         los         pop         mgs         rsp
162.0000000 236.0000000  74.0000000 180.0000000   0.3878086          NA
        rau        peak         msp         mau
         NA   0.4016648   0.3696331   0.3671837


$unc.df
NULL

> try(PhenoExtract(fit.elmore, plot=FALSE))  ## will fail

$metrics
 sos  eos  los  pop  mgs  rsp  rau peak  msp  mau
```

```
  NA    NA    NA    NA    NA    NA    NA    NA    NA    NA

$unc.df
NULL
```

10 The uncertainty estimation

One main functionality of the package is the uncertainty estimation. This is performed in different ways depending on the fitting equation. The basic idea behind the uncertainty estimation is how good the smoothing curve fits to the data. The residuals between fitted and observed is therefore used to generate random noise to the data and fitting is applied recursively to randomly - noised original data. This procedure results in an ensemble of curves, curve parameters and extracted phenophases that represent the uncertainty estimate. The uncertainty on curve parameters is automatically propagated to phenophase extraction. In the following example the uncertainty estimation is performed on Torgnon grassland data fitted with the approach of Klosterman et al. (2014), with 100 replications. Here is the code:

```
> fit.complete <- greenProcess(ts = filtered.data$max.filtered,
+         fit = 'gu',
+         threshold= 'gu',
+         plot = FALSE,
+         uncert = TRUE,
+         nrep = 100)

[1] "estimated computation time (8 cores): 1 mins"
```

And here is `fit.complete` printed:

```
> print(fit.complete)

 #### phenopix time series processing ####

FITTING: GU

PREDICTED VALUES:
     Index         predicted
 Min.   :  1.0   Min.   :0.3331
 1st Qu.:135.0   1st Qu.:0.3333
 Median :210.0   Median :0.3353
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3649
 Max.   :365.0   Max.   :0.3940

FITTING EQUATION:
expression(y0 + (a1/(1 + exp(-(t - t01)/b1))^c1) - (a2/(1 + exp(-(t -
    t02)/b2))^c2))

FITTING PARAMETERS:
          y0           a1           a2          t01          t02           b1
8.491011e-03 9.902810e-01 9.755980e-01 1.623428e+02 1.870489e+02 2.684337e+00
          b2           c1           c2
1.718645e+01 5.607741e-01 1.522668e+01

THRESHOLDS: GU  ENVELOPE:QUANTILES
          UD       SD       DD       RD   maxline    baseline        prr
10% 153.0528 166.4731 215.5336 264.5324 0.9961082 0.008501414 0.07278270
50% 153.1725 166.5865 216.0148 264.8879 0.9979358 0.009930979 0.07374140
90% 153.2739 166.6854 216.2015 265.2475 0.9998086 0.010956659 0.07441691
            psr plateau.slope
10% -0.02009070 -1.434371e-04
50% -0.01983884 -8.887847e-05
90% -0.01962452 -1.209037e-05

UNCERTAINTY: TRUE
 N of replications = 100

HYDROLOGICAL DAY OF YEAR: FALSE
```

As you can see from the output, the default behavior of `greenProcess()` for the computation of uncertainty is to provide the median, 10th and 90th percentile of the uncertainty ensemble. This may be changed by modifying the `envelope` argument. The other possible option is `min-max` to get minimum mean and maximum. In addition, the quantiles to be chosen with `envelope` = quantiles can be changed by modifying the `quantile` argument. Here is the example:

```
> print(update(fit.complete, 'gu', envelope='min-max', plot = FALSE))
```

```
 #### phenopix time series processing ####

FITTING: GU

PREDICTED VALUES:
     Index         predicted
 Min.   :  1.0   Min.   :0.3331
 1st Qu.:135.0   1st Qu.:0.3333
 Median :210.0   Median :0.3353
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3649
 Max.   :365.0   Max.   :0.3940

FITTING EQUATION:
expression(y0 + (a1/(1 + exp(-(t - t01)/b1))^c1) - (a2/(1 + exp(-(t -
    t02)/b2))^c2))

FITTING PARAMETERS:
          y0           a1           a2          t01          t02           b1
8.491011e-03 9.902810e-01 9.755980e-01 1.623428e+02 1.870489e+02 2.684337e+00
          b2           c1           c2
1.718645e+01 5.607741e-01 1.522668e+01

