

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

1,156

1 author:



Gianluca Filippa

Environmental Protection Agency of Aosta Valley

103 PUBLICATIONS 2,792 CITATIONS

SEE PROFILE

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



Unexplored drivers of plant phenology: Beyond climate [View project](#)



EGU 2020 Latest Developments and Software Tools for Ecosystem and Flux Data Analysis [View project](#)

Phenopix: pixel based phenology

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

September 3, 2020

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. Throughout 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 ($\geq 2.15.3$) and imports one or more functions from the following packages:

`zoo`, `plyr`, `SDMTools`, `jpeg`, `stringr` ($\geq 1.0.0$), `bcp`, `strucchange`, `parallel`, `foreach`, `doParallel`, `iterators`, `gtools`, `raster`, `sp`

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 18.04.4 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## 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 knitr_1.25
##
## loaded via a namespace (and not attached):
## [1] Rcpp_1.0.4.6      lattice_0.20-38    codetools_0.2-16
## [4] gtools_3.8.1      zoo_1.8-7          foreach_1.5.0
## [7] plyr_1.8.6        grid_3.6.3         magrittr_1.5
## [10] evaluate_0.14     stringi_1.4.6      sp_1.4-2
## [13] raster_3.1-5      doParallel_1.0.15  strucchange_1.5-2
## [16] sandwich_2.5-1    iterators_1.0.12   tools_3.6.3
## [19] stringr_1.4.0     jpeg_0.1-8.1       bcp_4.0.3
## [22] xfun_0.13         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 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 available in Publication - Methods of my research gate page.

Since throughout this vignette I often refer to the base vignette, I 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 differences 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 on CRAN and can be installed via the following command:

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

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 view of R objects, this implies the build 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

```
library(raster)

## Loading required package:  sp
##
## Attaching package:  'raster'
## The following object is masked from 'package:phenopix':
##
##      extract

library(phenopix)
setwd('/home/gian/sweave/SPATIAL_VIGNETTE/')
sample.file <- brick('files/REF/20130630T1000.jpg')
dim(sample.file)

## [1] 214 320    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']
print(fg.roi)
```

```
## $fg
## $fg$mask
## class      : RasterLayer
## dimensions : 214, 320, 68480 (nrow, ncol, ncell)
## resolution : 1, 1 (x, y)
## extent     : 0, 320, 0, 214 (xmin, xmax, ymin, ymax)
## crs        : NA
## source     : memory
## names      : X20130630T1000.1
## values     : 0, 1 (min, max)
##
##
## $fg$polygons
## class      : SpatialPolygons
## features    : 2
## extent     : 10.9687, 310.462, 12.2525, 106.284 (xmin, xmax, ymin, ymax)
## crs        : NA
```

This is a list with two elements, the first being a binary raster layer (a mask), and the second a SpatialPolygons object with vertices of roi polygons.

What matters in our context is the number of pixels included in the ROIs, and for which we want to perform the analysis. Here is how to assess it

```
freq(fg.roi$fg$mask)

##      value count
## [1,]      0 45786
## [2,]      1 22694
```

Pixels falling within the ROI are labeled 1. Hence, now we know that our ROI consists of more than 22k 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 I recommend to split the ROI into smaller subROIs of no more than 5-10k pixels as follows:

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

## [1] "roi1" "roi2" "roi3" "roi4" "roi5" "roi6" "roi7" "roi8"
## [9] "roi9" "roi10"
```

```
sapply(splitted.roi, function(x) length(which(values(x$mask)==1)))

## roi1 roi2 roi3 roi4 roi5 roi6 roi7 roi8 roi9 roi10
## 2269 2269 2269 2269 2269 2269 2269 2269 2269 2273
```

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/')
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,'/'), showWarnings=FALSE)
  dir.create(paste0('files/VI/VI',a,'/'), showWarnings=FALSE)
  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 explanation 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 your ROI is divided into.

```
img.path <- '/home/gian/sweave/IMG'
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',
             ncores=7,
             log.file='/home/gian/')
}

```

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]]))

## torgnon-nd_2018_01_01_100000.jpg torgnon-nd_2018_01_01_110000.jpg
##           "2018-01-01 10:00:00"           "2018-01-01 11:00:00"
## torgnon-nd_2018_01_01_120000.jpg torgnon-nd_2018_01_01_130000.jpg
##           "2018-01-01 12:00:00"           "2018-01-01 13:00:00"
## torgnon-nd_2018_01_01_140000.jpg torgnon-nd_2018_01_01_150000.jpg
##           "2018-01-01 14:00:00"           "2018-01-01 15:00:00"

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

## 'data.frame': 2269 obs. of  3 variables:
##  $ red   : num  146 145 144 144 144 144 143 142 140 140 ...
##  $ green: num  146 145 144 144 144 144 143 142 140 140 ...
##  $ blue  : num  148 147 146 146 146 146 145 144 142 142 ...

```


- Within each element of this list is another list with one element for each time step processed (note the output of `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.

