Some agricultural monitoring exercises with R [D4P1b]

Gregory Duveiller

September 17, 2015

Introduction

This document outlines the steps that will be taken in the practicals of agricultural monitoring of the ESA land training course 2015 (D4P1b). The objectives of this session are to familiarize students with the R programming language (https://www.r-project.org/) and show them how it is possible to make most (if not all) processing steps in a remote sensing analysis using only R. The advantages of R include its strength as a comprehensive statistical and graphical tool developed by statisticians and researchers, which is free and open source software, with no license restrictions, it is cross-platform and have a large community of users.

In this practical, we will learn to open, combine and analyse time series of remote sensing images. The exercise is focused on processing steps that could be relevant for research and developement in the scope of agricultural monitoring with remote sensing. We will combine spatio-temporal datasets of 3 different sources with 3 different spatial resolutions. To do so we will be using the open source RStudio software, an integrated development environment (IDE) for R https://www.rstudio.com/.

The first step is to open RStudio. In the console, we can type commands that are executed by R. As an example, type these lines:

```
a <- 2+2
b <- 5
a * b
```

```
## [1] 20
```

```
x <- c(2,a,b)
y <- seq(1,5,2)
x + y
```

```
## [1]  3  7 10
```

R works by calling packages, which can be installed directly from the console and later loaded to the current R session. One that will be heavily used in this practical is the `raster` package. The following lines show how install it:

```
install.packages('raster')
```

and load it:

```
require(raster)
```

```
## Loading required package: raster
## Loading required package: sp
```

The Data

The data needed for this practicals are all in a given place on your machine. Set up the path as follows (but changing the path accordingly):

```
wpath <- '/home/duveigr/Teaching/ESA_LandTrainingCourse_2015/TP/'
```

The data has already been slightly pre-processed to facilitate this course. They consist in:

High resolution NDVI images

These are derived from imagery captured by the SPOT4/HRVIR satellite and distributed under the SPOT4/Take5 initiative (https://spot-take5.org/) provided by CNES and CESBIO. The idea of SPOT4/Take5, and of its successor SPOT5/Take5, is to produce time series with decametric spatial resolution over selected zones, somewhat simulating what Sentinel2 will provide. The images here are extracts of NDVI calculated form the atmospherically and geometrically corrected bands provided in SPOT4/Take5. The area is in southern France during the 2013 growing season and have a spatial resolution of 20m.

Low resolution NDVI images

These consist of daily NDVI at ~250m spatial resolution calculated from the MODIS imagery acquired both from the Terra and the Aqua platforms. Data can be downloaded from the NASA LPDAAC (https://lpdaac.usgs.gov/dataset_discovery/modis/) but can also be managed using a dedicated R package called `MODIS` (see https://r-forge.r-project.org/R/?group_id=1252 for more info).

Temperature data

This dataset consists of 3 years of daily mean temperature over Europe at a 0.31x0.31 degrees lat/lon grid. It is obtained from the MARS site of the European Commission Joint Research Centre (JRC) at this site: http://agri4cast.jrc.ec.europa.eu/DataPortal/.

Opening and exploring imagery

We can start opening some images as follows:

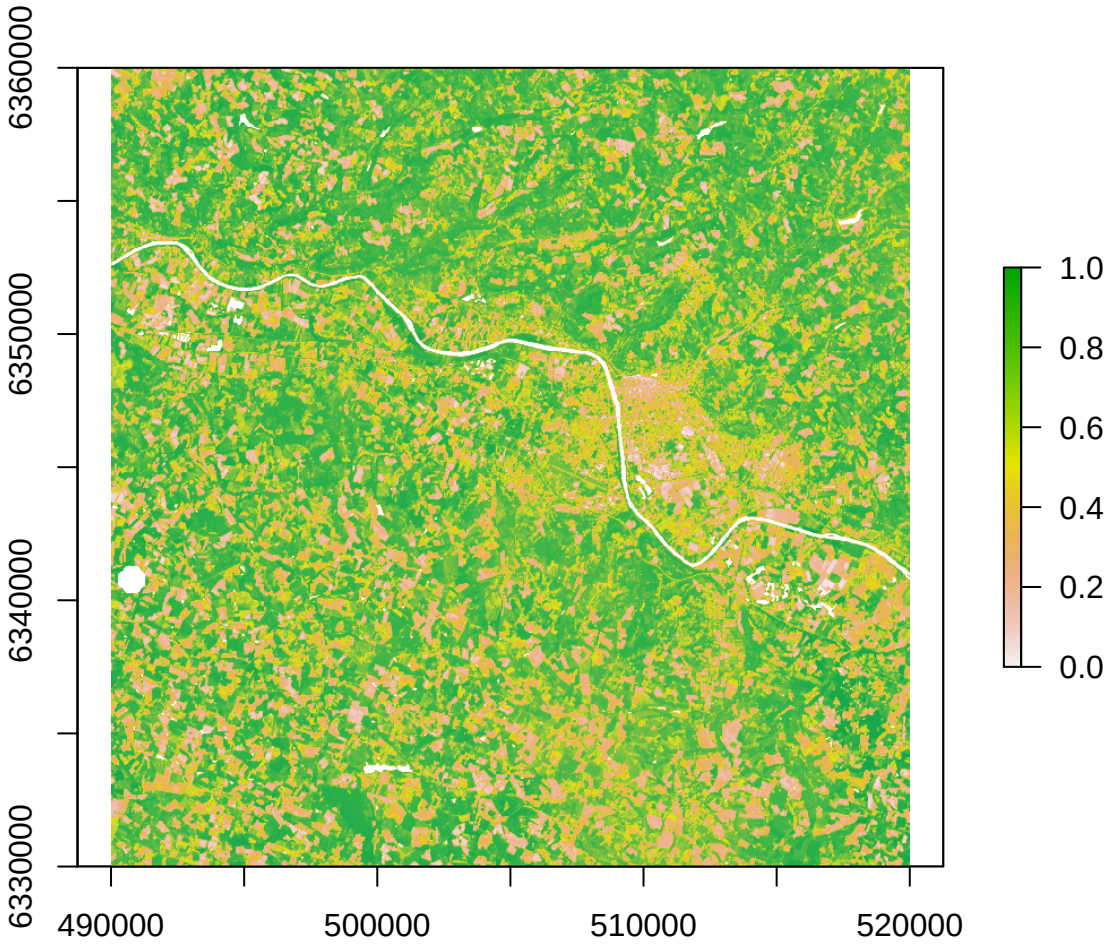
```
# open and explore data
dHR <- '20130616' # this is the date code of one of the images
rHR <- raster(paste0(wpath,'DATA/HiRes_ndvi/','SPOT4_HRVIR_XS_',dHR,'_N2A_NDVI_CSudmipy.tif'))
```

Typing the name of the variable `rHR` in R gives information on the raster:

```
## class       : RasterLayer
## dimensions  : 1500, 1500, 2250000  (nrow, ncol, ncell)
## resolution  : 20, 20  (x, y)
## extent      : 490000, 520000, 6330000, 6360000  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5 +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS8
## data source : /home/duveigr/Teaching/ESA_LandTrainingCourse_2015/TP/DATA/HiRes_ndvi/SPOT4_HRVIR_XS_2(
## names       : SPOT4_HRVIR_XS_20130616_N2A_NDVI_CSudmipy
## values      : -1, 1  (min, max)
```

The raster can also be visualized using the `plot` function

```
# plot it
plot(rHR,zlim=c(0,1))
```



Open the equivalent image for MODIS along with information of its view zenith angle:

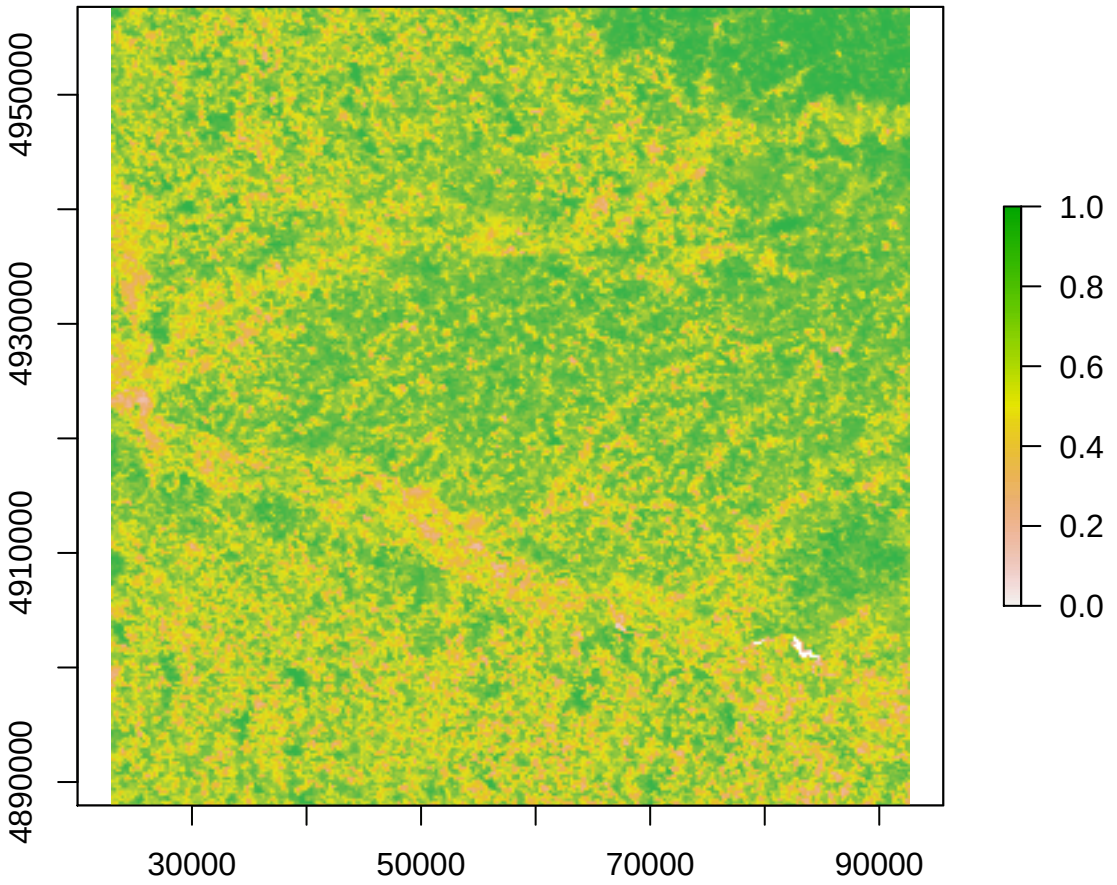
```
# get corresponding MODIS image...
dLR <- format(as.Date(dHR,'%Y%m%d'),'%Y%j')
rLR <- raster(paste0(wpath,'DATA/LoRes_ndvi/','TERRA_MODIS_',dLR,'_NDVI_CSudmipy.tif'))
# get the VZA data and plot it
rLRvza <- raster(paste0(wpath,'DATA/LoRes_ndvi/','TERRA_MODIS_',dLR,'_VZA_CSudmipy.tif'))
```

By looking at the raster metadata, you will notice the projection and extent are different from the high spatial resolution imagery:

```
## class       : RasterLayer
## dimensions  : 301, 301, 90601  (nrow, ncol, ncell)
## resolution  : 231.6564, 231.6564  (x, y)
## extent      : 22933.98, 92662.54, 4887949, 4957678  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=sinu +lon_0=0 +x_0=0 +y_0=0 +a=6371007.181 +b=6371007.181 +units=m +no_defs
## data source : /home/duveigr/Teaching/ESA_LandTrainingCourse_2015/TP/DATA/LoRes_ndvi/TERRA_MODIS_2013
## names       : TERRA_MODIS_2013167_NDVI_CSudmipy
## values      : -0.3222449, 0.8980682  (min, max)
```

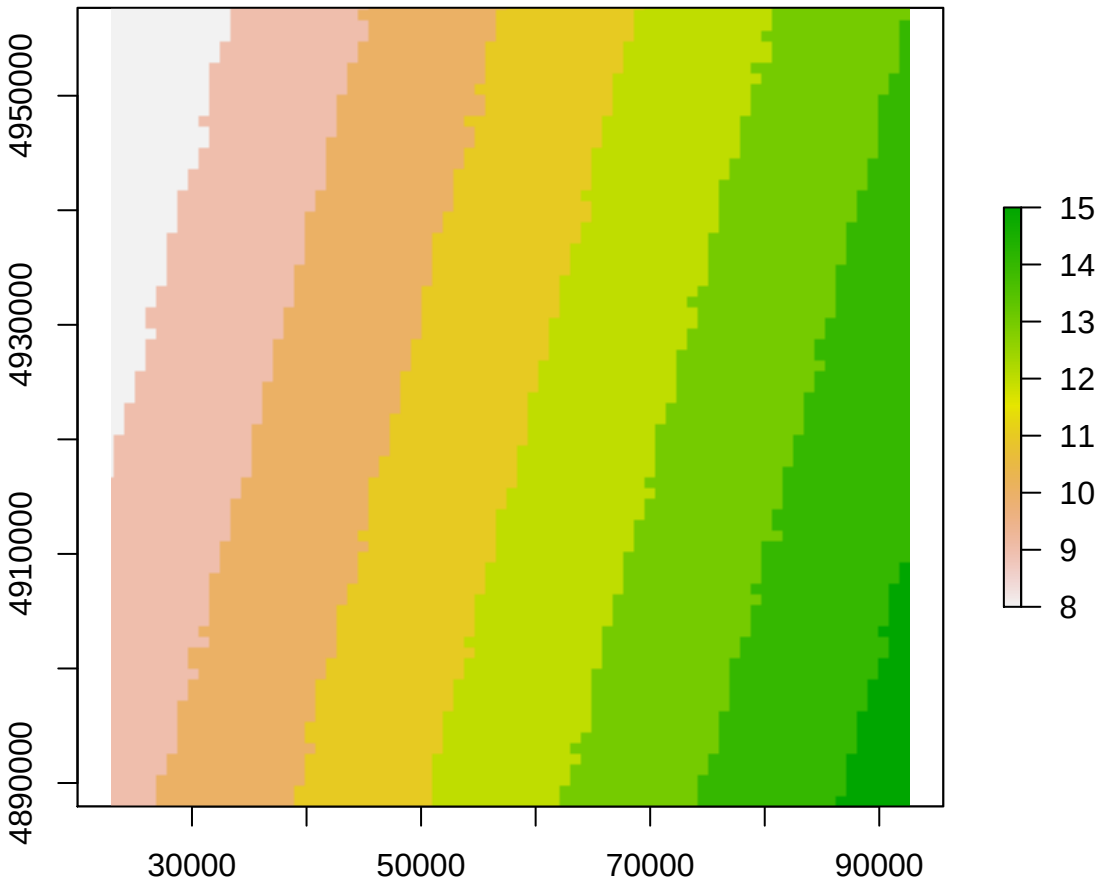
So you shouln't be surprised when you see a considerable difference when you plot the image:

```
plot(rLR,zlim=c(0,1))
```



The MODIS NDVI images are also accompanied by information on the viez zenith angle (VZA) which you can also visualize:

```
plot(rLRvza)
```



Now try explore the data for the images of `dHR <- '20130606'`.

Reprojecting imagery to a common grid

The high spatial resolution imagery is not in the same projection as the low spatial resolution imagery. We will try to bring these together by reprojecting the fine resolution into the coarse one. Doing so, we want to ensure that we also **nest** the fine resolution into the coarse, that is, we want coarse pixels to be composed of an integer number of fine resolution pixels and have both grids exactly aligned.

We will first define a new (empty) raster with the desired extent and resolution.

```
# reproject the extent of the HR image to that of the LR image
rHR.ext.prj <- projectExtent(rHR,crs=crs(rLR))

# align it so that is falls in the LR image
rHR.ext.prj.al <- alignExtent(rHR.ext.prj, rLR, snap='in')

# create a 'dummy' raster with a resolution 10 times finer than that of LR
dum <- raster(rHR.ext.prj.al,crs=crs(rLR), res=res(rLR)/10)
```

By typing `dum` we can see the properties of this new raster. Note how the dimensions, the CRS code, spatial resolution and extent are different.

```
## class       : RasterLayer
```

```
## dimensions  : 1320, 1330, 1755600  (nrow, ncol, ncell)
## resolution  : 23.16564, 23.16564  (x, y)
## extent      : 29420.36, 60230.65, 4896984, 4927562  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=sinu +lon_0=0 +x_0=0 +y_0=0 +a=6371007.181 +b=6371007.181 +units=m +no_defs
```

Now we can project the raster (i.e with all the data in it) to the desired output grid.

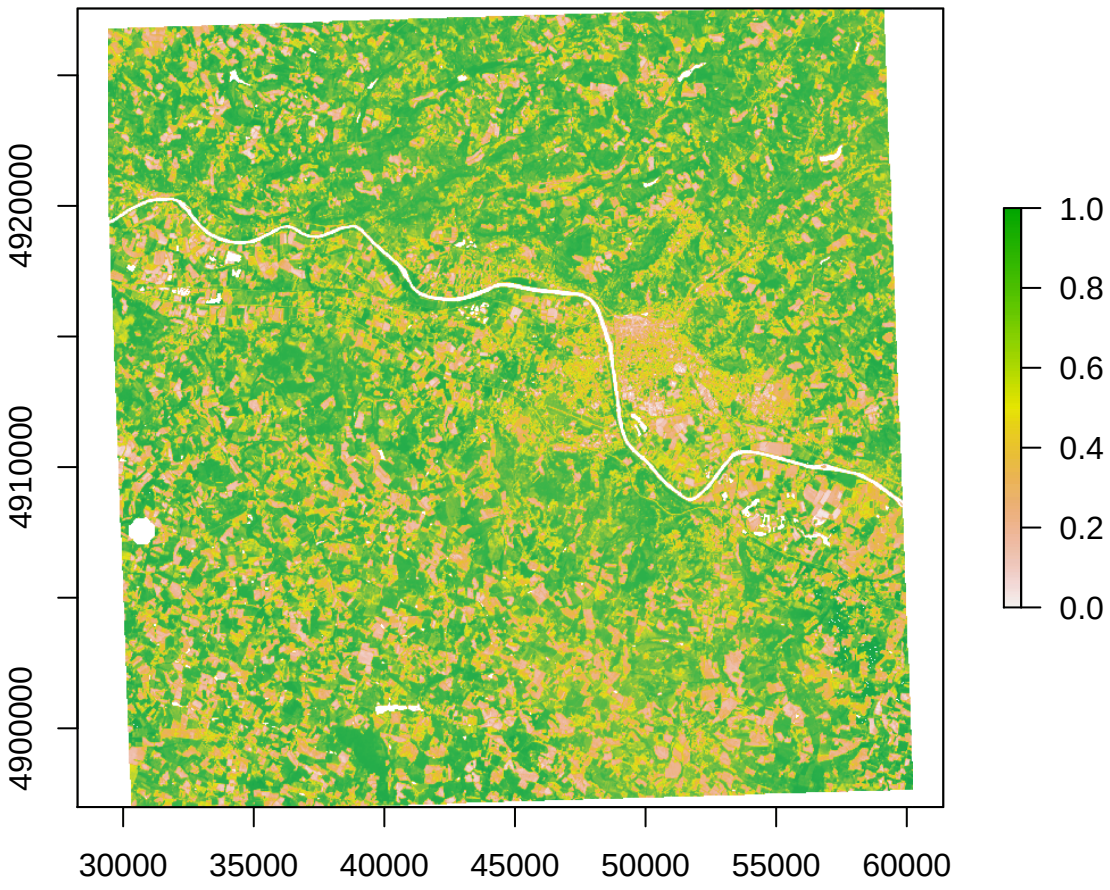
```
rHR.prj <- projectRaster(from = rHR, to=dum)
```