THRESHOLDS: GU  ENVELOPE:MIN-MAX
           UD       SD       DD       RD   maxline    baseline        prr
min  152.9372 166.3767 214.8171 264.3923 0.9936863 0.005844837 0.07202325
mean 153.1709 166.5814 215.8957 264.9011 0.9979170 0.009811857 0.07368706
max  153.3705 166.8085 216.3522 265.7162 1.0015688 0.012141509 0.07526725
             psr plateau.slope
min  -0.02018892 -1.775118e-04
mean -0.01984705 -8.099987e-05
max  -0.01941125  5.731581e-05

UNCERTAINTY: TRUE
 N of replications = 100

HYDROLOGICAL DAY OF YEAR:
```

Beside the few options available by default and described above, the uncertainty data.frame is accessible via the `extract` command, by running:

```
> phenopix::extract(fit.complete, 'metrics.uncert') ## get threshold uncertainty data`
> phenopix::extract(fit.complete, 'params.uncert') ## get parameters of each fitting curve
```

For example, if you want to use phenophases extracted from the true model and construct uncertainty envelope on them, you can access the uncertainty data.frame by the commands given above. Note than when the uncertainty is computed, also the `plot` function changes its behavior, in that it also shows the uncertainty curve ensemble and an error bar on extracted phases (fig.10.

```
> plot(fit.complete, type='p', pch=20)
```

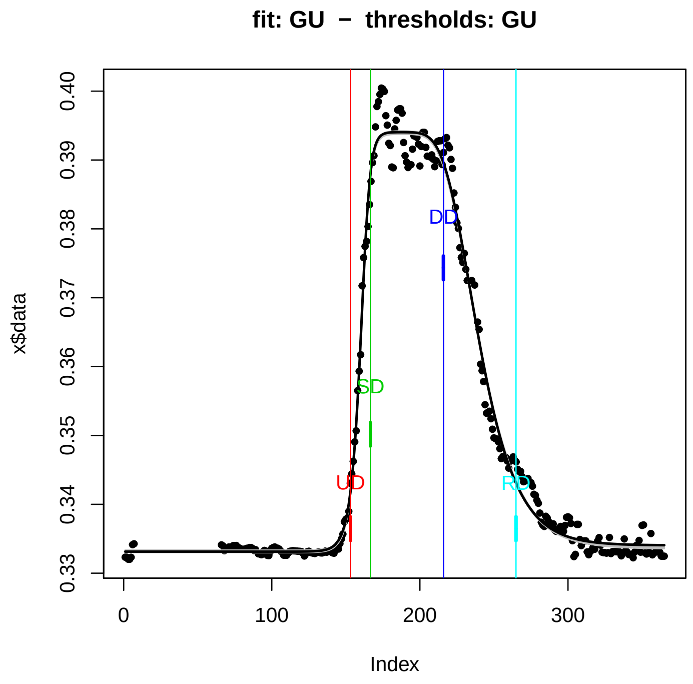

Figure 11: The Uncertainty Estimation (100 rep) on Klosterman fit and Gu phenophases

The distribution of uncertainty parameters (phenophases + curve parameters) can also be evaluated by means of box-plots with an extra option to the default `plot` method:

```
> plot(fit.complete, what='thresholds')
```

By using the `update` function you can also extract phenophases according to a different method, without refitting the data. Here is the code:

```
> update(fit.complete, 'klosterman', plot=FALSE)

 #### phenopix time series processing ####

FITTING: GU

PREDICTED VALUES:
     Index          predicted
 Min.   :  1.0   Min.   :0.3331
 1st Qu.:135.0   1st Qu.:0.3333
 Median :210.0   Median :0.3353
 Mean   :210.4   Mean   :0.3502
 3rd Qu.:289.0   3rd Qu.:0.3649
 Max.   :365.0   Max.   :0.3940

FITTING EQUATION:
expression(y0 + (a1/(1 + exp(-(t - t01)/b1))^c1) - (a2/(1 + exp(-(t -
    t02)/b2))^c2))

FITTING PARAMETERS:
          y0           a1           a2          t01          t02           b1
8.491011e-03 9.902810e-01 9.755980e-01 1.623428e+02 1.870489e+02 2.684337e+00
          b2           c1           c2
1.718645e+01 5.607741e-01 1.522668e+01

