ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/289355121

phenopix R package vignettes 2/3: spatial (pixel-based analysis) vignette

Method · February 2016

CITATIONS
0

READS
457

1 author:

Gianluca Filippa
Environmental Protection Agency of Aosta Valley
77 PUBLICATIONS **789** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Porter Canyon Experimental Watershed View project

EGU 2020 Latest Developments and Software Tools for Ecosystem and Flux Data Analysis View project

All content following this page was uploaded by Gianluca Filippa on 05 January 2016.

The user has requested enhancement of the downloaded file.

Phenopix: pixel based phenology

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

January 4, 2016

This vignette aims at illustrating specific features of the package `phenopix` related to pixel by pixel analysis of an image archive in the context of spatially explicit phenology. All features are shown based on Torgnon Grassland site (NW ITALY) belonging to the PHENOCAM network and leaded by ARPA Valle d'Aosta. Troughout this vignette relative green (ie. green digital number value relative to the sum of all digital numbers) is defined gcc (so called green chromatic coordinate).

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, sp`

This vignette was run on:

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

R version 3.2.2 (2015-08-14)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.3 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_GB.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] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] zoo_1.7-12   jpeg_0.1-8   phenopix_2.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.11.6       lattice_0.20-29   codetools_0.2-14  gtools_3.4.1
 [5] foreach_1.4.2     R.methodsS3_1.6.1 grid_3.2.2        plyr_1.8.1
 [9] magrittr_1.5      stringi_0.4-1     sp_1.0-16         raster_2.3-40
[13] doParallel_1.0.8  strucchange_1.5-0 R.oo_1.18.0       R.utils_1.33.0
[17] sandwich_2.3-2    rgdal_0.9-1       iterators_1.0.7   stringr_1.0.0
[21] bcp_3.0.1         parallel_3.2.2    SDMTools_1.1-221
```

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

2 Topics covered

This vignette covers the pixel-by-pixel analysis. other available vignettes include the base vignette and Camera NDVI extraction (coming soon).

For a general introduction to the package functions, please see the main vignette by running:

```
> library(phenopix)
```

```
> vignette('phenopix')
```

Since throughout this vignette we often refer to the base vignette, we strongly recommend to read the latter, first.

3 Why a pixel by pixel analysis

Images of a vegetation canopy are usually analysed on a given region of interest (ROI). Red green and blue values of the pixels belonging to the ROI are usually averaged and computed over a season of images, and information on an average phenological signal is drawn. However, recent literature suggested that pixel by

pixel analysis of a ROI may be a promising tool to evaluate within-individuals and between-individuals different phenologies in various contexts, i.e. grasslands (with the possibility to evaluate inter-species diffeences in phenology, spatial variability, and biodiversity) and forests (inter-species differences, age-related differences in phenology, intra-individual differences in phenology as related to e.g. position of leaves on the tree), experimental plots (to see for example the effects of treatments on a portion of the plot) and so on.

4 Install the package

The package `phenopix` is hosted in the r-forge repository and can be installed via the following command:

```
> install.packages("phenopix", repos="http://R-Forge.R-project.org")
```

Note that by running this command you will likely be asked to install the dependencies, which are available via the usual command:

```
> install.packages('package.name')
```

5 Functions overview

Base functions needed for the processing and not covered by this vignette:

- `DrawROI()` to draw a region of interest in your pictures (see phenopix vignette)

- `updateROI()` to apply ROI coordinates to an image of different size (see phenopix vignette)

Functions discussed in the present vignette:

- `splitROI()` to split a ROI in subrois for faster computation.
- `extractVIs()` to extract pixel raw color values and indices
- `spatialFilter()` to filter the data.
- `spatialGreen()` to fit a curve to the data and extract thresholds (aka phenophases)
- `plotSpatial()` to plot results in phenophase maps
- `extractParameters()` to extract thresholds and curve parameters after the pixel-based analysis.
- `greenClusters()` to apply a cluster analysis on extracted phenophases.

6 Prior to the analysis

The spatial analysis consist of extracting green values from each pixel in a region of interest, filter out data, fit a curve and store results. From the point of vue of R objects, this implies the building up of very large objects, which is an issue for R, because it can lead to RAM saturation quite quickly. The first thing a user must do is compute how many pixels are included into the target region of interest. Let's load a sample picture

```
> setwd('/home/gian/sweave/SPATIAL_VIGNETTE/')
> library(jpeg)
> sample.file <- readJPEG('files/20130630T1000.jpg')
> dim(sample.file)
```

```
[1] 428 640   3
```

```
> load('files/roi.data.Rdata', verbose=TRUE) ## this ROI has two
```

```
Loading objects:
  roi.data
```

```
> ## subrois one called fg (foreground) and bg (background)
> ## we focus on foreground
> fg.roi <- roi.data['fg']
> str(fg.roi)
```

```
List of 1
 $ fg:List of 2
  ..$ pixels.in.roi:'data.frame':        273920 obs. of  3 variables:
  .. ..$ rowpos: num [1:273920] 0.00156 0.00313 0.00469 0.00625 0.00781 ...
  .. ..$ colpos: num [1:273920] 0.00156 0.00156 0.00156 0.00156 0.00156 ...
  .. ..$ pip   : int [1:273920] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ vertices     :List of 2
  .. ..$ x: num [1:8] 0.0202 0.233 0.4341 0.5895 0.6656 ...
  .. ..$ y: num [1:8] 0.33 0.352 0.372 0.383 0.383 ...
```

Within the relatively complex structure of the roi object what matters is the number of pixels in the roi which can be accessed by the following command:

```
> table(fg.roi$fg$pixels.in.roi$pip)
```

```
     0      1
180321  93599
```

This is a vector of 0 and 1, where 1 denotes a pixel falling within the ROI. Hence, now we know that our ROI consists of 93599 pixels. As a rough estimate consider that extracting VIs on 4000 images on a ROI of 100000 pixels would produce an R object occupying more than 4.5 Gb space. This is why we recommend to split the ROI into smaller subROIs of no more than 10000 pixels as follows:

```
> splitted.roi <- splitROI(fg.roi, nsplit=10)
> names(splitted.roi)

 [1] "roi1"  "roi2"  "roi3"  "roi4"  "roi5"  "roi6"  "roi7"  "roi8"  "roi9"
[10] "roi10"

> sapply(splitted.roi, function(x) length(which(x$pixels.in.roi$pip==1)))

 roi1  roi2  roi3  roi4  roi5  roi6  roi7  roi8  roi9 roi10
 9359  9359  9359  9359  9359  9359  9359  9359  9359  9368
```

The number of pixels in each subROI is now convenient, we are ready to run the analysis on each of the 10 subROIs. Before doing so it is a good idea to structure various subfolders where objects will be stored (not mandatory, but suggested):

```
> structureFolder('files/')

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

> for (a in 1:length(splitted.roi)) {
+         roi.data <- splitted.roi[a]
+         final.name <- paste0('files/ROI/ROI',a,'/roi.data.Rdata')
+         dir.create(paste0('files/ROI/ROI',a,'/'))
+         dir.create(paste0('files/VI/VI',a,'/'))
```

```
+         save(roi.data, file=final.name)
+ }
```

The first command takes as argument the path to a folder and creates subfolders for the analysis (for more details see the `phenopix` main vignette). The `for` loop renames each subROI and saves it into subfolders, and at the same time creates parallel subfolders in the VI folder. After running the above code you will have a number of numbered folders equal to the number of subrois.

7 Extract vegetation indices with `extractVIs()`

The function `extractVIs()` is explained in the base vignette of the package and the reader is referred to it for an explaination of its use. In the pixel-by-pixel analysis the argument `spatial` is set to TRUE. In the following code the function is already embedded in the for loop that allows to extract each of the 10 subROIs in which your ROI is divided.