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` 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:

```
str(spatialFilter)

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

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 parallel 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/'`) where 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 some 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/', ncores=6)  
  save.name <- paste0(act.vi.path, '/filtered.tmp.Rdata')  
  save(filtered.tmp, file=save.name)  
}
```

With this for loop for each subROI I load the VI extracted, filter them out and 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 include 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.      :2018-01-01 00:00:00   Min.      :0.3331  
##   1st Qu.:2018-05-15 00:00:00   1st Qu.:0.3348  
##   Median :2018-07-29 00:00:00   Median :0.3391  
##   Mean    :2018-07-29 08:46:40   Mean     :0.3636
```

##	3rd Qu.:	2018-10-16 00:00:00	3rd Qu.:	0.3951
##	Max.	:2018-12-31 00:00:00	Max.	:0.4578

```

sampling.vect <- sample(dim(filtered.tmp)[2], 500)
library(zoo)

##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##   as.Date, as.Date.numeric

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)

```

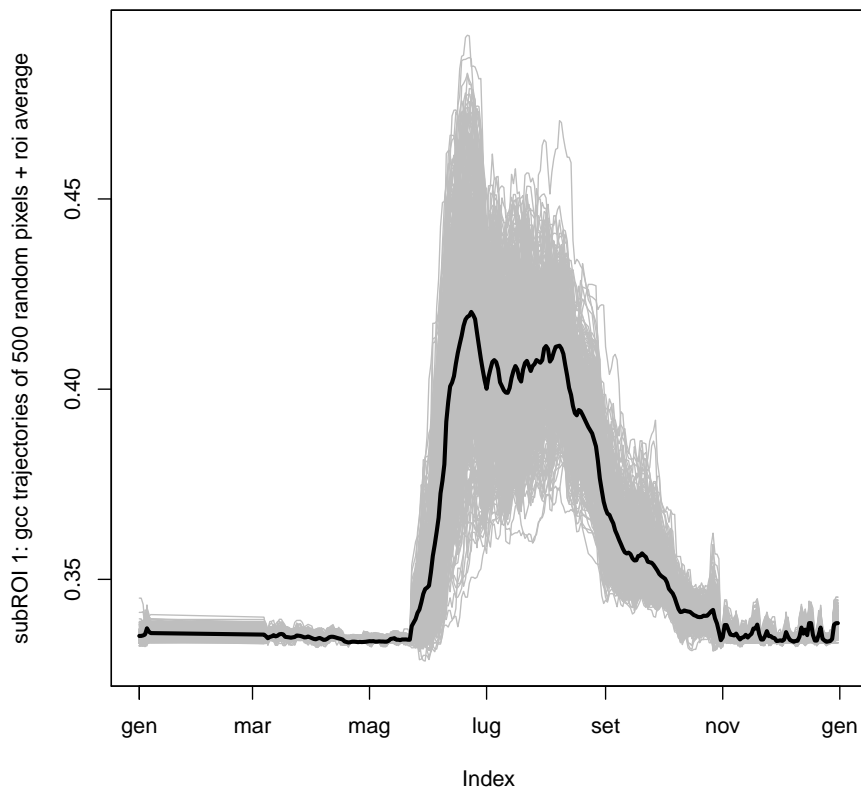


Figure 1: The seasonal trajectory of gcc of 500 random pixels from subROI 1

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 remember 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 I illustrate the usage of the function that performs fittings pixel by pixel, i.e. `spatialGreen()`. Here is the usage:

```
str(spatialGreen)

## function (filtered.data, fit, threshold, ncores = "all", log.file = 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, pattern='filtered')
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 `phenopix::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)

## Loading objects:
##   fitted.tmp

## extraction of a matrix (because of sapply)
fitted.values <- sapply(fitted.tmp, phenopix::extract, what='fitted')
## convert to a zoo object
fitted.values2 <- zoo(fitted.values,
                      order.by=index(phenopix::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')
```

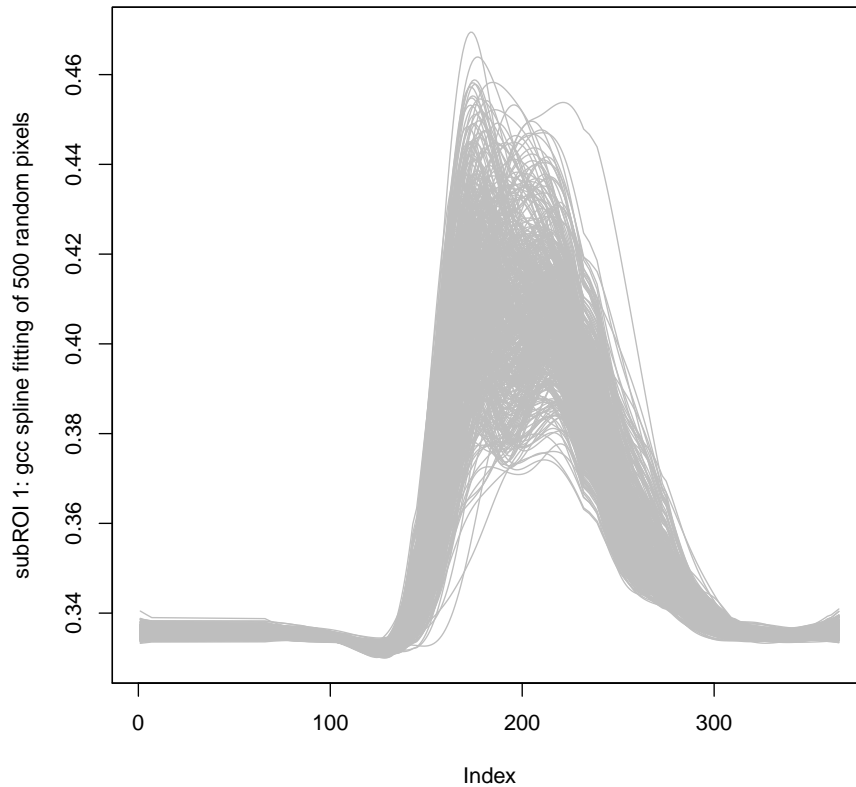


Figure 2: The seasonal trajectory of fitted gcc of 500 random pixels from subROI 1

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:

```
str(extractParameters)

## function (list, update = 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.klosterman` 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 the number of pixels analysed.