Note that the coarse resolution raster should also be cropped to remove border effects.

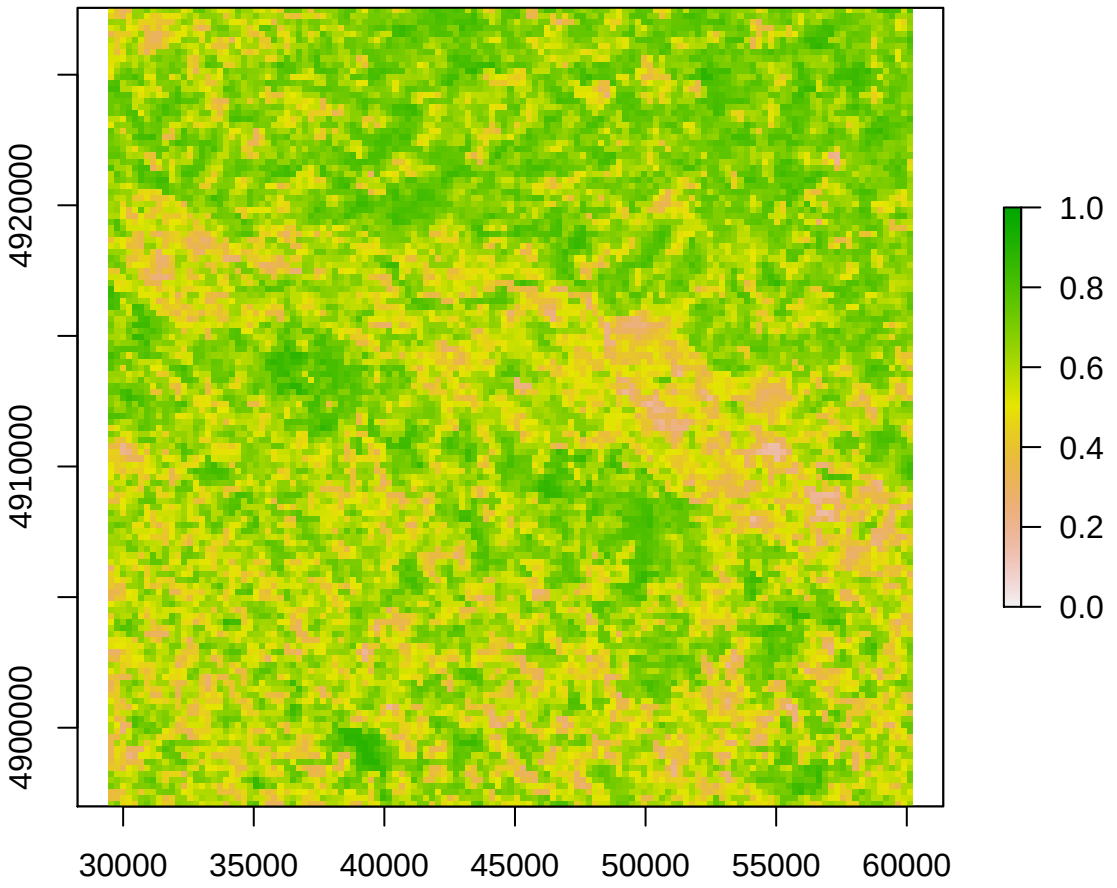
```
rLR.crp <- crop(rLR,extent(rHR.prj))
```

The results can be plotted for visual inspection.

```
plot(rHR.prj,zlim=c(0,1))
```

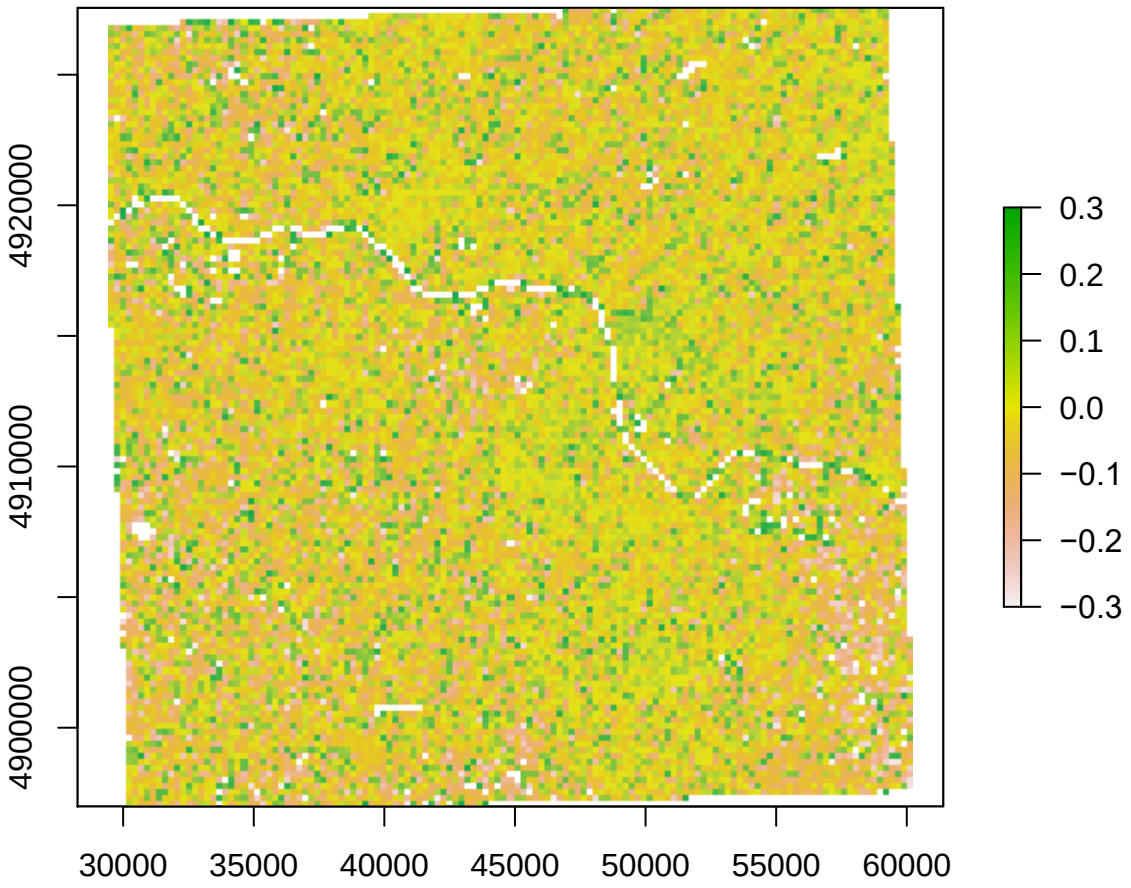



```
plot(rLR.crp,zlim=c(0,1))
```



We now have 2 raster covering the same area, in the same projection and with small pixels nested into the larger ones. We can now aggregate the `HR` to the resolution of `LR` and see the differences in values at every pixel.

```
rHR.agg <- aggregate(rHR.prj,fact=10)
plot(rLR.crp-rHR.agg,zlim=c(-0.3,0.3))
```



We can summarize these steps in a single `function` in R which we can then call to apply it on other images. The function could look like this:

```
project.HiRes <- function(iIMG){
  outname <- paste0(wpath,'DATA/HiRes_ndvi_prj/',basename(iIMG))
  rHR <- raster(iIMG)
  rHR.ext.prj <-projectExtent(rHR,crs=crs(rLR))
  rHR.ext.prj.al <- alignExtent(rHR.ext.prj, rLR, snap='in')
  dum <- raster(rHR.ext.prj.al,crs=crs(rLR), res=res(rLR)/10)
  rHR.prj <- projectRaster(from = rHR, to=dum, filename = outname, overwrite=TRUE)
}
```