```
> img.path <- '/path/to/your/images/'
> roi.paths <- list.files('files/ROI/', full.names=TRUE)
> vi.paths <- list.files('files/VI/', full.names=TRUE)

> for (a in 1:10) {
+         act.roi.path <- paste0(roi.paths[a], '/')
+         act.vi.path <- paste0(vi.paths[a], '/')
+         extractVIs(img.path,act.roi.path,
+                 vi.path=act.vi.path,
+                 begin=NULL, spatial=TRUE,
+                 date.code='yyyy_mm_dd_HHMMSS')
+ }
```

Be aware that pixel extraction can take as long as hours, depending on your image size and image sampling setup. At the end of the process in your Vi subfolder you will have an object called (VI.data.spatial.Rdata). Let's look inside it:

```
> load('files/VI/VI1/VI.data.spatial.Rdata', verbose=TRUE)

Loading objects:
  VI.data

> class(VI.data[[1]])
```

```
[1] "list"
```

```
> head(names(VI.data[[1]]))
```

```
[1] "2013-04-15 12:17:00" "2013-04-15 15:00:00" "2013-04-15 16:00:00"
[4] "2013-04-15 17:00:00" "2013-04-16 10:00:00" "2013-04-16 11:00:00"
```

```
> str(VI.data[[1]][[1]])
```

```
'data.frame':        9359 obs. of  3 variables:
 $ red  : num  169 171 171 172 172 171 171 172 171 171 ...
 $ green: num  177 179 181 180 180 179 179 178 179 179 ...
 $ blue : num  188 190 191 191 191 190 190 190 190 190 ...
```

- Within each element of this list is another list with one element for each time step processed (note the output from names(VI.data[[1]]))

- Within each element of this list is a data.frame with three columns corresponding to the digital number of red green and blue, respectively, and there is a row for each pixel in the ROI.

Quite rarely the user will need to explore contents of the VI.data object, but it is worth knowing its structure. In the previous sections we used a for loop to show how all subROIs are processed. From here on we will present new functions as if they were run on a single subROI. In a final section we will show how to paste again together the output from each subROIs and hence get an output for the entire original ROI.

8 Filter the data with function `spatialFilter`

Next step of the analysis is data filtering. The function responsible for this task is `spatialFilter` which in turn calls the function `autoFilter` which is explained in the base package, including the meaning and the effects of all filters. `spatialFilter` allows to apply the filtering procedure on the VI object produced before. The usage of the function is as follows:

```
> args(spatialFilter)
```

```
function (spatial.list, filter = c("night", "spline", "max"),
    filter.options = NULL, ncores = "all", log.file = NULL)
NULL
```

The argument `spatial.list` will be the `VI.data` object in output from `extractVIs()`. The argument `filter` is exactly as for `autoFilter()` (see the

base vignette) and controls which filters will be used. `filter.options` is also explained in the base vignette. The argument `ncores` allows the user to set the number of cores to be used in parallel computation (it is highly recommended to use parallell computation). The argument `log.file` allows to write a txt file to monitor the progress of the processing. If you want to use this option (which is by default switched off) you can enter a quoted path to a folder (e.g. `'path/to/log/file/'`) were a file named 'log.txt' will be saved and updated with one row written for each pixel processed (and a percent progress). The filtering process is time consuming. Consider that filtering 10000 pixels of a series of 5000 images requires about 7 hours on a 4-cores CPU. The object in output from the filtering process is a multivariate zoo series with daily filtered gcc data and one column for each pixel. In contrast to the behavior of `autoFilter()`, where there is a column in output from each filtering step, `spatialFilter()` returns only data filtered with the last filtering method specified in `filter` argument, by default 'max'.

Following the example above, with the large ROI splitted into 10 subROIs, here will be the code to filter out the data:

```
> for (a in 1:10) {
+         act.vi.path <- vi.paths[a]
+         load(paste0(act.vi.path, '/VI.data.spatial.Rdata'))
+         filtered.tmp <- spatialFilter(VI.data, log.file='/home/gian')
+         save.name <- paste0(act.vi.path,'/filtered.tmp.Rdata')
+         save(filtered.tmp, file=save.name)
+ }
```

With this for loop for each subROI we load the VI extracted, we filter them out and we save the filtering product in the same subfolder. Note that I specified a path for saving the above mentioned log file, which will be written in my home folder in order to track the ongoing process. Out of the package I have written a small Rscript that allows reading the txt file without opening it from an ubuntu terminal. This function/script is not included in the package but I incude the code here for convenience.

```
> all.lines <- readLines('log.txt')
> cat(paste(length(all.lines)-1, 'pixels done\n'))
> cat(paste(all.lines[length(all.lines)], '\n'))
```

If you save this script in a file named `log.R` from a console (opened in the folder where your log.txt is being written) you simply run `Rscript log.R`. In

this way you can check the progress of your processing.

Let's now look at the structure of one of those filtered objects (it is a multivariate zoo time series).

```
> load(paste0(vi.paths[1],'/filtered.tmp.Rdata'), verbose=TRUE)

Loading objects:
  filtered.tmp

> summary(filtered.tmp[,1])

     Index                     filtered.tmp[, 1]
 Min.   :2013-04-15 00:00:00   Min.   :0.3298
 1st Qu.:2013-06-19 00:00:00   1st Qu.:0.3331
 Median :2013-08-23 00:00:00   Median :0.3353
 Mean   :2013-08-23 00:14:56   Mean   :0.3455
 3rd Qu.:2013-10-27 00:00:00   3rd Qu.:0.3515
 Max.   :2013-12-31 00:00:00   Max.   :0.4000
```

```
> sampling.vect <- sample(dim(filtered.tmp)[2], 500)
> library(zoo)
> plot(filtered.tmp[,sampling.vect], plot.type='single', col='grey',
+         ylab='subROI 1: gcc trajectories of 500 random pixels + roi average')
> avg.trajectory <- apply(filtered.tmp,1,mean, na.rm=TRUE)
> avg.trajectory <- zoo(avg.trajectory, order.by=index(filtered.tmp))
> lines(avg.trajectory, lwd=3)
```

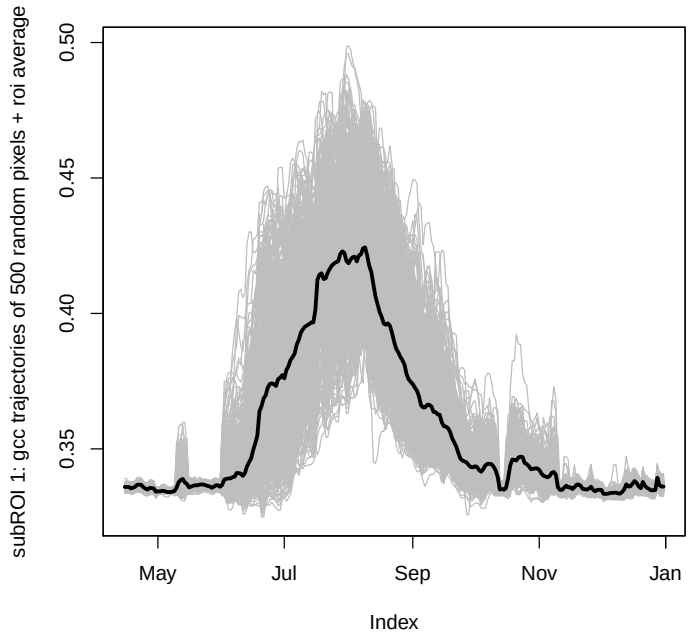

Figure 1: The seasonal trajectory of gcc of 500 random pixels from subROI 1

The plot in fig. 1 shows the seasonal trajectories of gcc for 500 random pixels in the subROI 1 together with the overall average seasonal trajectory. For reference, compare them to the filtered trajectory of the ROI-averaged approach in the base vignette, but remeber this is only one out of 10 subROIs!

9 Fit a curve to the filtered data with function `spatialGreen()`

After filtering, pixel gcc seasonal trajectories are ready to be fitted with a smoothing spline or a double logistic function. Details on fitting options are

available in the main vignette. here we illustrate the usage of the function that performs fittings pixel by pixel, i.e. `spatialGreen()`. Here is the usage:

```
> args(spatialGreen)

function (filtered.data, fit, threshold, ncores = "all", log.file = NULL)
NULL
```

The argument `filtered.data` will be the object in output from `spatialFilter()` function. The argument `fit` allows to choose a fitting function; the argument `threshold` allows to choose a phenophase method. Arguments `ncores` and `log.file` are as for `spatialFilter()`.

Following our 10 subROIs example the code to fit a smoothing spline to each pixel and extract phenophases according to the Klosterman method will be the following:

```
> ##first we create appropriate folders to store the fitted data
> for (a in 1:10) dir.create(paste0('files/FITTED/fitted',a), recursive=TRUE)
> ## list and grep all filtered files
>         filtered.files <- list.files(vi.paths, full.names=TRUE, recursive=TRUE)
> filtered.files <- filtered.files[grep('filtered.tmp', filtered.files)]
> fitted.paths <- list.files('/files/FITTED/', full.names=TRUE)
> for (a in 1:10) {
+         act.file.to.process <- filtered.files[a]
+         act.fitted.path <- fitted.paths[a]
+         load(act.file.to.process, verbose=TRUE)
+         fitted.tmp <- spatialGreen(filtered.tmp, fit='spline',
+                 threshold='klosterman',
+                 log.file='/home/gian')
+         save.name <- paste0(act.fitted.path,'/fitted.tmp.RData')
+         save(fitted.tmp, file=save.name)
+ }
```

The R object in output from `spatialGreen()` (in this example, `fitted.tmp` is a list with length equal to the number of pixels where each element is an object of class `phenopix` that can be handled with the same methods explained in the base vignette. One would like to see for example the fitted curves for all pixels and plot them all together. For such analysis we can use the function `extract` to extract the fitted values for each pixel. Since `fitted.tmp` is a list, we will use a `sapply()` function as follows:

```
> load('files/FITTED/fitted1/fitted.tmp.RData', verbose=TRUE)
> ## extraction of a matrix (because of sapply)
> fitted.values <- sapply(fitted.tmp, extract, what='fitted')
> ## convert to a zoo object
> fitted.values2 <- zoo(fitted.values,
+         order.by=index(extract(fitted.tmp[[1]], what='fitted')))

> sampling.vect <- sample(dim(filtered.tmp)[2], 500)
> plot(fitted.values2[,sampling.vect], plot.type='single', col='grey',
+         ylab='subROI 1: gcc spline fitting of 500 random pixels')
```

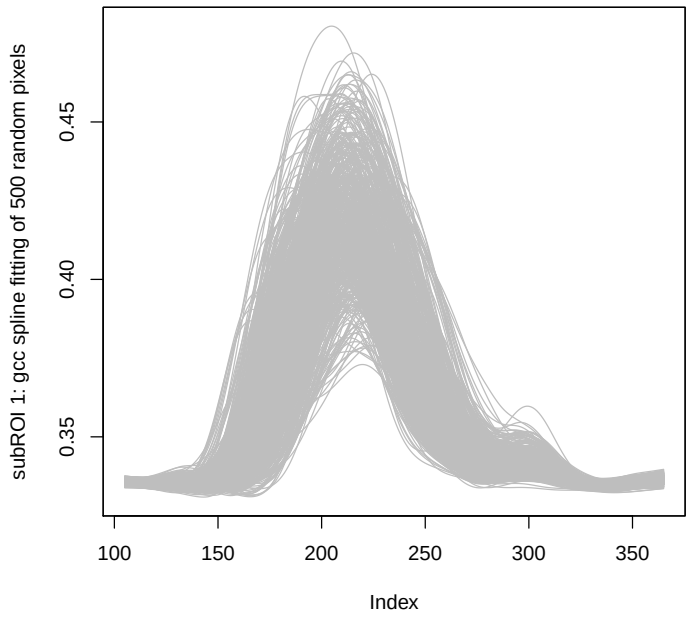

Figure 2: The seasonal trajectory of fitted gcc of 500 random pixels from subROI 1

10 Extracting phenophases with `extractParameters()`

However, looking at hundreds of seasonal trajectories is not so informative. Much better will be to extract curve parameters, and especially phenophases

from all pixels and find a convenient way to show them. A specific function is designed to extract useful information from each fitted pixel.

`extractParameters` allows to extract phenophases. Its usage is:

```
> args(extractParameters)

function (list, update = NULL, ...)
NULL
```

where `list` is the object in output from `spatialGreen()`. The argument update can be one between 'trs', 'derivatives', 'klosterman' and 'gu' to extract different phenophases compared to the default phenophases extracted with `spatialGreen()`, similarly to the `update()` function described in the base vignette. Further arguments (currently one) can be defined in the function. For example let's extract the default thresholds (klosterman method) from `fitted.tmp`:

```
> spatial.phenophases.klosterman <- extractParameters(fitted.tmp)
```

If you look at the structure of the object `spatial.phenophases` you'll see it is a dataframe with a column for each phase (so the number depends on the phenophase extraction method used) and a last column reporting the RMSE of the fit. The number of rows is th number of pixels analysed.

```
> summary(spatial.phenophases.klosterman)

    Greenup         Maturity       Senescence       Dormancy
 Min.   :134.0   Min.   :164.0   Min.   :197.0   Min.   :212.0
 1st Qu.:149.0   1st Qu.:219.0   1st Qu.:229.0   1st Qu.:253.0
 Median :154.0   Median :225.0   Median :233.0   Median :289.0
 Mean   :159.2   Mean   :219.7   Mean   :236.1   Mean   :279.5
 3rd Qu.:170.0   3rd Qu.:229.0   3rd Qu.:239.0   3rd Qu.:291.0
 Max.   :238.0   Max.   :259.0   Max.   :299.0   Max.   :365.0
      RMSE
 Min.   :0.002245
 1st Qu.:0.004144
 Median :0.004921
 Mean   :0.005018
 3rd Qu.:0.005762
 Max.   :0.010310
```

Note the last column with RMSE, showing how good the fit is on each pixel. Note also how large can be the range of a given phenophase across all pixels.

Next, one may want to use the `update` function to extract different phenophases on the same pixels or same method with different options. Here are two examples:

```
> spatial.phenophases.trs <- extractParameters(fitted.tmp, update='trs')
> spatial.phenophases.trs2 <- extractParameters(fitted.tmp, update='trs', trs=0.25)
```

And their comparison:

```
> plot(spatial.phenophases.trs$sos,
+         spatial.phenophases.trs2$sos,
+         pch=20, xlim=c(150, 210),
+         ylim=c(140, 200),
+         xlab='start of season as doy when 50% GCC is reached',
+         ylab='start of season as doy when 25% GCC is reached')
```

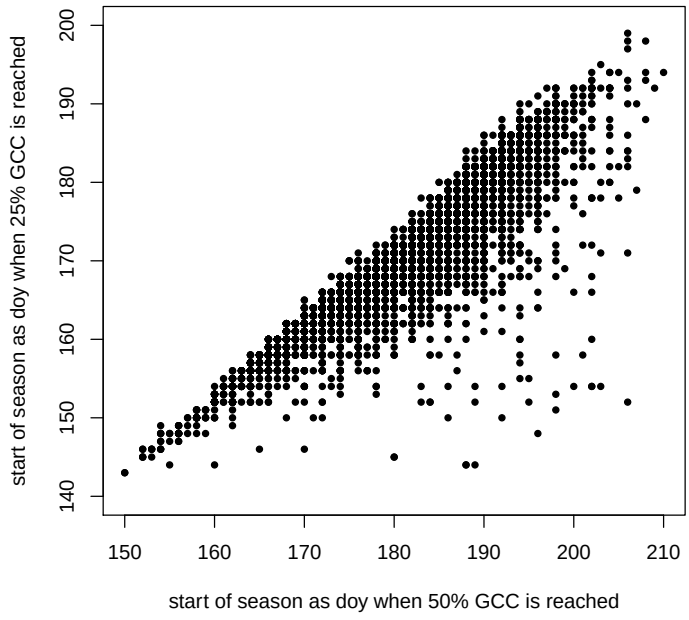

Figure 3: Scatter comparing phenophase trs methods with two different thresholds

The second method applied allows to update the analysis by extracting phenophases with the 'trs' method, where beginning and end of the growing season are defined as the day when gcc reaches the 25% values of its maximum

either in the increasing and decreasing part of the curve.

11 Rendering pixel by pixel phenophases in a phenophase map with `plotSpatial`

Now let's render the phenophases extracted for each pixel on a so called phenophase map. Before doing so, we have to put togeth the results for all rois analysed. Here is the code:

```
> dir.create('files/PHENOPHASES/')
> global.phenophases.list <- list()
> for (a in 1:10) {
+         act.fitted.path <- fitted.paths[a]
+         act.number.big <- substr(act.fitted.path,
+                 nchar(act.fitted.path)-10,
+                 nchar(act.fitted.path)) ## this line requires
+ ## that you don't have numbers
+    ## in last ten characters of your folder path
+         track.number <- as.numeric(gsub("[^0-9]", "", act.number.big), "")
+         to.load <- list.files(act.fitted.path, full.names=TRUE)
+         load(to.load, verbose=TRUE)
+         spatial.phenophases.klosterman.tmp <- extractParameters(fitted.tmp)
+         global.phenophases.list[[track.number]] <- spatial.phenophases.klosterman.tmp
+ }
> global.phenophases <- NULL
> for (a in 1:10) {
+         tmp.data.frame <- global.phenophases.list[[a]]
+         global.phenophases <- rbind(global.phenophases, tmp.data.frame)
+ }
```

There must be special care on how the various subROI objects are binded together. This is because the file listing within R are ordered in a binary style and not with increasing numbering, resulting in a mismatch between file listing and roi numbers. This is why we have to retrieve the number of the roi we are processing (with `act.number.big` and `track.number`), use this index to build an orderd list (`global.phenophases.list`) and then unlist it and bind objects to get the final `global.phenophases` dataset. Such dataset is computed by fitting each pixel with the `spline` method. Hence, there is no curve parameters behind such fitting. Conversely, the data.frame `global.phenophases.klosterman` is

the result of `extractParameters` run after fitting each pixel with the Klosterman equation, where to each pixel are associated not only the phenophases, but also the curve parameters of the fitted equations. Hence here you are the names of the two data.frames:

```
> names(global.phenophases)
```

```
[1] "Greenup"    "Maturity"   "Senescence" "Dormancy"   "RMSE"
[6] "gsl"
```

```
> names(global.phenophases.klosterman)
```

```
 [1] "Greenup"    "Maturity"   "Senescence" "Dormancy"   "a1"
 [6] "a2"         "b1"         "b2"         "c"          "B1"
[11] "B2"         "m1"         "m2"         "q1"         "q2"
[16] "v1"         "v2"         "RMSE"
```

As you see the phenophase names are the same, as we have used the same phenophase method (i.e. Klosterman) but only the `global.phenophases.klosterman` data.frame also containes the curve parameters.

Now we are ready to use the `plotSpatial` function. Here is the usage:

```
> args(plotSpatial)
```

```
function (data, param, roi.data.path, image.path, probs = c(0.01,
    0.99), ...)
NULL
```

`data` will be the `global.phenophases` object created before, `param` is a character matching one of `names(data)`, i.e. the parameter that will be plotted. `roi.data.path` is the complete path to our full ROI data, `image.path` will be the full path to the image we will use as a background. Let's proceed:

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> plotSpatial(global.phenophases, param='Greenup',
+         roi.data.path='files/roi.data.Rdata',
+         image.path='files/20130630T1000.jpg',
+         col=colors(100),
+         axis.args=list(col.axis='white'),
+         legend.args=list(text='Greenup (doy)', col='white'))
```

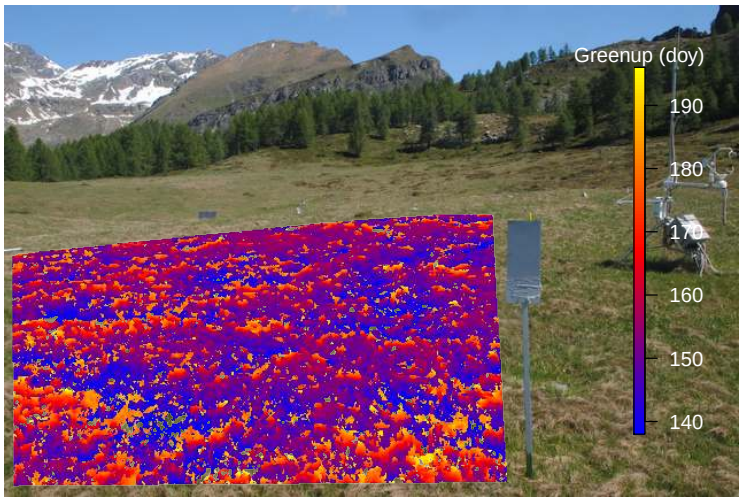

Figure 4: PhenoMap of Greenup phenophase

Note the flexibility to add aditional arguments to customize the legend. For any additional argument see the help file of `fields::image.plot` upon which the code for the legend is built.

Let's get an overview of all phases extracted in the data.frame `global.phenophases`.