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

##	Greenup	Maturity	Senescence	Dormancy
##	Min. :126	Min. :170.0	Min. :188.0	Min. :201.0
##	1st Qu.:133	1st Qu.:178.0	1st Qu.:205.0	1st Qu.:259.0
##	Median :135	Median :180.0	Median :213.0	Median :260.0
##	Mean :138	Mean :188.5	Mean :217.7	Mean :261.6
##	3rd Qu.:139	3rd Qu.:187.0	3rd Qu.:231.0	3rd Qu.:262.0
##	Max. :204	Max. :245.0	Max. :263.0	Max. :365.0
##	RMSE			
##	Min. :0.002497			
##	1st Qu.:0.004154			
##	Median :0.004798			
##	Mean :0.004894			
##	3rd Qu.:0.005536			
##	Max. :0.008488			

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.trrs <- extractParameters(fitted.tmp, update='trrs')
spatial.phenophases.trrs2 <- extractParameters(fitted.tmp,
  update='trrs', trrs=0.25)
```

And their comparison:

```
plot(spatial.phenophases.trrs$sos,
  spatial.phenophases.trrs2$sos,
  pch=20, xlim=c(140, 210),
  ylim=c(140, 210),
  xlab='start of season as day when 50% GCC is reached',
  ylab='start of season as day when 25% GCC is reached')
```

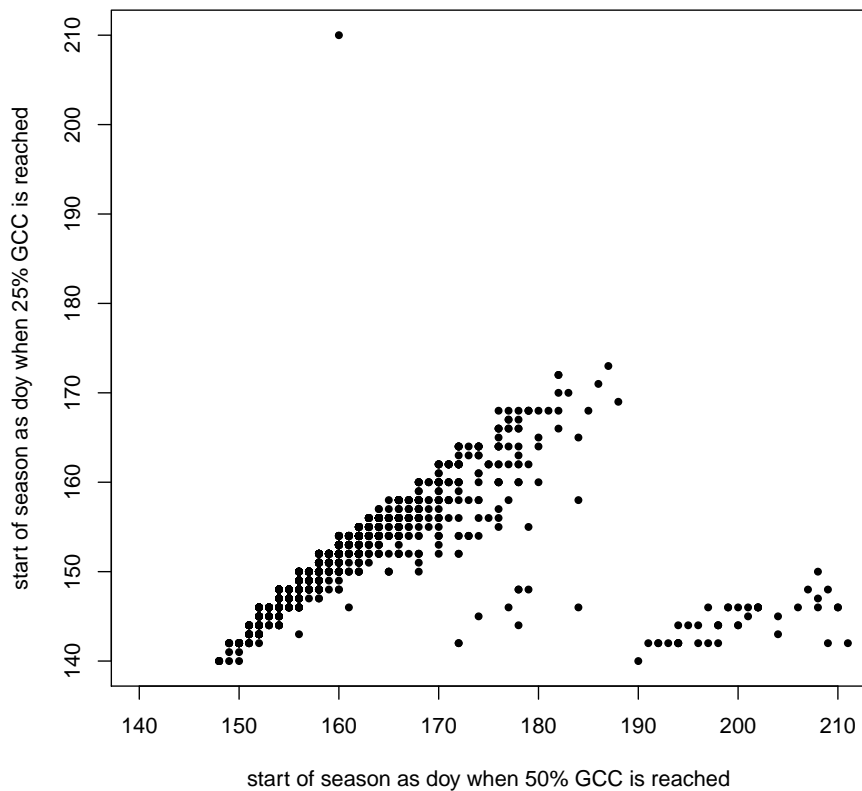


Figure 3: Scatter comparing phenophase trrs methods with two different thresholds

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 together 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.tmp <- extractParameters(fitted.tmp)
  global.phenophases.list[[track.number]] <- spatial.phenophases.tmp
}

global.phenophases <- NULL
for (a in 1:10) {
  tmp.data.frame <- global.phenophases.list[[a]]
  global.phenophases <- rbind(global.phenophases, tmp.data.frame)
}
```

It must be paid special attention on how the various subROI objects are binded together. This is because the file listing 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 ordered list (`global.phenophases.list`) and then unlist it and bind objects to get the final `global.phenophases` dataset. That being said, let's look at the `global.phenophases` data.frame:

```
names(global.phenophases)
## [1] "Greenup"      "Maturity"     "Senescence"   "Dormancy"     "RMSE"
```