This function can be applied like this:

```
# create a new directory to place HiRes imagery
dir.create(paste0(wpath,'DATA/HiRes_ndvi_prj'),showWarnings = F)

# get all files...
HRlist <- list.files(path=paste0(wpath,'DATA/HiRes_ndvi'), pattern = 'NDVI_CSudmipy', full.names = T)
test <- project.HiRes(HRlist[1])
plot(test)
```

Now we want to do this for all images. This can be easily done by applying the function `project.HiRes` to all elements of the list using a single code line:

```
lapply(HRlist,project.HiRes)
```

Another way to do this faster is using the `parallel` package, to distribute each function run to a different processor on your machine. This works on Linux, but may not work well on Windows.

```
require(parallel)

# see how many cores you have
ncores <- detectCores()

# and apply the function to them all...
mclapply(HRlist,project.HiRes,mc.cores=ncores-1) # we keep (at least) one core free...
```

Now we can make the same thing for the `LoRes` imagery to crop these accordingly:

```
# create a new directory for LoRes
dir.create(paste0(wpath,'DATA/LoRes_ndvi_crp'),showWarnings = F)

# a new function...
LRlist <- list.files(path=paste0(wpath,'DATA/LoRes_ndvi'), pattern = 'NDVI_CSudmipy', full.names = T)
crop.LRlist <- function(iIMG){
  outname <- paste0(wpath,'DATA/LoRes_ndvi_crp/',basename(iIMG))
  rLR <- raster(iIMG)
  rLR.crp <- crop(rLR,extent(rHR.prj))
  rLR.crp <- writeRaster(rLR.crp, filename = outname, format='GTiff', overwrite=T)
}

# and batch apply it
mclapply(LRlist,crop.LRlist,mc.cores=ncores-1)

#repeat for VZA values
LRlist_VZA <- sub('NDVI','VZA',LRlist)
mclapply(LRlist_VZA,crop.LRlist,mc.cores=ncores-1)
```

Extracting time series of data

This next step will show you how to make raster stacks for the regions of interest. We will then extract the time series of a given number of pixels and explore how the 3 instruments (SPOT, MODIS/TERRA and MODIS/AQUA) provide different sets of observations.

First we start by making raster stacks as follows for MODIS TERRA imagery:

```
# create stacks of MODIS TERRA NDVI and VZA..
LRlist_AM <- list.files(path=paste0(wpath,'DATA/LoRes_ndvi_crp'),
                        pattern = 'TERRA.+._NDVI_CSudmipy', full.names = T)
LRstack_AM <- stack(LRlist_AM)
```

We need then to get a similar stack for the VZA:

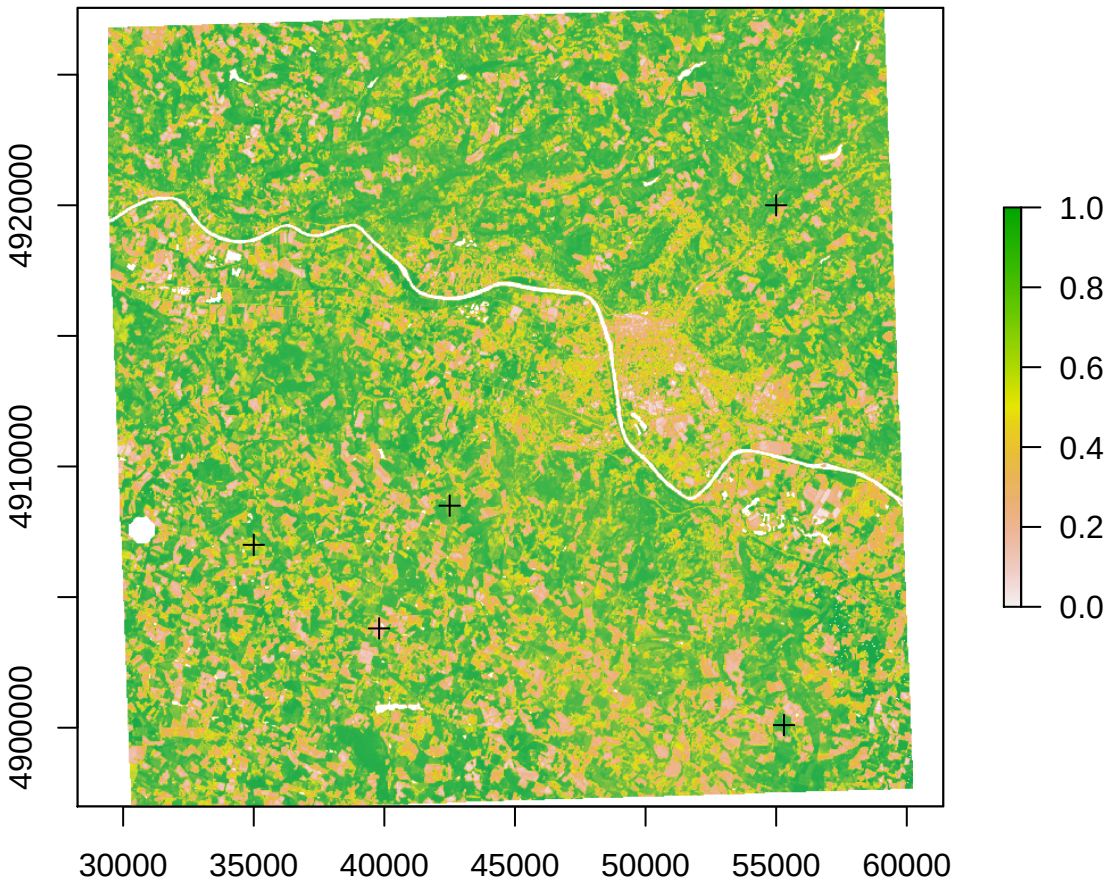
```
LRlist_AM_VZA <- sub('NDVI','VZA',LRlist_AM)
LRstack_AM_VZA <- stack(LRlist_AM_VZA)
```

We then do the same for MODIS AQUA and the series of SPOT high resoluton imagery.

```
# create stacks of MODIS AQUA NDVI and VZA..
LRlist_PM <- list.files(path=paste0(wpath,'DATA/LoRes_ndvi_crp'),
                        pattern = 'AQUA_.+._NDVI_CSudmipy', full.names = T)
LRstack_PM <- stack(LRlist_PM)

LRlist_PM_VZA <- list.files(path=paste0(wpath,'DATA/LoRes_ndvi_crp'),
                            pattern = 'AQUA_.+._NDVI_CSudmipy', full.names = T)
LRstack_PM_VZA <- stack(LRlist_PM_VZA)

# create stacks of SPOT4 NDVI ...
HRlist <- list.files(path=paste0(wpath,'DATA/HiRes_ndvi_prj'),
                     pattern = 'NDVI_CSudmipy', full.names = T)
HRstack <- stack(HRlist)
```

We now will define some georeferrenced points. You can choose your own, but make sure they are within the extents of our area of interest.

```
# make a dataframe with points...
df <- data.frame(x=c(  39800,  35000,  55300,  42500, 55000),
                 y=c(4903800,4907000,4900100,4908500, 4920000))
# create a spatialPoints object
sp.df <- SpatialPoints(coords = df,proj4string=crs(LRstack_AM))


plot(HRstack[[13]],zlim=c(0,1))
plot(sp.df,add=T)
```



Next, we need to extract the corresponding information in the stacks from the points.

```
# extract data///
HR.ts <- extract(HRstack,sp.df)
LR_AM.ts <- extract(LRstack_AM,sp.df)
LR_PM.ts <- extract(LRstack_PM,sp.df)
LR_AM.vza.ts <- extract(LRstack_AM_VZA,sp.df)
LR_PM.vza.ts <- extract(LRstack_PM_VZA,sp.df)
```

We need to reshape the data in a more convenient format. To do so, we will use the package `reshape2` that provides the useful function `melt`. So we start with `HR.ts`, which is in an 5 by 11 table.

```
require(reshape2)
```

```
## Loading required package: reshape2
```

```
# For SPOT
df.HR <- melt(HR.ts, value.name = "NDVI",na.rm = T)
```

The resulting dataframe looks like this:

```
##   Var1                                        Var2      NDVI
## 1    1 SPOT4_HRVIR_XS_20130216_N2A_NDVI_CSudmipy 0.1891448
## 2    2 SPOT4_HRVIR_XS_20130216_N2A_NDVI_CSudmipy 0.2465568
## 3    3 SPOT4_HRVIR_XS_20130216_N2A_NDVI_CSudmipy 0.4914964
## 4    4 SPOT4_HRVIR_XS_20130216_N2A_NDVI_CSudmipy 0.5788316
## 5    5 SPOT4_HRVIR_XS_20130216_N2A_NDVI_CSudmipy 0.4970773
## 6    1 SPOT4_HRVIR_XS_20130221_N2A_NDVI_CSudmipy 0.1564364
```

We can improve it by adding the date and other infor:

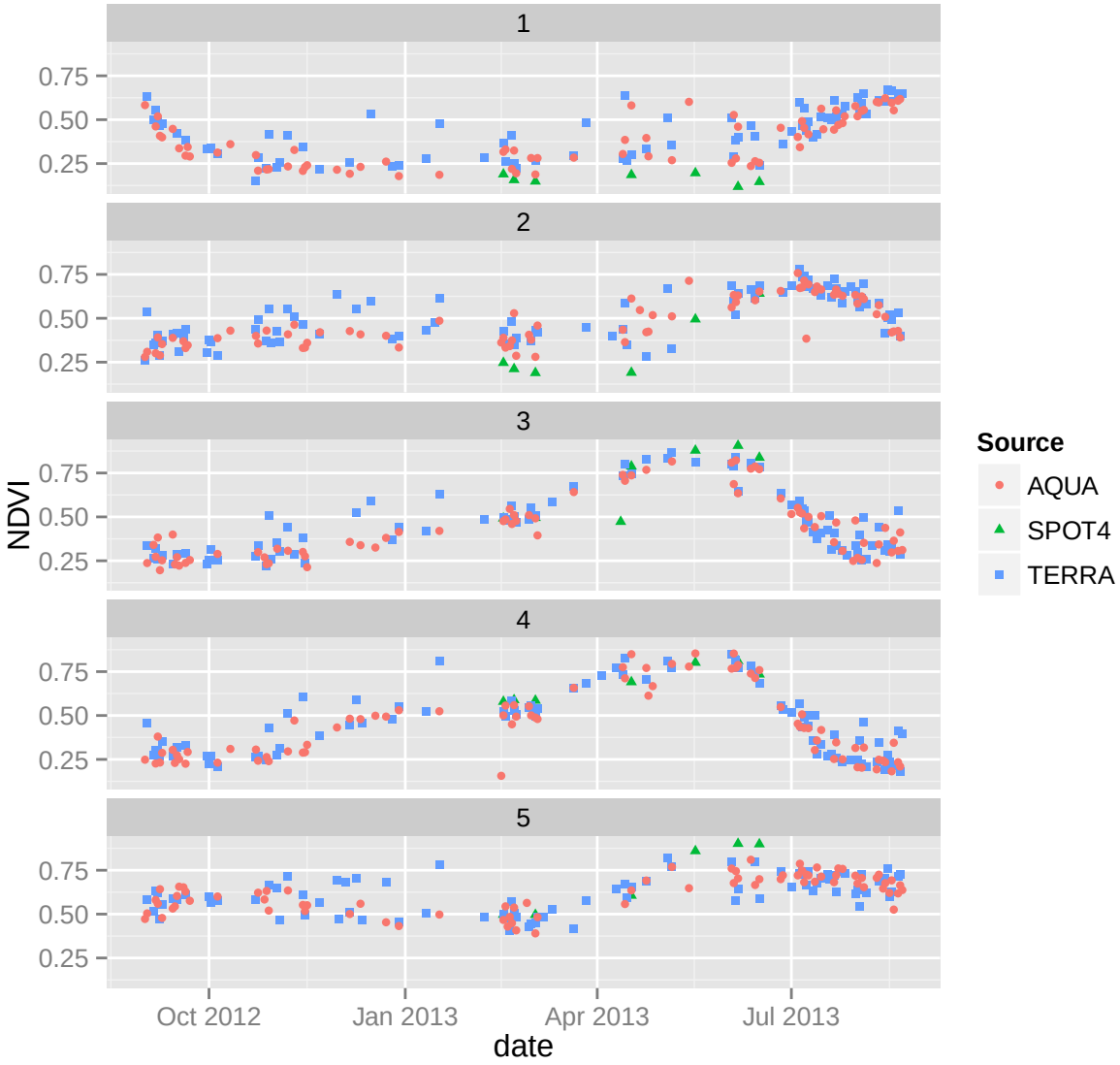
```
df.HR$date <- as.Date(df.HR$Var2,'SPOT4_HRVIR_XS_%Y%m%d_N2A_NDVI_CSudmipy')
df.HR$Source <- substr(df.HR$Var2,1,5)
df.HR$VZA <- 0 # let us consider it all at nadir (which is FALSE in reality)

# For MODIS TERRA
df.LR_AM <- melt(LR_AM.ts, value.name = "NDVI",na.rm = T)
df.LR_AM$VZA <- melt(LR_AM.vza.ts, value.name = "VZA",na.rm = T)$VZA
df.LR_AM$date <- as.Date(df.LR_AM$Var2,'TERRA_MODIS_%Y%j_NDVI_CSudmipy')
df.LR_AM$Source <- substr(df.LR_AM$Var2,1,5)


# For MODIS AQUA
df.LR_PM <- melt(LR_PM.ts, value.name = "NDVI",na.rm = T)
df.LR_PM$VZA <- melt(LR_PM.vza.ts, value.name = "VZA",na.rm = T)$VZA
df.LR_PM$date <- as.Date(df.LR_PM$Var2,'AQUA__MODIS_%Y%j_NDVI_CSudmipy')
df.LR_PM$Source <- substr(df.LR_PM$Var2,1,4)

df <- rbind(df.HR,df.LR_AM,df.LR_PM)
colnames(df)[1:2] <- c('PointID','Source_long')
```

For visualizing, we will use the `ggplot2` package, a great tool for visualization that forces you to keep your data tidy (partly by forcing you to use organised dataframes).

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

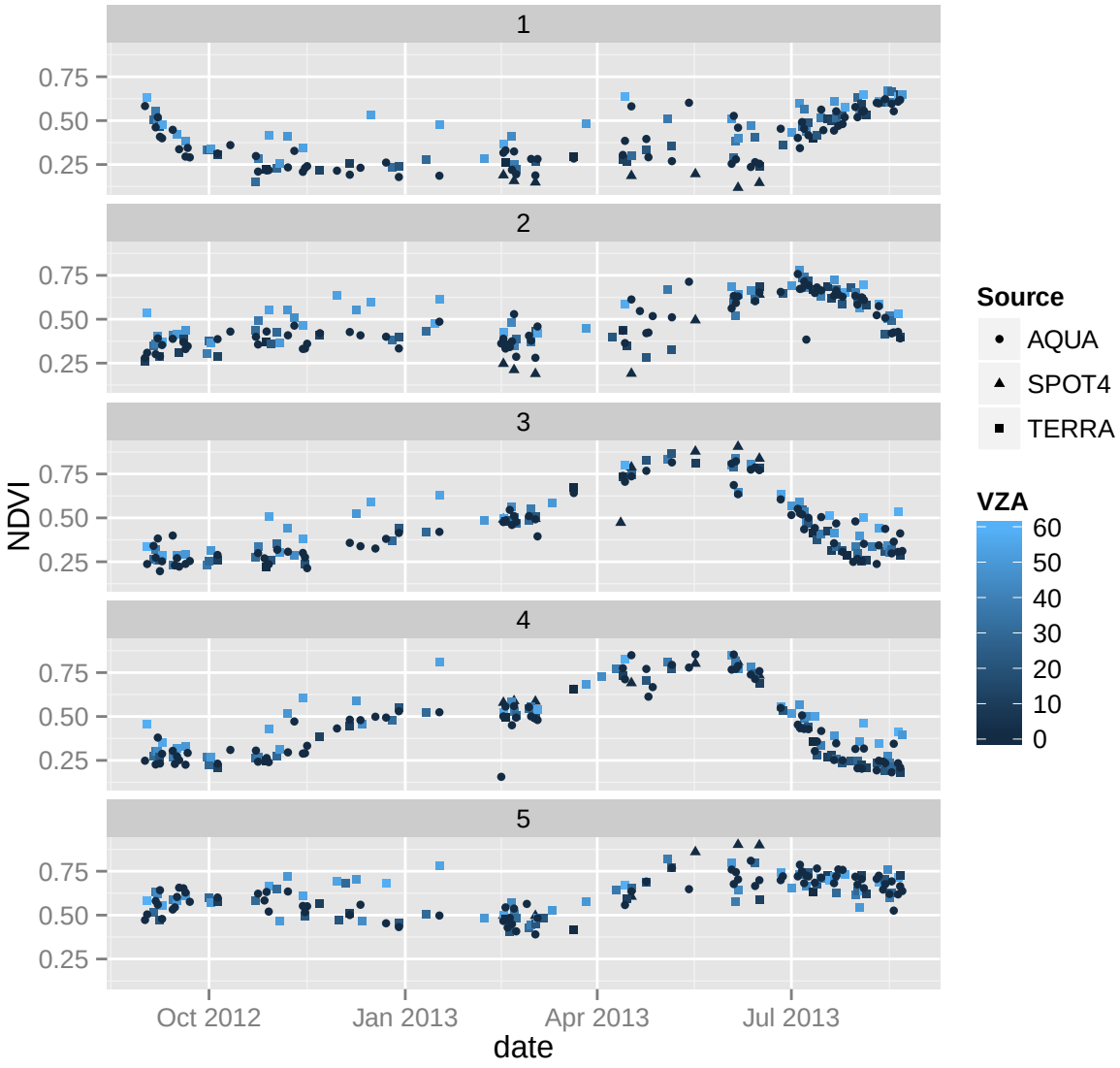
```
ggplot(df,aes(x=date,y=NDVI,shape=Source,colour=Source))+
  geom_point(size=1.5)+
  facet_wrap(~PointID,ncol=1)
```



Notice the different shapes of the curves (linked to different crop types) but also the differences in the noise (due to differences in sub-pixel heterogeneity). Notice also the difference in observation density between the SPOT and the MODIS data.