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> par(mfrow=c(2,2))
> plotSpatial(global.phenophases, param='Greenup', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=colors(100),
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='Greenup', col='white'), legend.mar=7)
> plotSpatial(global.phenophases, param='Maturity', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=colors(100),
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='Maturity', col='white'), legend.mar=7)
> plotSpatial(global.phenophases, param='Senescence', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=colors(100),
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='Senescence', col='white'), legend.mar=7)
> plotSpatial(global.phenophases, param='Dormancy', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=colors(100),
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='Dormancy', col='white'), legend.mar=7)
```

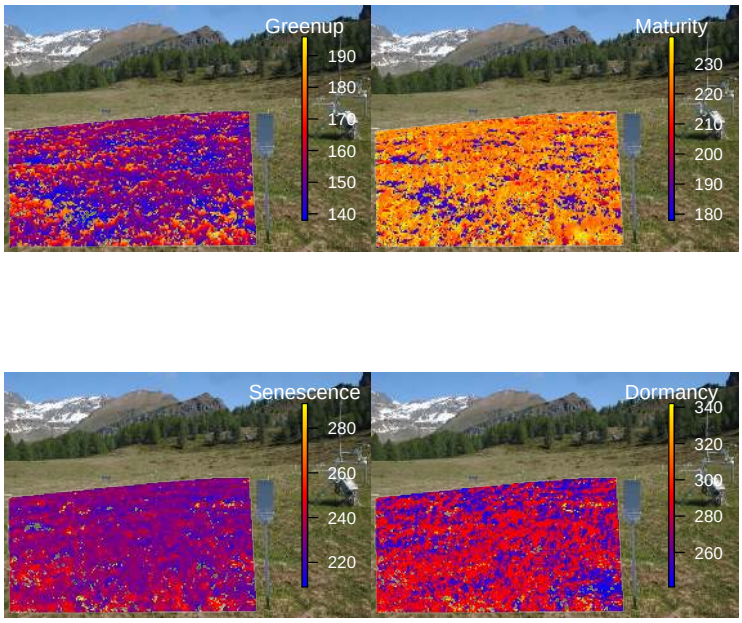

Figure 5: PhenoMap of phenophases extracted with Klosterman method

In fig.5 note the adjustment of legend margin (arg `legend.mar`, which defaults to 5.1) to have the labels properly included in the plotting region.

One may want to compute th length of the growing season as defined by the difference in days between Greenup and Dormancy. This can be done by simply:

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> global.phenophases$gsl <- global.phenophases$Dormancy - global.phenophases$Greenup
> plotSpatial(global.phenophases, param='gsl',
+         roi.data.path='files/roi.data.Rdata',
+         image.path='files/20130630T1000.jpg',
+         col=colors(100),
+         axis.args=list(col.axis='white'),
+         legend.args=list(text='Growing season length \n(days)', col='white'))
```

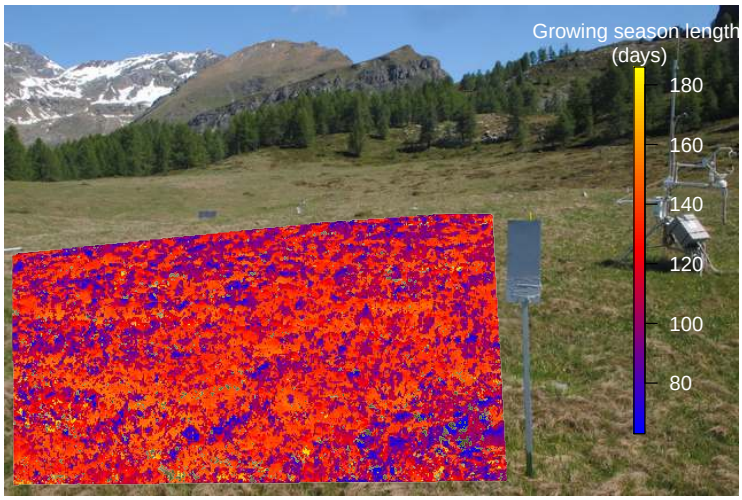

Figure 6: PhenoMap of growing season length (days)

Additionally, one may be interested in checking what is the impact of different curve fitting (spline vs klosterman methods) on the final computation of a given phenophase. In our case we can compare `global.phenophases` and `global.phenophases.klosterman` as in the following example:

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> greenup.consistency <- data.frame(
+         difference=global.phenophases$Greenup-global.phenophases.klosterman$Greenup
+         )
> plotSpatial(greenup.consistency, param='difference',
+         roi.data.path='files/roi.data.Rdata',
+         image.path='files/20130630T1000.jpg',
+         col=colors(100),
+         axis.args=list(col.axis='white'),
+         legend.args=list(text='Difference (days)', col='white'))
```

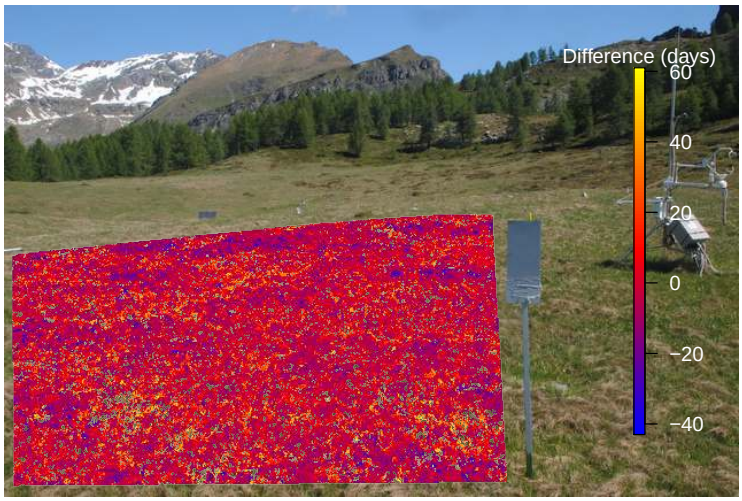

Figure 7: Comparison between two different methods (spline vs Klosterman) for phenophase extraction

There is no evidence of spatially distributed bias between the two methods, however the overall range of the difference is +60 -40, which is quite a lot. But if we look at the summary of the differences:

```
> summary(greenup.consistency)

   difference
 Min.   :-85.0000
```

```
 1st Qu.: -9.0000
 Median : -4.0000
 Mean   :  0.1383
 3rd Qu.:  6.0000
 Max.   :102.0000
 NA's   :2631
```

we see that 50% of pixels show a difference no higher than 10 days.

To complete, let's see the distribution of Maturity according to klosterman fitting method. We will use it later on.

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> na.pc <- length(
+         which(
+                 is.na(
+                         global.phenophases.klosterman$Maturity)))/length(
+ global.phenophases.klosterman$Maturity)*100
> round(na.pc)

[1] 3

> plotSpatial(global.phenophases.klosterman, param='Maturity',
+         roi.data.path='files/roi.data.Rdata',
+         image.path='files/20130630T1000.jpg',
+         col=colors(100),
+         axis.args=list(col.axis='white'),
+         legend.args=list(text='Greenup (doy)', col='white'))
```

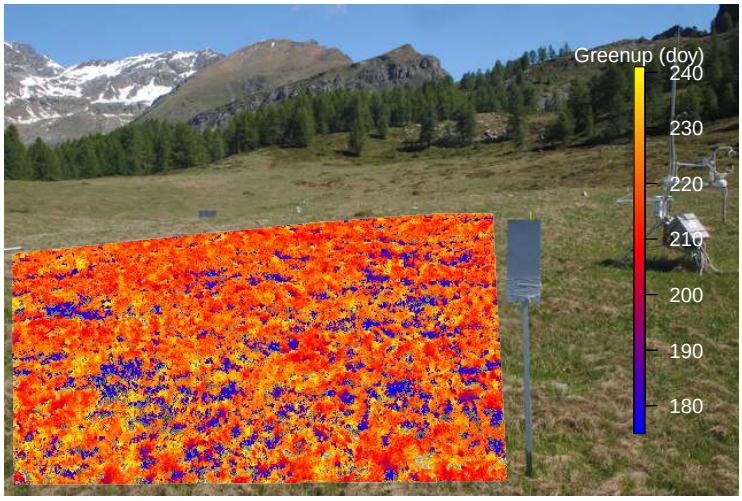

Figure 8: PhenoMap of Maturity phenophase after Klosterman curve fitting

Note that we also calculated the percentage of unfitted pixels with the klosterman fitting appraach, which is 3% in this case. It is normal to have

a certain number of pixels which cannot be fitted by double logistic functions. Actually, 3% is a small number that can be much greater with more noisy signals or different ecosystems. So, the suggestion is: in case of many unfitted pixels use the spline appoach, much faster and with no evidence of bias compared to double logistic fittings.