THRESHOLDS: KLOSTERMAN  ENVELOPE:QUANTILES
    Greenup Maturity Senescence Dormancy
10%     148      172        208      267
50%     148      172        208      268
90%     148      172        208      269

UNCERTAINTY: TRUE
 N of replications = 100

HYDROLOGICAL DAY OF YEAR:
```

Phenophase extraction method `trs` allows to set an extra argument that controls which threshold in the trajectory be used. Default is when 50% of seasonal maximum gcc is reached (indicated as 0.5). Let's see how it works:

```
> phenopix::extract(update(fit.complete, 'trs', plot=FALSE), 'metrics')

    sos eos los pop       mgs rsp rau      peak       msp       mau
10% 160 241  81 188 0.3876677  NA  NA 0.3938797 0.3645451 0.3600082
50% 160 242  82 188 0.3877659  NA  NA 0.3939921 0.3646647 0.3601956
90% 160 242  82 189 0.3880413  NA  NA 0.3941073 0.3648008 0.3613780

> ## default to 50% of increasing and decreasing traj.
> phenopix::extract(update(fit.complete, 'trs', trs=0.2, plot=FALSE), 'metrics')

    sos eos los pop       mgs rsp rau      peak       msp       mau
10% 154 262 108 188 0.3791217  NA  NA 0.3938797 0.3500856 0.3443156
50% 154 263 109 188 0.3792607  NA  NA 0.3939921 0.3501765 0.3445131
90% 154 264 110 189 0.3793431  NA  NA 0.3941073 0.3503040 0.3449001

> ## changed to 20%
```

There is a last method to define thresholds on a time series that does not need a fitting. It implements the use of break points from the package `strucchange` and works as follows:

```
> print(PhenoBP(x = filtered.data$max.filtered,
+          breaks = 3, plot = FALSE,
+          confidence= 0.99))

             bp1        bp2        bp3
0.5%  1528408800 1532901600 1537048800
mean  1528495200 1532988000 1537135200
99.5% 1528581600 1533074400 1537221600
```

The user can set the maximum number of breakpoints to be identified, the confidence interval at which the calculation must be performed and an option or a plot. The output dataframe contains the day of the year for each of the breakpoints and their respective confidence intervals.

All fitting and some phenophase methods illustrated above can be combined in order to obtain a sort of model ensemble of phenophases. These phases with an uncertainty estimation can be useful when phenology is required as input for biological models. The function `combineUncertainty` with its associated functions `summarizePhases` and `plotSum` are designed to this end. The following lines show how to use them.

```
> combined.fit <- combineUncertainty(na.approx(filtered.data$max.filtered), nrep=20)
```

```
[1] "Fitting BECK 1/4"
[1] "estimated computation time (8 cores): 1 mins"
[1] "Fitting ELMORE 2/4"
[1] "estimated computation time (8 cores): 0 mins"
[1] "Fitting KLOSTERMAN 3/4"
[1] "estimated computation time (8 cores): 6 mins"
[1] "Fitting GU 4/4"
[1] "estimated computation time (8 cores): 1 mins"
```