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

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

`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/REF/20130630T1000.jpg',
  col=colors(100),
  axis.args=list(col.axis='white'),
  legend.args=list(text='Greenup (doy)', col='white'),
  horizontal=TRUE,
  smallplot=c(0.5, 0.7, 0.78, 0.8))

```

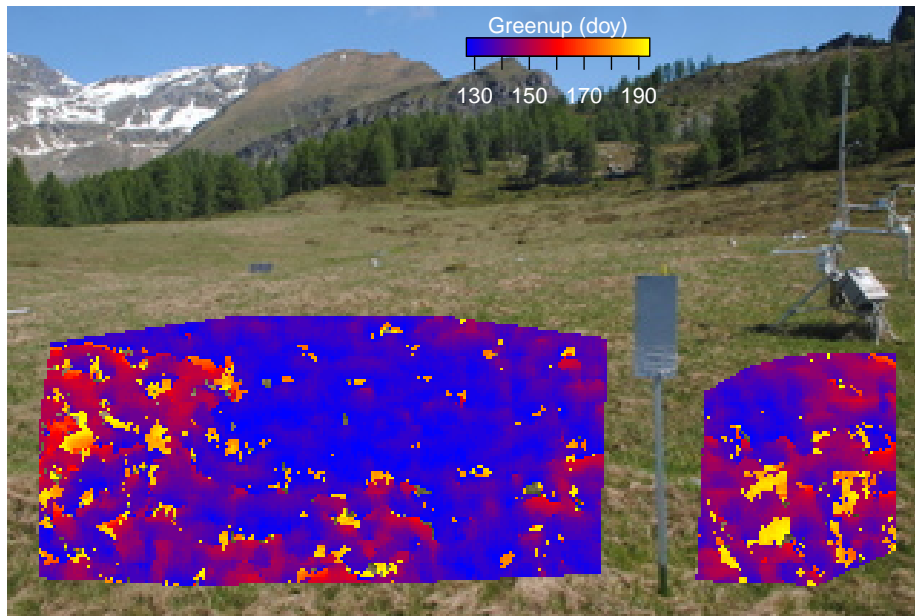


Figure 4: PhenoMap of Greenup phenophase

Note the flexibility to add additional 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(2,1))
plotSpatial(global.phenophases, param='Greenup', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=colors(100),
  axis.args=list(col.axis='white', cex=.8),
  legend.args=list(text='Greenup', col='white'), legend.mar=7,
  horizontal=TRUE,
  smallplot=c(0.5, 0.8, 0.76, 0.78))

plotSpatial(global.phenophases, param='Maturity', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=colors(100),
  axis.args=list(col.axis='white', cex=.8),
  legend.args=list(text='Maturity', col='white'), legend.mar=7,
  horizontal=TRUE,
  smallplot=c(0.5, 0.8, 0.76, 0.78))

```

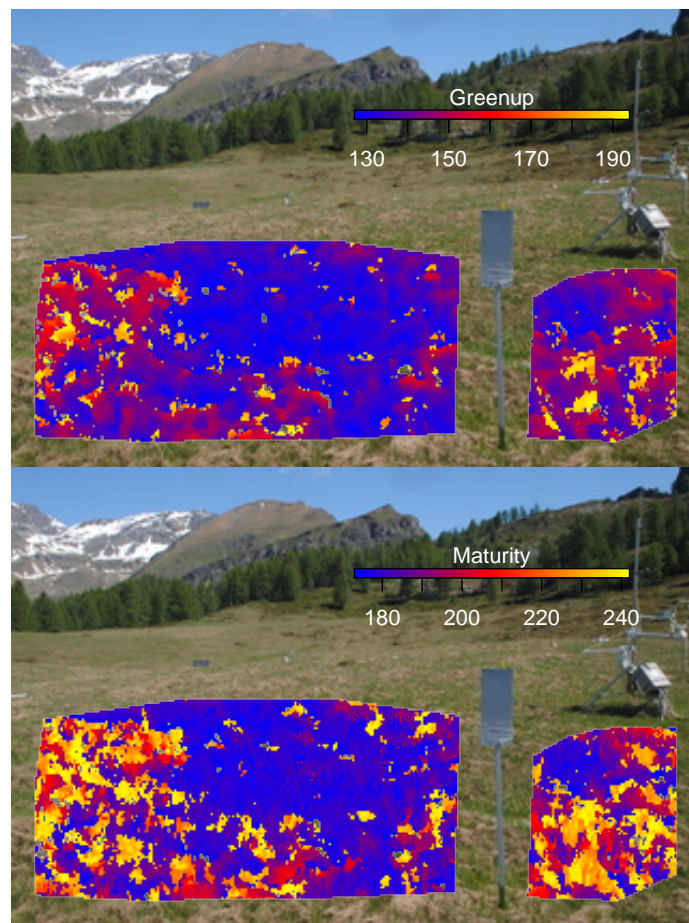


Figure 5: PhenoMap of phenophases extracted with Klosterman method

```

par(mfrow=c(2,1))
plotSpatial(global.phenophases, param='Senescence', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=colors(100),
  axis.args=list(col.axis='white', cex=.8),
  legend.args=list(text='Senescence', col='white'), legend.mar=7,
  horizontal=TRUE,
  smallplot=c(0.5, 0.8, 0.76, 0.78))

plotSpatial(global.phenophases, param='Dormancy', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=colors(100),
  axis.args=list(col.axis='white', cex=.8),
  legend.args=list(text='Dormancy', col='white'), legend.mar=7,
  horizontal=TRUE,
  smallplot=c(0.5, 0.8, 0.76, 0.78))