12 Pushing forward the analysis: cluster analysis

The phenophase maps provide useful information to evaluate spatial patterns of phenology within an image scene. The next step that we propose is to simplify the signal of the phenophase maps by grouping pixels according to the extracted phenophases. The function `greenClusters` allows to perform scaled k-means clustering on an input data.frame, that can be either the complete phenopase dataset or a selection of them. Here is the usage of the function:

```
> args(greenClusters)

function (data.clusters, data.curve = NULL, nclusters, formula = NULL,
    plot = TRUE)
NULL
```

Argument `data.clusters` must be a data.frame containing one or more variables (normally phenophases) upon which the clustering will be performed. `data.curve` is an optional data.frame of curve parameters that, if provided, is used to generate average curves for each of the cluster. We can plot results of the cluster analysis as we did with the phenophase maps. For this example, we will use as input matrix phenophases Greenup and Maturity since they show the more interesting spatial patterns. Here is the the code to perform a cluster analysis fixing the number of clusters to 4:

```
> cluster.data <- greenClusters(global.phenophases[,1:2], nclusters=4)
> str(cluster.data)

List of 3
 $ curves  : logi NA
 $ napos   : int(0)
 $ clusters:'data.frame':        93599 obs. of  1 variable:
  ..$ clusters: int [1:93599] 3 3 3 2 2 2 1 1 1 1 ...
```

If we look at the structure of the object `cluster.data` it is a list of three elements. The first `curves` is empty for the moment. `napos` is empty because there are no NA in the `global.phenophases` data.frame. `clusters` is the data.frame with a single column containing an integer vector 1:nclusters long npixels which assigns each pixel to a cluster. And this is what we will plot:

```
> colors <- palette()[1:4]
> plotSpatial(cluster.data$clusters, param='clusters',
+         roi.data.path='files/roi.data.Rdata',
+         image.path='files/20130630T1000.jpg',
+         col=colors,
+         axis.args=list(col.axis='white'),
+         legend.args=list(text='Clusters', col='white'))
```

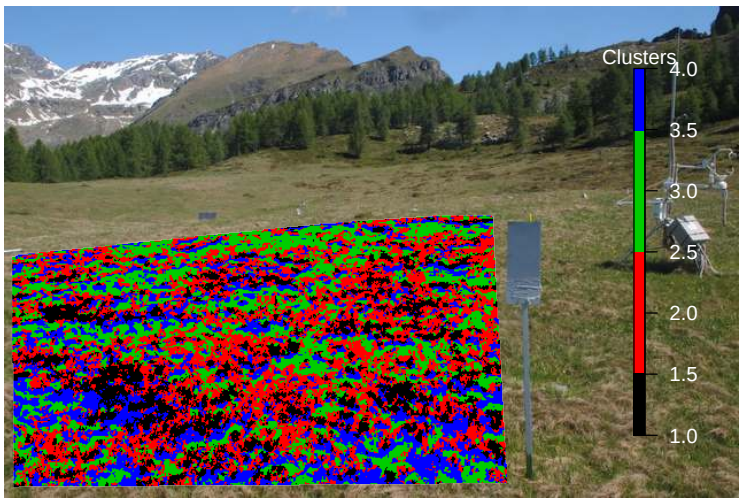

Figure 9: An example of cluster analysis

How can we evaluate if the number of clusters is enough, or is excessive? A number of functions can do taht in R. However to my knowledge none of them can handle a huge dataset as those in output from the spatial analysis. Therefore what we can suggest is to explore the distribution of pixels on different clusters, as below:

```
> round(table(cluster.data$clusters)/length(cluster.data$clusters$clusters)*100)

 1  2  3  4
19 33 26 22

> cluster.data3 <- greenClusters(global.phenophases[,1:2], nclusters=3)
> round(table(cluster.data3$clusters)/length(cluster.data$clusters$clusters)*100)

 1  2  3
24 19 57

> cluster.data2 <- greenClusters(global.phenophases[,1:2], nclusters=2)
> round(table(cluster.data2$clusters)/length(cluster.data$clusters$clusters)*100)

 1  2
25 75

> cluster.data5 <- greenClusters(global.phenophases[,1:2], nclusters=5)
> round(table(cluster.data5$clusters)/length(cluster.data$clusters$clusters)*100)

 1  2  3  4  5
31 22 22  7 16
```

With a high number of clusters, one cluster group may be strongly underrepresented compared to other clusters, an indication that one cluster might be removed. For example with 5 clusters, only three clusters are highly represented, suggesting that 5 clusters are too much. Let's look at those clusters plotted.

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> par(mfrow=c(2,2))
> plotSpatial(cluster.data5$clusters, param='clusters', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=palette()[1:5],
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='5 clusters', col='white'), legend.mar=7)
> plotSpatial(cluster.data$clusters, param='clusters', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=palette()[1:4],
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='4 clusters', col='white'), legend.mar=7)
> plotSpatial(cluster.data3$clusters, param='clusters', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=palette()[1:3],
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='3 clusters', col='white'), legend.mar=7)
> plotSpatial(cluster.data2$clusters, param='clusters', 'files/roi.data.Rdata',
+         'files/20130630T1000.jpg', col=palette()[1:2],
+         axis.args=list(col.axis='white', cex=.8),
+         legend.args=list(text='2 clusters', col='white'), legend.mar=7)
```

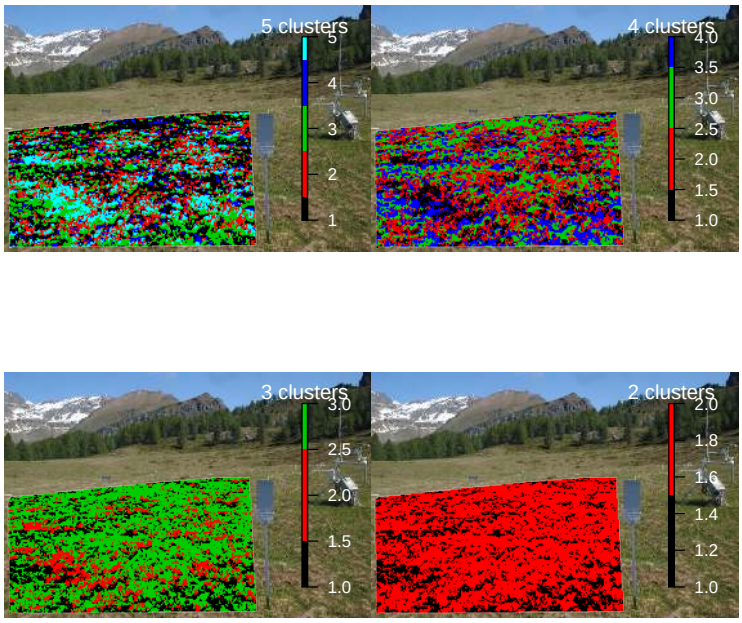

Figure 10: Cluster analysis with different number of clusters

The visual assessment confirms that 5 clusters may be too much. On the other side, with two clusters there is a large loss of information. We will try to go on with 3 clusters. An interesting question would be to look at an "average behavior" of each of those clusters. This can be done by exploiting other arguments of the function `greenClusters`. In particular if we supply the `data.curve` argument, average curve parameters are computed for each cluster and a fitted trajectory for them is extracted. To extract fitted trajectories from a set of parameters we must first tell R which formula was used to fit the original curve. We therefore source a processed time series stored as an example in the package (`bartlett2009.processed`) which was fitted with the klosterman method, as in our case. From this fit we retrieve the `formula`, which must eb given as third argument to `greenClusters`. Here is the code:

```
> data(bartlett2009.processed)
> klosterman.formula <- bartlett2009.processed$fit$fit$formula
> cluster.data.params <- greenClusters(global.phenophases[,1:2],
+         data.curve=global.phenophases.klosterman[,5:17],
+         nclusters=3, formula=klosterman.formula,
+         plot=FALSE)

> str(cluster.data.params)

List of 3
 $ curves  :âĂŸzooâĂŹ series from 1 to 365
  Data: num [1:365, 1:3] 0.323 0.323 0.322 0.322 0.322 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "cluster1" "cluster2" "cluster3"
  Index:  int [1:365] 1 2 3 4 5 6 7 8 9 10 ...
 $ napos   : int(0)
 $ clusters:'data.frame':        93599 obs. of  1 variable:
  ..$ clusters: int [1:93599] 3 3 3 3 3 3 2 2 2 2 ...
```

Note the difference in the structure of `cluster.data.params` compared to the previous `cluster.data`. Now we have the element `curves` filled with a zoo series, the average seasonal trajectory of gcc for each of the clusters. They can be of course plotted:

```
> colors <- palette()[1:3]
> plot(cluster.data.params$curves, plot.type='single', col=colors)
```

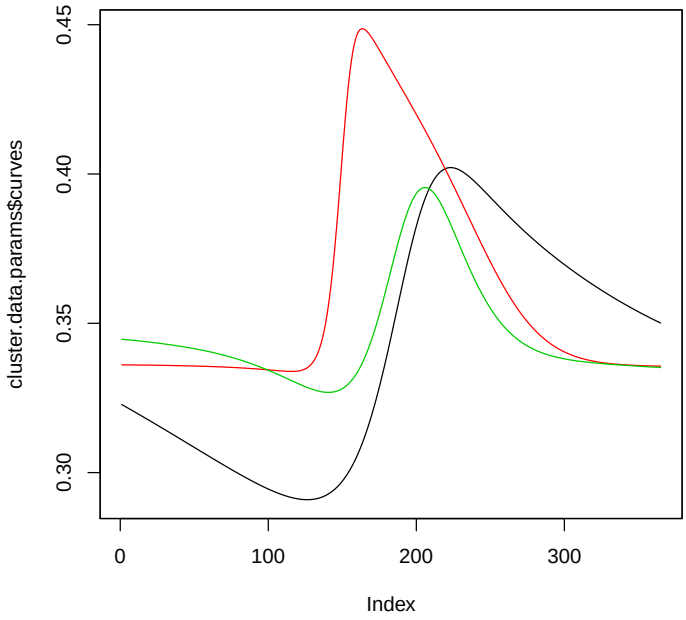

Figure 11: Average curves for each of the 3 clusters analysed

Looking at this time series one can argue that the three clusters have strong different phenology. That's good news. However making an average of all curve parameters belonging to the same cluster is a very rough method (tought very fast) to evaluate the differences between clusters. A more formal approach would be to use the four cluster groups to subset pixels right after the VI filtering, and process them as if they would belong to the same region of interest. Easier to show it than to explain it. First we must remeber that our filtered data are still splitted over the 10 subROIS. First thing is then to group them with the following:

```
> filtered.list <- list()
> for (a in 1:length(filtered.files)) {
+         act.filtered.path <- vi.paths[a]
+         act.number.big <- substr(act.filtered.path,
+                 nchar(act.filtered.path)-10,
+                 nchar(act.filtered.path)) ## this line requires that you don't have numb
```

```
+     ## in last ten characters of your folder path
+         track.number <- as.numeric(gsub("[^0-9]", "", act.number.big), "")
+         load(paste0(act.filtered.path, '/filtered.tmp.Rdata'), verbose=TRUE)
+         filtered.list[[track.number]] <- filtered.tmp
+ }
> library(zoo)
> global.filtered.data <- zoo(order.by=index(filtered.list[[1]]))
> for (a in 1:10) {
+         tmp.df <- filtered.list[[a]]
+         global.filtered.data <- cbind(global.filtered.data, tmp.df)
+ print(a)
+ }
```

Now, `global.filtered.data` is a huge zoo series occupying 140Mb in my folder, which dimensions are:

```
> dim(global.filtered.data)

[1]   261 93599
```

which means we have a time index and one column for each processed pixel. It is fairly unsafe to load and handle such a large object in R, but however it seems to work. Now we simply have to use the vector of clusters contained in `cluster.data.params$clusters` to subset and average out our large zoo series. One way to do that is the following:

```
> cl1.pos <- which(cluster.data.params$clusters$clusters==1)
> cl2.pos <- which(cluster.data.params$clusters$clusters==2)
> cl3.pos <- which(cluster.data.params$clusters$clusters==3)
> avg.cl1 <- apply(global.filtered.data[,cl1.pos], 1, 'mean')
> avg.cl2 <- apply(global.filtered.data[,cl2.pos], 1, 'mean')
> avg.cl3 <- apply(global.filtered.data[,cl3.pos], 1, 'mean')
> avg.clusters <- zoo(cbind(avg.cl1, avg.cl2, avg.cl3),
+         order.by=index(global.filtered.data))
```

Now let's plot them:

```
> colors <- palette()[1:3]
> plot(avg.clusters, plot.type='single', col=colors)
> legend('topright', col=colors, legend=paste('cluster', 1:3), lty=1, bty='n')
```

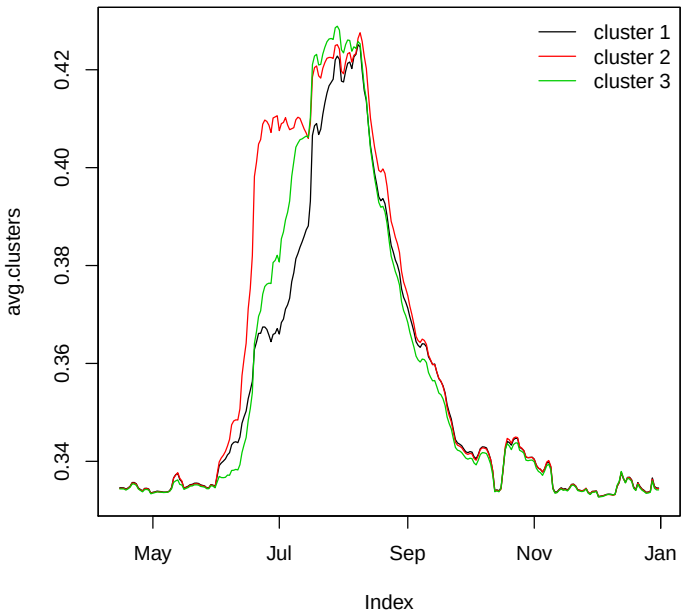

Figure 12: TRUE Average curves for each of the 3 clusters analysed

And then fit them as if they were extracted from the ROI-averaged approach (see the base vignette for explainations)