Another interesting graph to make is to use different colours for different VZA, to see if these are linked to the noise in the temporal profile:

```
ggplot(df,aes(x=date,y=NDVI,shape=Source,colour=VZA))+
  geom_point(size=1.5)+
  facet_wrap(~PointID,ncol=1)
```



Read temperature data

The last source of data to explore consists of ancillary non-remotely-sensed information: temperature. Start by reading the stack (which is in NetCDF format, but which can be read with `raster` all the same if the `ncdf` package is installed).

So here we go:

```
# Load data
avgT <- stack(paste0(wpath,'DATA/Temperature/','AvgTemp_EU.nc'))
```

```
## Loading required namespace: ncdf
```

Explore some data visualization

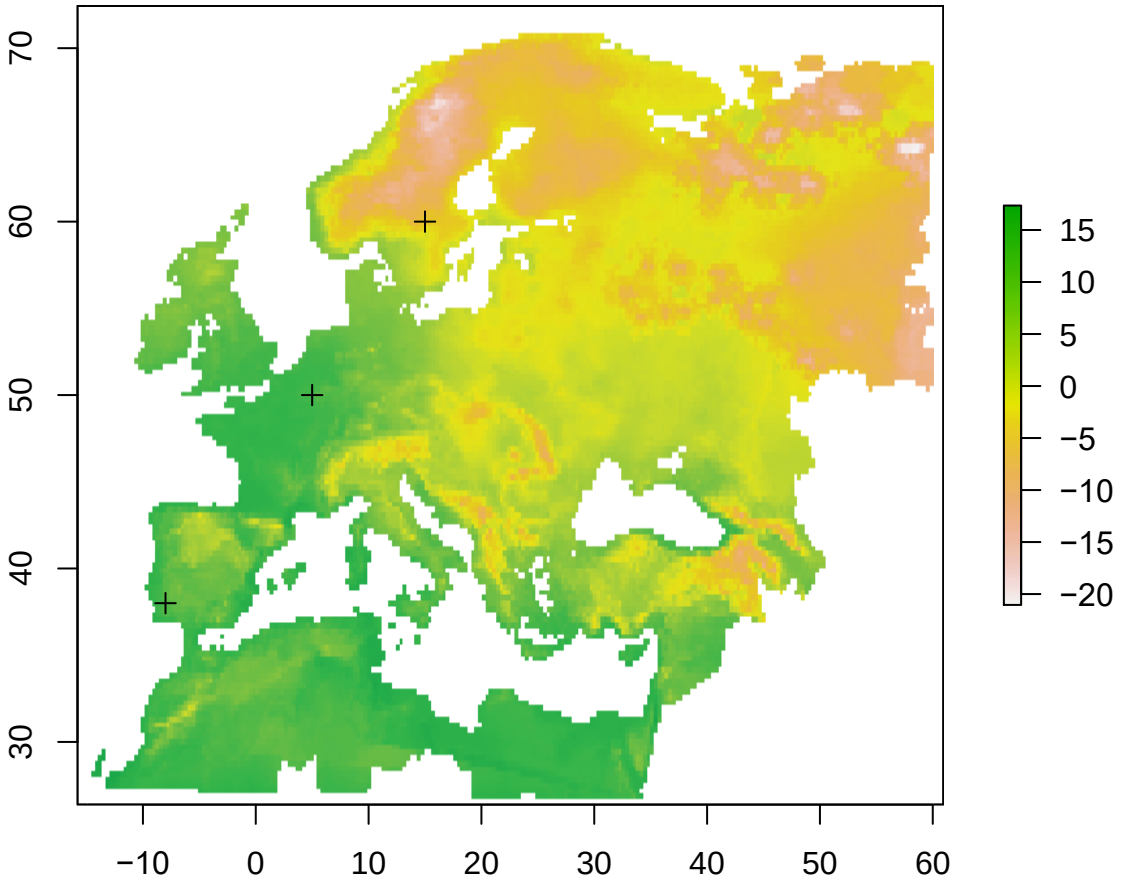
```
plot(avgT[[1]])
plot(avgT[['X2013.07.02']])
plot(avgT[[seq(1,365,7)]],zlim=c(-30,20))
```

Define and extract some points

```
sp.df.T <- SpatialPoints(coords =data.frame(x=c(-8,5,15),y=c(38,50,60)),
                         proj4string=crs(avgT))
```

You can check their position on one layer of the stack:

```
plot(avgT[[1]])
plot(sp.df.T, add=TRUE)
```



And now extract and melt the results:

```
df.t <- extract(avgT,sp.df.T)
df.t <- melt(df.t)
```

Do some changes in the dataframe

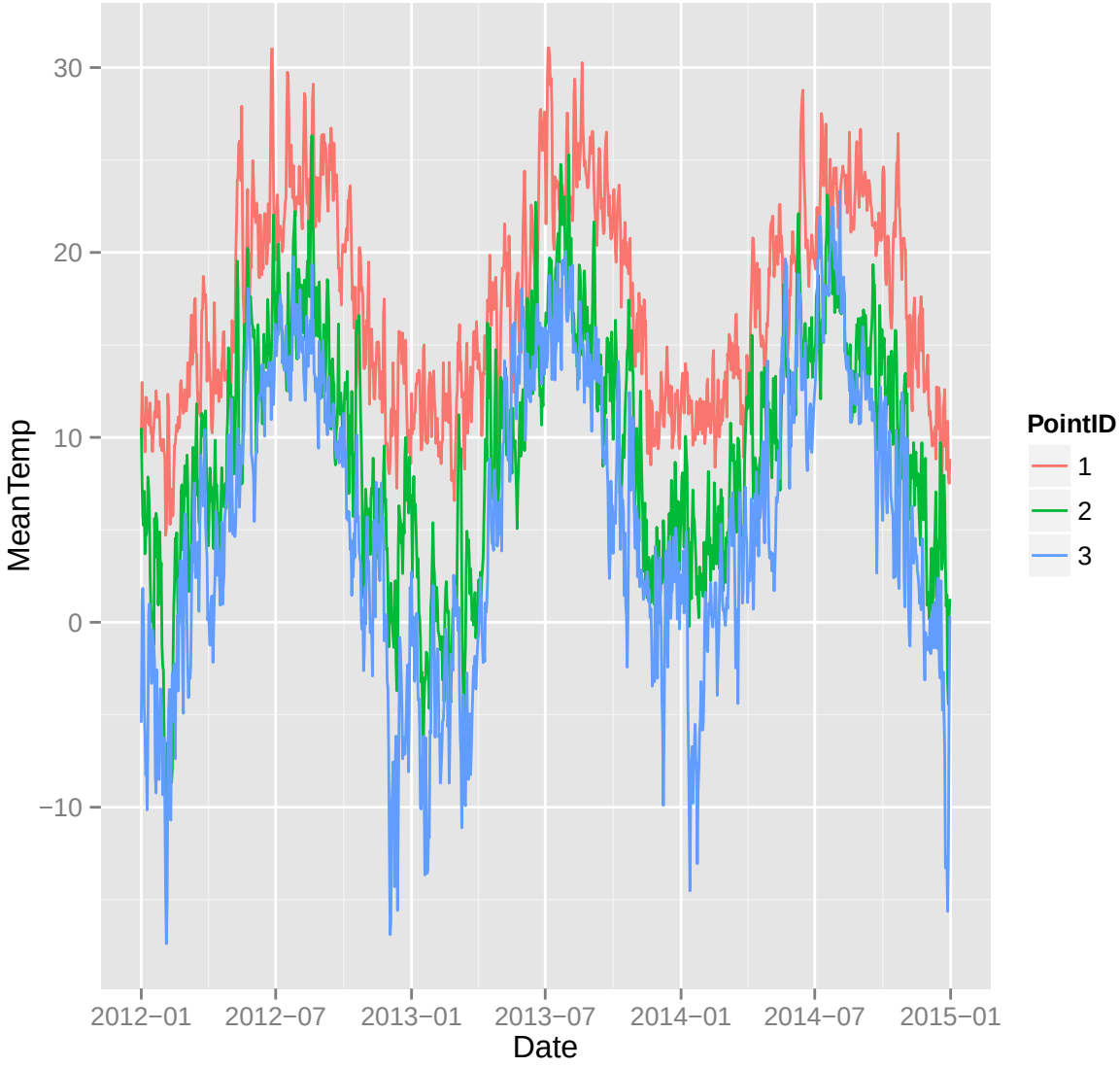
```
colnames(df.t) <- c('PointID','Date','MeanTemp')
df.t$Date <- as.Date(df.t$Date,"X%Y.%m.%d")
df.t$PointID <- as.factor(df.t$PointID)
```

The resulting data frame should look like this:

```
##   PointID       Date  MeanTemp
## 1       1 2012-01-01 10.128602
## 2       2 2012-01-01 10.509105
## 3       3 2012-01-01 -5.432161
## 4       1 2012-01-02 12.975585
## 5       2 2012-01-02  7.485018
## 6       3 2012-01-02  1.199917
```

And it can be used to plot this graph:

```
# the ggplot
ggplot(df.t,aes(x=Date,y=MeanTemp,colour=PointID))+
  geom_line()
```



Calculate degree days for one point

In this section, we will try to calculate growing degree days (GDD) for a time series.