```

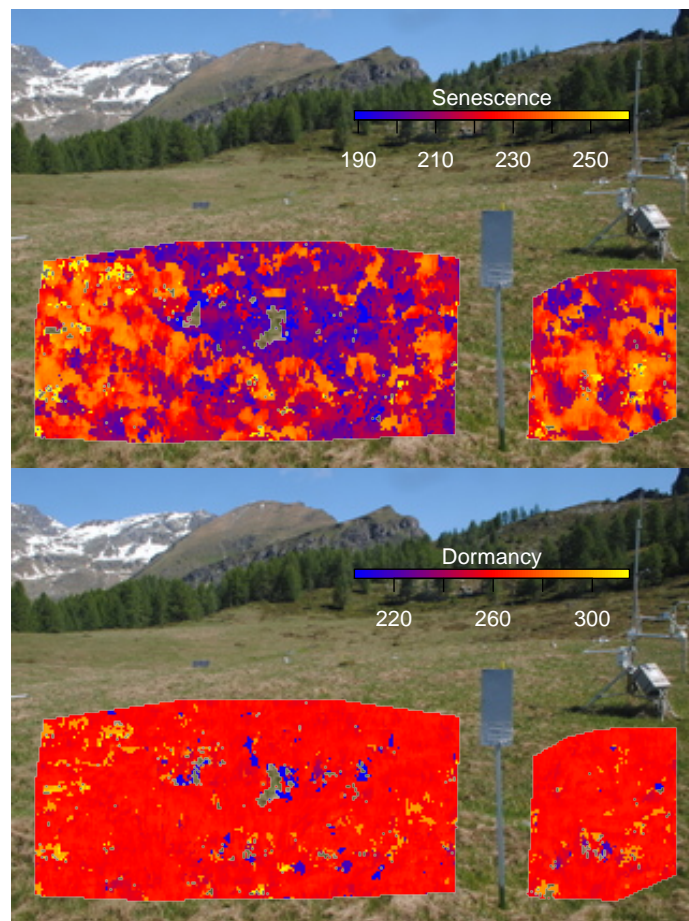


Figure 6: PhenoMap of phenophases extracted with Klosterman method (cont.)

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 the 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/REF/20130630T1000.jpg',
  col=colors(100),
  axis.args=list(col.axis='white'),
  legend.args=list(text='Growing season length \n(days)', col='white'),
  horizontal=TRUE,
  smallplot=c(0.5, 0.9, 0.76, 0.78))

```

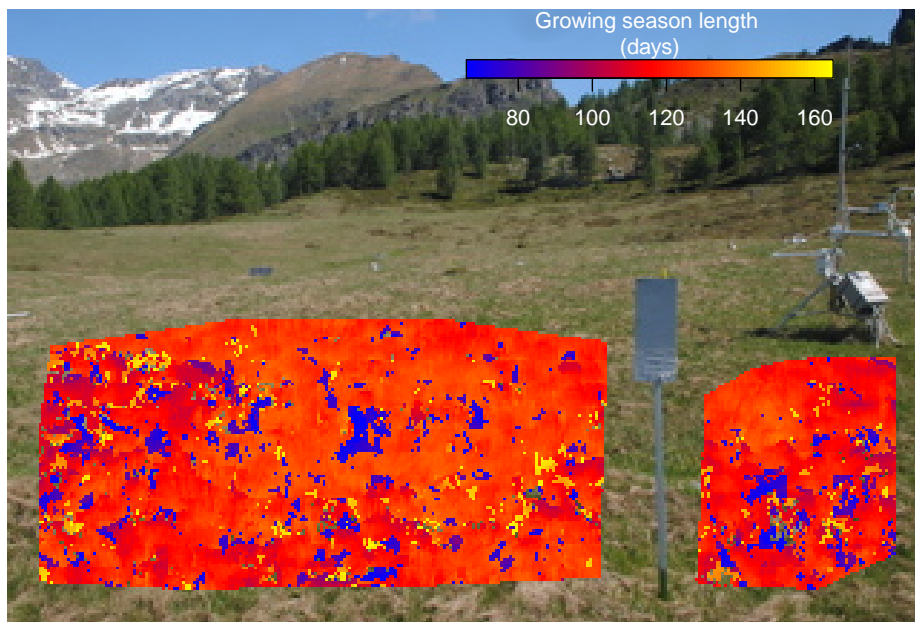


Figure 7: PhenoMap of growing season length (days)

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 I 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 phenophase dataset or a selection of them. Here is the usage of the function:

```
str(greenClusters)

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

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 I did with the phenophase maps. For this example, I will use as input matrix phenophases `Greenup` and `Maturity` since they show the more interesting spatial patterns. Here is 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 : NULL
## $ clusters:'data.frame': 22694 obs. of 1 variable:
## ..$ clusters: int [1:22694] 4 4 4 4 4 1 4 4 4 4 ...
```

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/REF/20130630T1000.jpg',
  col=colors,
  axis.args=list(col.axis='white'),
  legend.args=list(text='Clusters', col='white'),
  horizontal=TRUE,
  smallplot=c(0.5, 0.9, 0.76, 0.78))