```
> ## 20 replications for each fitting
> names(combined.fit)
```

```
[1] "trs"         "derivatives" "klosterman"  "gu"          "fits"
```

```
> ## a dataframe for each phenoMethod + a list with all fittings
> fit.summary <- summarizePhases(combined.fit, across.methods=TRUE)
> ## again a list with one element for each fitting method + two additional items
> ## if across.methods is TRUE, which combines gu + klosterman phenophase methods
> ## in a single method, and the same happens for trs and derivatives
> plotSum(filtered.data$max.filtered, fit.summary, which='kl.gu')
```

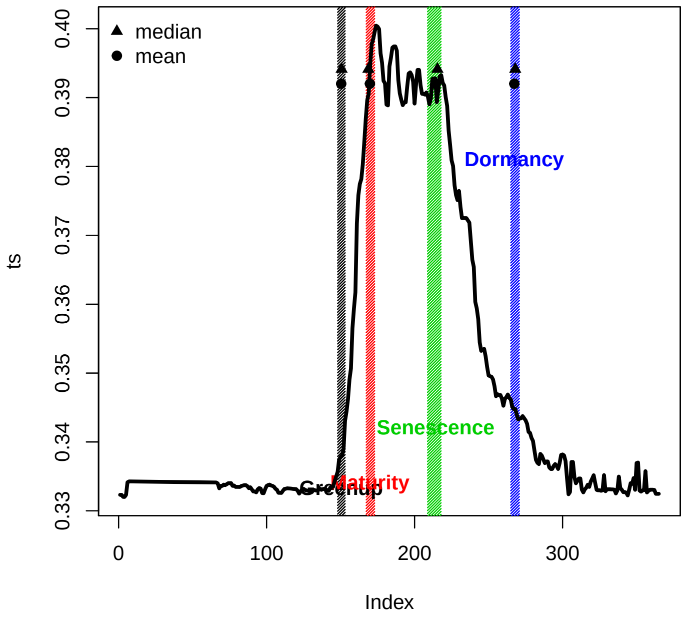

Figure 12: Ensemble of phases plotted with function `plotSum`

11 Pushing forward the analysis: pixel - based phenology

In order to thoroughly exploit the capabilities of an imagery archive, spatial analysis represents the most promising feature. Hence, specific functions are built to fit curves and extract phenophases on each pixel included in a region of interest instead of averaging the greenness index over the entire ROI. A specific vignette of this package is devoted to explain details on the pixel-based analysis.

12 Other functions

A number of other functions are available in the package, that do not necessarily enter the main workflow of the processing but still may be worth to mention.

`plotVI()` gets in input a VI.data data.frame as produced by extractVIs and reproduces the default plots from extractVIs. Useful when you use `extractVIs` with argument `begin` switched on and you want to update existing plots. `hydrodoy` to convert from and to hydrological day of year, to be used in conjuction with `greenProcess` with hydro=TRUE

13 Summary and future of the package

The `phenopix` package is currently available for download from the R-forge. The package was tested on approx 300 site-years belonging to the phenocam imagery archive, on the camera network of the project e-pheno and will soon be deployed to process images in the European Network of Flux Towers. A paper presenting the software will be soon published.

The R package `phenopix` is available on CRAN and directily within R by running the command:

```
> install.packages("phenopix")
```

It is under constant maintainance by Gianluca Filippa and the co-authors. Feel free to write me in case of any problem with the package.

14 References

Filippa, G.et al.Phenopix: A R package for image-based vegetation phenology.Agric. For. Meteorol220,141–150 (2016)

14.1 Referenced use of phenopix in publications

Cerasoli, Sofia & Campagnolo, Manuel & Faria, Joana & Nogueira, Carla & Caldeira, Maria. (2018). On estimating the gross primary productivity of Mediterranean grasslands under different fertilization regimes using vegetation indices and hyperspectral reflectance. Biogeosciences. 15. 5455-5471. 10.5194/bg-15-5455-2018.

Zhang, Qiang; Kong, Dongdong; Shi, Peiyun; Singh, Vijay; Peng, Sun. (2017). Vegetation phenology on the Qinghai-Tibetan Plateau and its response to climate change (1982-2013). Agricultural and Forest Meteorology. 248. 407-17. 10.1016/j.agrformet.2017.10.026.

Arslan, Ali; Tanis, Cemal; Metsämäki, Sari; Aurela, Mika; Böttcher, Kristin; Linkosalmi, Maiju; Peltoniemi, Mikko. (2017). Automated Webcam Monitoring of Fractional Snow Cover in Northern Boreal Conditions. Geosciences. 7(3). 10.3390/geosciences7030055.

Snyder, K.A.; Wehan, B.L.; Filippa, G.; Huntington, J.L.; Stringham, T.K.; Snyder, D.K. Extracting Plant Phenology Metrics in a Great Basin Watershed: Methods and Considerations for Quantifying Phenophases in a Cold Desert. Sensors 2016, 16, 1948.

Helbig, Manuel; Quinton, William; Sonnentag, Oliver. (2017). Warmer spring conditions increase annual methane emissions from a boreal peat landscape with sporadic permafrost. Environmental Research Letters. 12. 10.1088/1748-9326/aa8c85.

Pinjarla, Bhavani; Roy, Parth; Vishnubhotla, Chakravarthi; Kanawade, Vijay. (2017). Satellite Remote Sensing for Monitoring Agriculture Growth and Agricultural Drought Vulnerability Using Long-Term (1982–2015) Climate Variability and Socio-economic Data set. Proceedings of the National Academy of Sciences, India Section A: Physical Sciences. 87. 10.1007/s40010-017-0445-7.

14.2 General background for digital image analysis and phenocams

Richardson, A.D., Braswell, B.H., Hollinger, D.Y., Jenkins, J.P., Ollinger, S.V., 2009. Near-surface remote sensing of spatial and temporal variation in canopy phenology. Ecological Applications 19, 1417-28.

Sonnentag, O., Hufkens, K., Teshera-Sterne, C., Young, A.M., Friedl, M., Braswell, B.H., Milliman, T., OKeefe, J., Richardson, A.D., 2012. Digital repeat photography for phenological research in forest ecosystems. Agricultural and Forest Meteorology 152, 159-177.

Wingate, L., Ogee, J., Cremonese, E., Filippa, G., Mizunuma, T., Migliavacca, M., Moisy, C., Wilkinson, M., Moureaux, C., Wohlfahrt, G., Hammerle, A., Hortnagl, L., Gimeno, C., Porcar-Castell, A., Galvagno, M., Nakaji, T., Morison, J., Kolle, O., Knohl, A., Kutsch, W., Kolari, P., Nikinmaa, E., Ibrom, A., Gielen, B., Eugster, W., Balzarolo, M., Papale, D., Klumpp, K., Kostner, B., Grunwald, T., Joffre, R., Ourcival, J.M., Hellstrom, M., Lindroth, A., Charles, G., Longdoz, B., Genty, B., Levula, J., Heinesch, B., Sprintsin, M., Yakir, D., Manise, T., Guyon, D., Ahrends, H., Plaza-Aguilar, A., Guan, J.H., Grace, J., 2015. Interpreting canopy development and physiology using the EUROPhen camera network at flux sites. Biogeosciences Discussions 12, 7979-8034.

14.3 Curve fitting and filtering

Forkel, M., Migliavacca, M., Thonicke, K., Reichstein, M., Schaphoff, S., Weber, U., Carvalhais, N., 2015. Codominant water control on global interannual variability and trends in land surface phenology and greenness. Global Change Biology 21, 3414-3435.

Gu L, Post WM, Baldocchi D, Black TA, Suyker AE, Verma SB, Vesala T, Wofsy SC. (2009) Characterizing the Seasonal Dynamics of Plant Community Photosynthesis Across a Range of Vegetation Types. In: Phenology of Ecosystem Processes (Ed: Noormets A, Springer New York), pp 35-58.

Klosterman ST, Hufkens K, Gray JM, Melaas E, Sonnentag O, Lavine I, Mitchell L, Norman R, Friedl MA, Richardson A D (2014) Evaluating remote sensing of deciduous forest phenology at multiple spatial scales using PhenoCam imagery, Biogeosciences, 11, 4305-4320, doi:10.5194/bg-11-4305-2014.

Migliavacca M., Galvagno M., Cremonese E., Rossini M., Meroni M., Cogliati S., Manca G., Diotri F., Busetto L., Colombo R., Fava F., Pari E., Siniscalco C., Morra di Cella U., Richardson A.D. (2011) Using digital repeat photography and eddy covariance data to model grassland phenology and photosynthetic CO2 uptake. Agricultural and forest meteorology 151:1325-1337.

Papale D, Reichstein M, Aubinet M et al. (2006) Towards a standardized processing of Net Ecosystem Exchange measured with eddy covariance technique: algorithms and uncertainty estimation. Biogeosciences, 3, 571-583.

Sonnentag O., Hufkens K., Teshera-Sterne C., Young A.M., Friedl M., Braswell B.H., Milliman T., O'Keefe J., Richardson A.D. (2012) Digital repeat photography for phenological research in forest ecosystems. Agricultural and forest meteorology 152:159-177.

Zhang X, Friedl MA, Schaaf CB, Strahler AH, Hodges JCF, Gao F, Reed BC, Huete A (2003) Monitoring vegetation phenology using MODIS, Remote Sens. Environ., 84, 471-475.