We start by extracting again temperature data for our original points in our local study area.

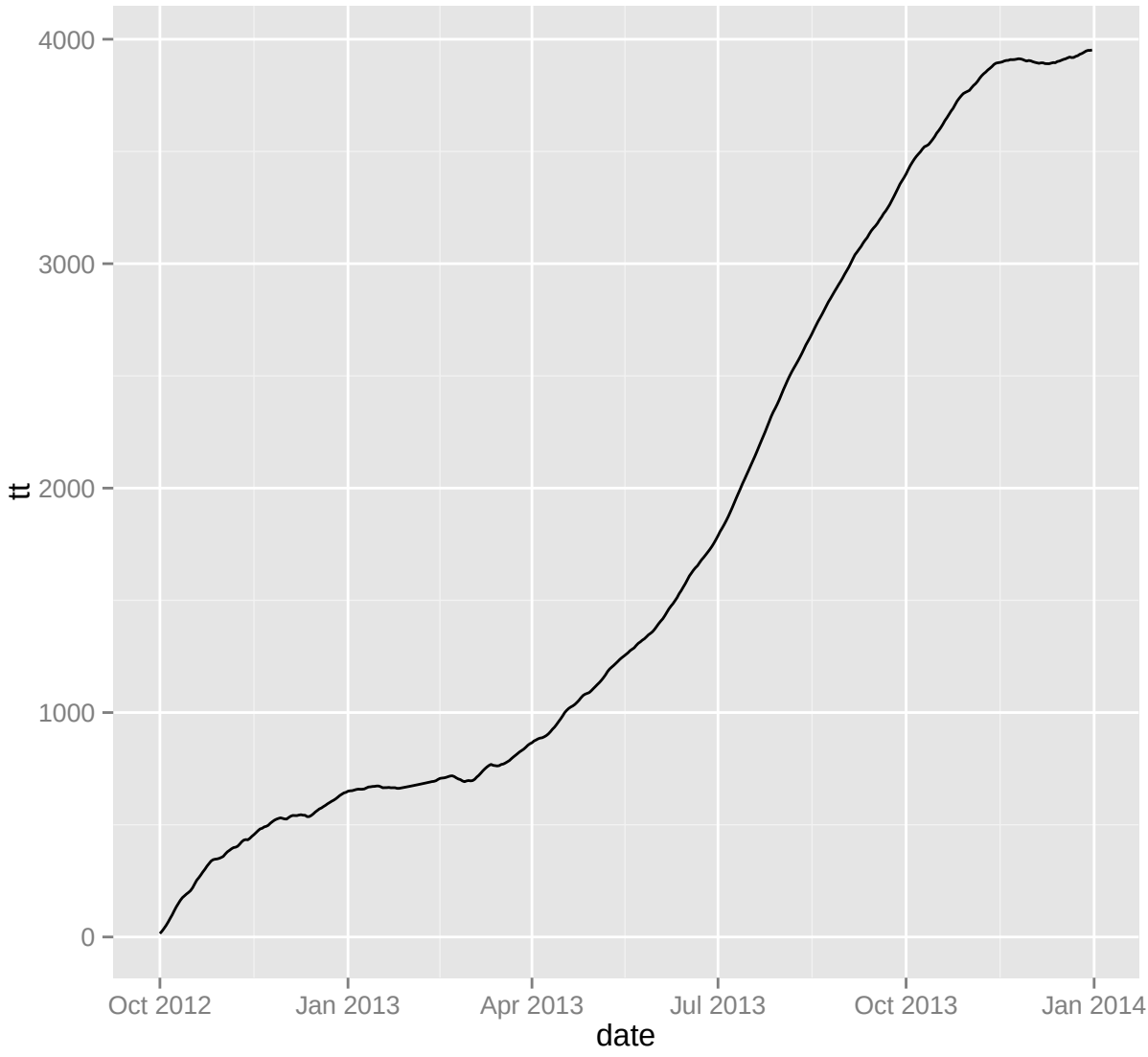
```
dumt <- extract(avgT,sp.df)

df.t2 <- melt(dumt, varnames = c('PointID','CodeDate'), value.name = "MeanTemp")
df.t2$date <- as.Date(df.t2$CodeDate,'X%Y.%m.%d')
```

We need to subset the temperature data in time to match our time series of images:

```
# subset in time and space (i.e. get a single point)
Idx <- df.t2$date >= as.Date('2012-10-01')
Idx <- Idx & (df.t2$date < as.Date('2014-01-01'))
Idx <- Idx & (df.t2$PointID == 4)
df.tt <- df.t2[Idx,]
```

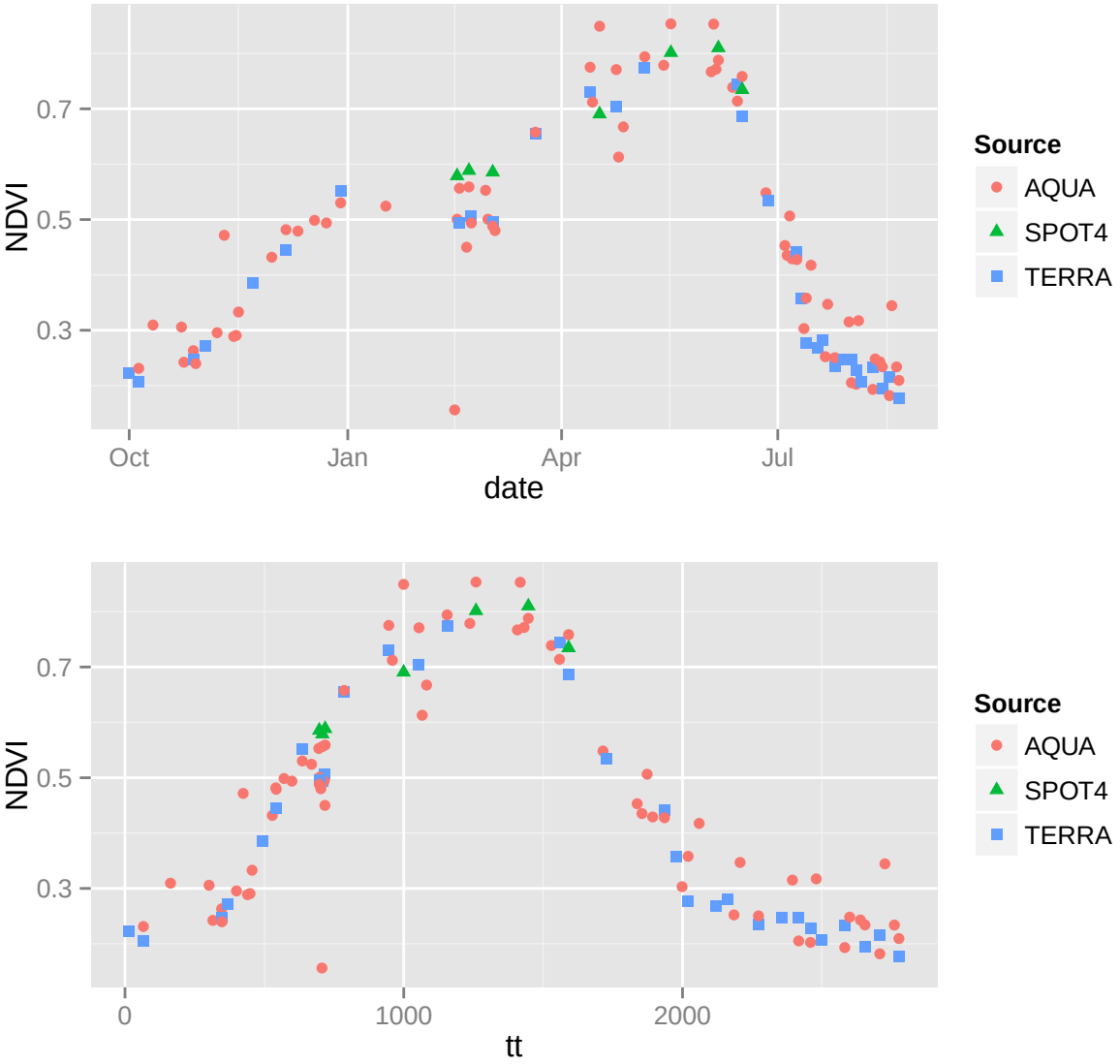
Now we calculate the growing degree days above a base temperature of 4 (commonly used for winter wheat).

```
# calc degree days
Tbase = 4
df.tt$tt <- df.tt$MeanTemp
for(i in 2:dim(df.tt)[1]){ df.tt$tt[i] <- (df.tt$tt[i] - Tbase)+df.tt$tt[i-1]}

#plot to see difference between time and ttime
ggplot(df.tt,aes(x=date,y=tt))+geom_line()
```




```
# fuse with RS data
df.pt1 <- merge(df,df.tt,by=c('PointID','date'),all=F)

# remove strong vza
df.pt1 <- df.pt1[df.pt1$VZA<30,]

# plot normal time
g1 <- ggplot(df.pt1,aes(x=date,y=NDVI,colour=Source,shape=Source))+geom_point(size=2)

# plot thermal time
g2 <- ggplot(df.pt1,aes(x=tt,y=NDVI,colour=Source,shape=Source))+geom_point(size=2)

# small trick to plot them both together...
require(gridExtra)

## Loading required package: gridExtra
## Loading required package: grid

gBoth <- arrangeGrob(g1,g2,ncol=1)
```

```
gBoth
```



Notice the difference at the middle of the growing part of the curves: with thermal time it becames smoother because time is warped according to the rate of growh of the crop which is dependent on temperature.