```

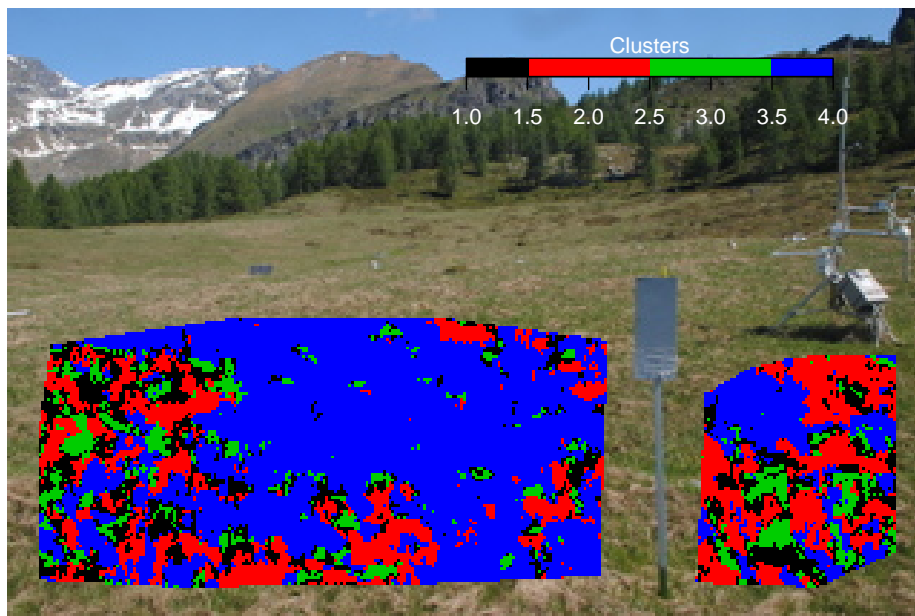


Figure 8: An example of cluster analysis

How can we evaluate if the number of clusters is enough, or is excessive? A number of functions can do that in R, even though some of them struggle with large datasets. Here I propose a more naive approach, by observing the distribution of pixels on different clusters, as below:

```

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

##
##  1  2  3  4
## 16 21  9 54

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

##
##  1  2  3
## 18 74  9

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

##
##  1  2
## 25 75

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

##
##  1  2  3  4  5
## 13 39 16  9 24

```

With a high number of clusters, one cluster group may be strongly under-represented compared to other clusters, an indication that one cluster might be removed. For example with 5 clusters, only 2 of them 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$clusters, param='clusters', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=palette()[1:5],
  axis.args=list(col.axis='white', cex=.8), legend=FALSE)
plotSpatial(cluster.data$clusters, param='clusters', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=palette()[1:4],
  axis.args=list(col.axis='white', cex=.8), legend=FALSE)
plotSpatial(cluster.data3$clusters, param='clusters', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=palette()[1:3],
  axis.args=list(col.axis='white', cex=.8),
  legend=FALSE)

plotSpatial(cluster.data2$clusters, param='clusters', 'files/roi.data.Rdata',
  'files/REF/20130630T1000.jpg', col=palette()[1:2],
  axis.args=list(col.axis='white', cex=.8),
  legend=FALSE)

```

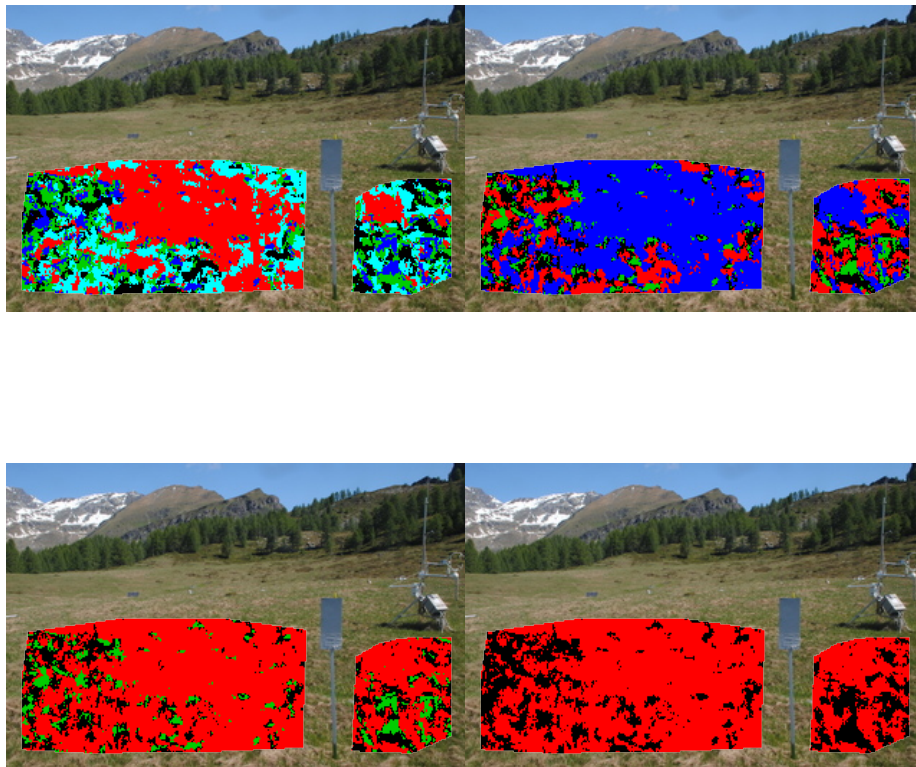


Figure 9: Cluster analysis with different number of clusters, from topleft, clockwise: 5, 4, 2 and 3 clusters

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 4 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. I therefore source a phenocam time series stored as an example in the package (`bartlett2009.filtered`) which we will fit with the `elmore` method, as in our case. From this fit we retrieve the `formula`, which must be given as third argument to `greenClusters`. Here is the code:

```
data(bartlett2009.filtered)
elmore.fit <- greenProcess(bartlett2009.filtered, fit='elmore')
elmore.formula <- elmore.fit$fit$fit$formula
cluster.data.params <- greenClusters(global.phenophases[,1:2],
                                     data.curve=global.phenophases.elmore[,5:11],
                                     nclusters=4, formula=elmore.formula,
                                     plot=FALSE)

str(cluster.data.params)