```
> fit1 <- greenProcess(avg.clusters[,1], 'spline', 'gu', plot=FALSE)
> fit2 <- greenProcess(avg.clusters[,2], 'spline', 'gu', plot=FALSE)
> fit3 <- greenProcess(avg.clusters[,3], 'spline', 'gu', plot=FALSE)
> par(mfrow=c(3,1), oma=c(5,4,4,2), mar=rep(.5,4))
> plot(fitted(fit1))
> abline(v=extract(fit1, 'metrics')[1:4])
> plot(fitted(fit2))
> abline(v=extract(fit2, 'metrics')[1:4])
> plot(fitted(fit3))
> abline(v=extract(fit3, 'metrics')[1:4])
```

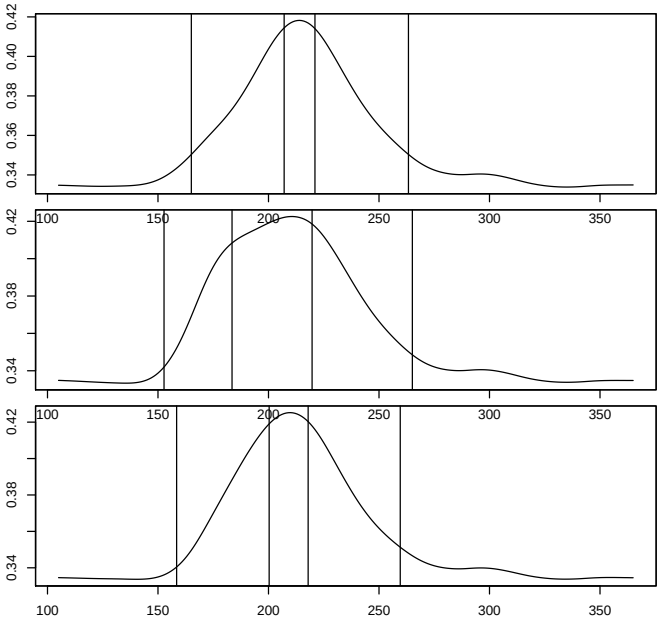

Figure 13: Fitted curves and extracted thresholds in the three ROIs defined with the cluster analysis, from top to bottom, cluster 1, 2 and 3

The more formal approach shows markedly different trajectories, considering that we are looking at a 10square meters area in an alpine homogeneous grassland! The differences shown here are tightly coupled to the species distribution on the ROI, with a mosaic of early and late developing species occurring at a very short distance.

We lastly show a further analysis conducted on a subalpine larch stand shown in the following figure.

```
> PrintROI('files/20121004T1400.jpg', 'files/data2load/')
```

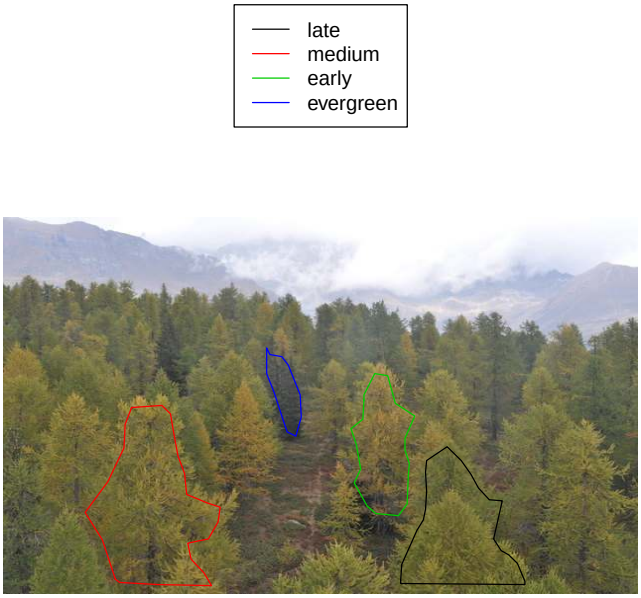

Figure 14: ROIs extracted at the Torgnon Larch site

At this site 4 rois were analysed separately, one evergreen tree which will not be show later and three groups of larch trees showing a markedly different autumn phenology. These rois were spatially processed separately and then merged together in a named list as follows:

```
> all.spatial <- list(early=early.spatial, late=late.spatial,
+ medium=medium.spatial)
```

This list of spatial phenophases can be plotted in the same figure with `plotSpatial` as follows:

```
> colors <- colorRampPalette(c('blue', 'red', 'yellow'), space='rgb')
> plotSpatial(all.spatial, param='DD',
+         roi.data.path='files/data2load/roi.data.Rdata',
+         image.path='files/20121004T1400.jpg',
+         col=colors(100),
+         axis.args=list(col.axis='blue'),
+         legend.args=list(text='Downturn date (doy)', col='blue'),
+         probs=c(0.05, 0.99))
```

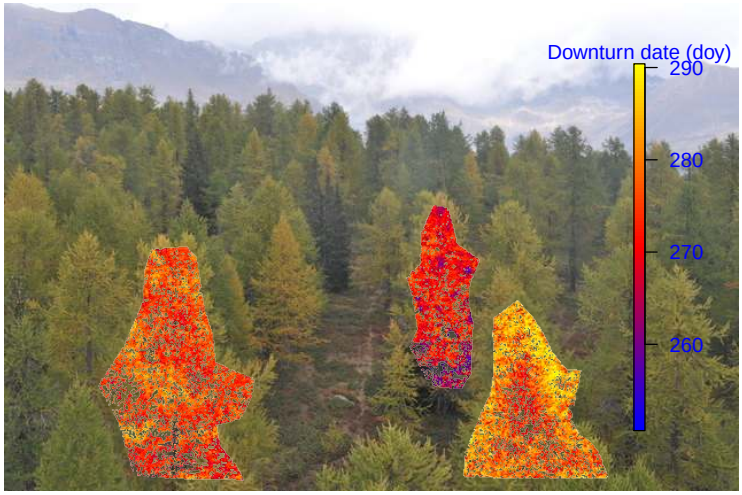

Figure 15: PhenoMap of Downturn date (beginning of senescence) in the Torgnon larch stand

Spatially explicit phenophases show different patterns of autumn phenology across the selected individuals of larch trees.

13 Summary

In this vignette we showed how to perform a spatially explicit pixel-based analysis using the `phenopix` R package. First, the region of interest is splitted to ensure the creation of objects of an appropriate size that R can handle. Second,

data are filtered, fitted and phenophases are extracted for each pixel. Then, phenophases maps are produced and overplotted to a given reference image. Fourth, cluster analysis can be conducted on the extracted phenophases or curve parameters and pixel data can be re-processed according to the clustering.

The R package `phenopix` is available at the R forge site and directily within R by running the command:

```
> install.packages("phenopix", repos="http://R-Forge.R-project.org")
```

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

email: `gian.filippa@gmail.com`

14 References

14.1 General background to digital image analysis

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.

14.2 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.

View publication stats