Calculate SNR for one time series MODIS imagery

The final step in this tutorial consists in applying a function on the daily MODIS raster stacks in order to calculate a proxy for spatial heterogeneity. This proxy is named the temporal signal-to-noise ratio (SNR). The rationale for using it (particularly for agricultural landscapes) and a demonstration over various sites is provided in the paper: Duveiller et al. 2015 RSE (http://dx.doi.org/doi:10.1016/j.rse.2015.06.001).

We start by defining a function that will calculate this SNR on a given time series.

```
# function to calculate the SNR
funSNR=function(z){
  d <- which(!is.na(z)==T)
  if(length(d) > 15){ # we define a min of 15 obs per time seires
```

```
    s <- smooth.spline(d, z[d], df=8)
    SNR <- var(s$y)/var(s$y-z[d])}
  else { SNR <- NA }
  return(SNR)
}
```

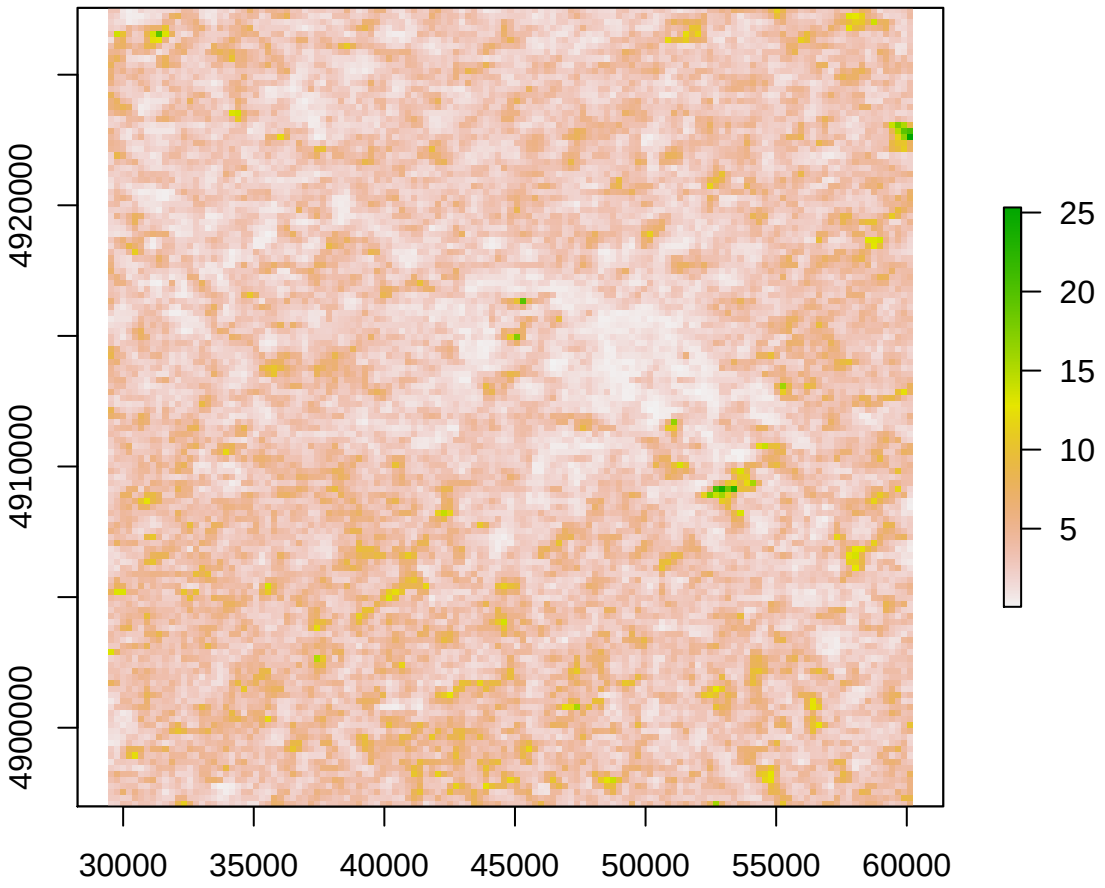
We can apply it to one of our previously extracted time series.

```
dum <- melt(LR_AM.ts, value.name = "NDVI")
z <- dum$NDVI[which(dum$Var1==3)]
snr <- funSNR(z)
```

```
## [1] 5.863703
```

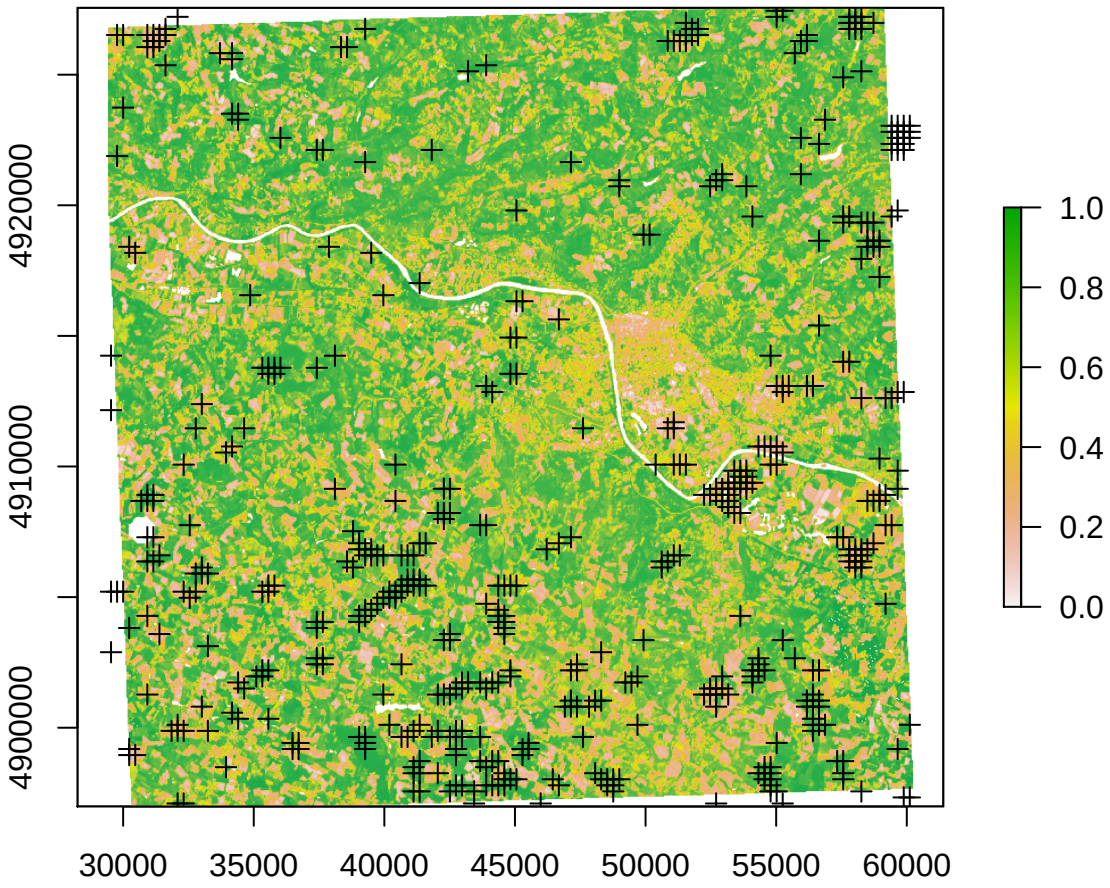
Now, to apply it to the entire stack, we will process by blocks...

```
# a function to divide in convenient blocksizes with respect to memory
bs <- blockSize(LRstack_PM)

# a fucntion to apply the funSNR function to each block
process.SNR.block <- function(i){
  dumBrick <- crop(LRstack_PM,extent(LRstack_PM,bs$row[i],bs$row[i]+bs$nrows[i]-1,1,dim(LRstack_PM)[2]))
  rasterSNR <- calc(dumBrick,funSNR)
  return(rasterSNR)
}

# Apply using parallel (or simply lapply)
outList <- mclapply(X = 1:bs$n, FUN = process.SNR.block, mc.cores = 4)

# merge all blocks together
snr <- do.call(merge,outList)

# and plot the result
plot(snr)
```



The resulting image indicates the spatial heterogeneity of the MODIS time series. We can do a sub-selection of the most temporally coherent time series and overlay them over the HR image. This sub-selection consists of purer pixels.

```
cellVct <- as.vector(snr>8)
xyList<-xyFromCell(snr, which(cellVct==T), spatial=T)

plot(rHR.prj,zlim=c(0,1))
plot(xyList,add=T)
```


4920000
4910000
4900000
30000
35000
40000
45000
50000
55000
60000
1.0
0.8
0.6
0.4
0.2
0.0