## List of 3
## $ curves : 'zoo' series from 1 to 365
## Data: num [1:365, 1:4] 0.0261 0.0261 0.0261 0.0261 0.0261 ...
## .. attr(*, "dimnames")=List of 2
## .. ..$ : NULL
## .. ..$ : chr [1:4] "cluster1" "cluster2" "cluster3" "cluster4"
## Index: int [1:365] 1 2 3 4 5 6 7 8 9 10 ...
## $ napos : NULL
## $ clusters: 'data.frame': 22694 obs. of 1 variable:
## ..$ clusters: int [1:22694] 1 1 1 1 1 2 1 1 1 1 ...
```

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 plotted:

```

colors <- palette()[1:4]
plot(cluster.data.params$curves, plot.type='single', col=colors)

```

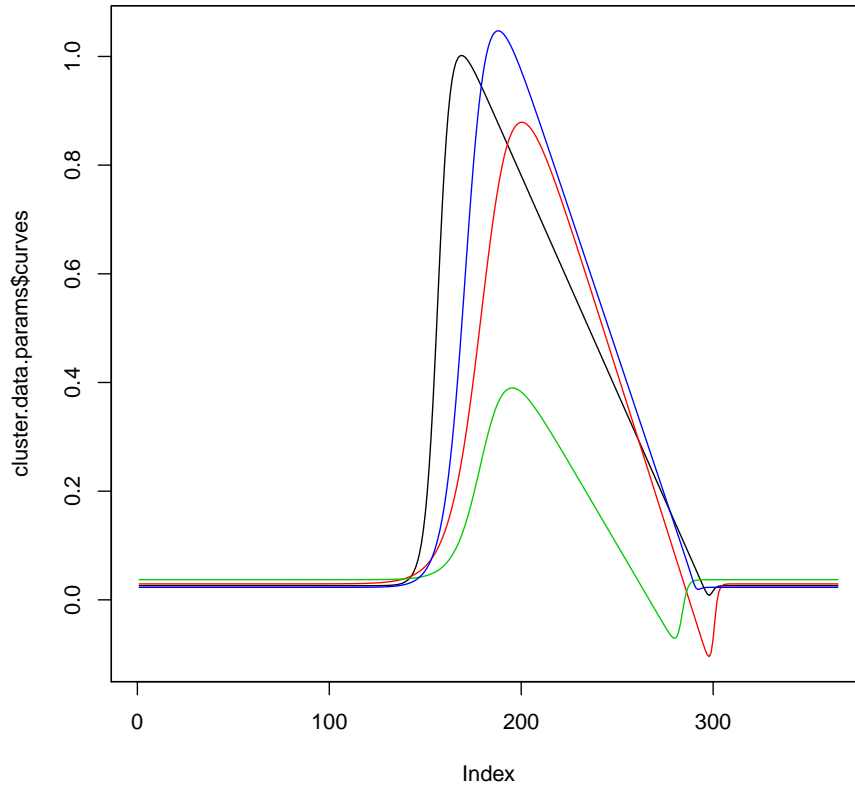


Figure 10: Average curves for each of the 4 clusters analysed

Looking at this time series one can argue that the three clusters have strongly different phenology. That's good news. However making an average of all curve parameters belonging to the same cluster is a very rough method (thought very fast) to evaluate the differences between clusters. This also results in strange curve behaviour at the end of the summer, with negative values. 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 remember that our filtered data are still split 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 number
  ## 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 fairly large zoo series, which dimensions are:

```

dim(global.filtered.data)

## [1] 297 22694

```

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)
cl4.pos <- which(cluster.data.params$clusters$clusters==4)
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.cl4 <- apply(global.filtered.data[,cl4.pos], 1, 'mean')

```

```
avg.clusters <- zoo(cbind(avg.cl1, avg.cl2, avg.cl3, avg.cl4),
  order.by=index(global.filtered.data))
```

Now let's plot them:

```
colors <- palette()[1:4]
plot(avg.clusters, plot.type='single', col=colors)
legend('topright', col=colors, legend=paste('cluster', 1:4), lty=1, bty='n')
```

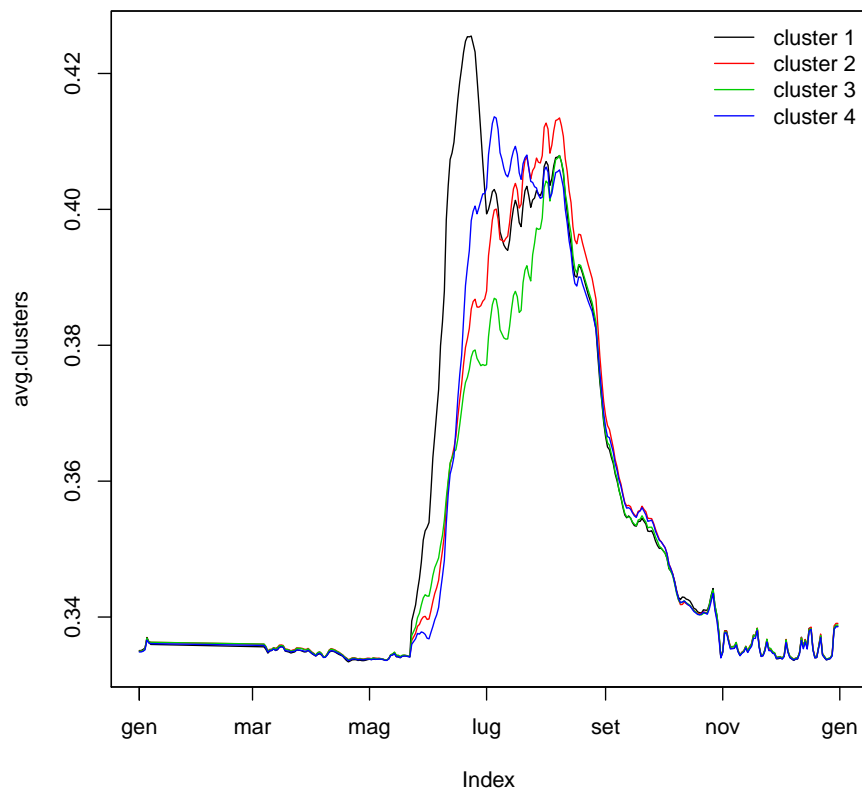


Figure 11: TRUE Average curves for each of the 3 clusters analysed

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


```

fit1 <- greenProcess(avg.clusters[,1], 'gu', 'gu', plot=FALSE)
fit2 <- greenProcess(avg.clusters[,2], 'gu', 'gu', plot=FALSE)
fit3 <- greenProcess(avg.clusters[,3], 'gu', 'gu', plot=FALSE)
fit4 <- greenProcess(avg.clusters[,4], 'gu', 'gu', plot=FALSE)

par(mfrow=c(4,1), oma=c(5,4,4,2), mar=rep(.5,4))
plot(fitted(fit1))
abline(v=phenopix::extract(fit1, 'metrics')[1:4])
points(phenopix::extract(fit1, 'data'), pch=20, col='grey')
plot(fitted(fit2))
abline(v=phenopix::extract(fit2, 'metrics')[1:4])
points(phenopix::extract(fit2, 'data'), pch=20, col='grey')
plot(fitted(fit3))
abline(v=phenopix::extract(fit3, 'metrics')[1:4])
points(phenopix::extract(fit3, 'data'), pch=20, col='grey')
plot(fitted(fit4))
abline(v=phenopix::extract(fit4, 'metrics')[1:4])
points(phenopix::extract(fit4, 'data'), pch=20, col='grey')

```

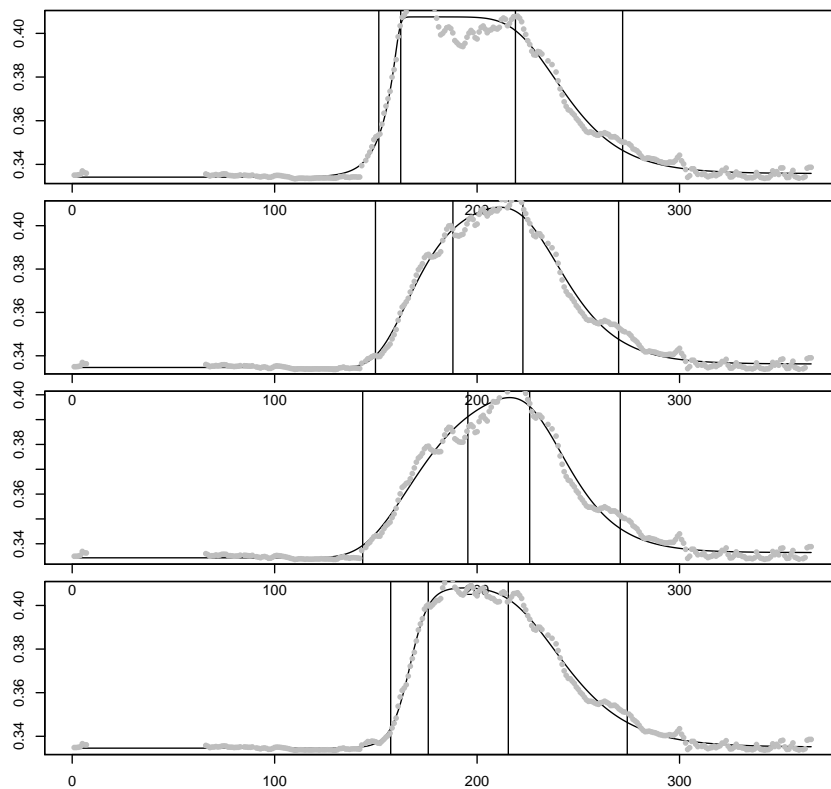


Figure 12: Fitted curves and extracted thresholds in the three ROIs defined with the cluster analysis, from top to bottom, cluster 1, 2, 3 and 4

The more formal approach shows markedly different trajectories especially during the early developing stages. This is not surprising because I used exactly the two spring phases to cluster my data. Considering that we are looking at a 10square meters area in an alpine homogeneous grassland, there is a striking phenodiversity within this area. 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.

13 Summary

In this vignette I showed how to perform a spatially explicit pixel-based analysis using the **phenopix** R package. First, the region of interest is split 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 on CRAN and directly within R by running the command:

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

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

Filippa, G.et al.Phenopix: A R package for image-based vegetation phenology.Agric. For. Meteorol220,141150 (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; Metsmki, Sari; Aurela, Mika; Bttcher, 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 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.

14.3 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.