Monitoring Vegetation in a Changing Climate [D5P3a]

Gregory Duveiller

August 3, 2018

Introduction

This document outlines the steps that will be taken in the practicals of *Monitoring Vegetation in a Changing Climate* of the ESA land training course 2017 (D4P1b). The objectives of this session are to familiarize students with the R programming language (https://www.r-project.org/) particularly in the context of analyzing spatialized scientific datasets used in climate science.

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. A special focus will be given to some packages within what is now known as the `tidyverse`. Quoting from the dedicated website (https://www.tidyverse.org/): "The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs." This suite of packages is sometimes considered as a specific R dialect. It will be particularly useful to manipulate data (`dplyr` package) and to visualize it (`ggplot2` package).

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:

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

Packages

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 used in this practical is the `ncdf4` package, useful to read and manipulate NetCDF files. The following lines show how install it:

```r
install.packages('ncdf4')
```

and load it:

```r
require(ncdf4)
```

```
## Loading required package: ncdf4
```

The following packages also need to be installed.

```r
install.packages(c('tidyverse','rgdal','raster','scales','RColorBrewer'))
```

Quick introduction to dataframes

A versatile way to store data in R is using data.frames. These are essentially tables with data in them. A good way to keep data tidy in data.frames, following precepts of the `tidyverse`, is too keep each variable is in its own column, and each observation, or case, is in its own row.

R comes with some examples of data built into it, such as the `CO2` dataset. To get a quick view of the data, use the the `head` command:

```
head(CO2)
```

```
##   Plant   Type  Treatment conc uptake
## 1   Qn1 Quebec nonchilled   95   16.0
## 2   Qn1 Quebec nonchilled  175   30.4
## 3   Qn1 Quebec nonchilled  250   34.8
## 4   Qn1 Quebec nonchilled  350   37.2
## 5   Qn1 Quebec nonchilled  500   35.3
## 6   Qn1 Quebec nonchilled  675   39.2
```

To known more about how the dataset is structured, try the following command.

```
str(CO2)
```

```
## Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   84 obs. of  5 variables
##  $ Plant    : Ord.factor w/ 12 levels "Qn1"<"Qn2"<"Qn3"<..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ Type     : Factor w/ 2 levels "Quebec","Mississippi": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Treatment: Factor w/ 2 levels "nonchilled","chilled": 1 1 1 1 1 1 1 1 1 1 ...
##  $ conc     : num  95 175 250 350 500 675 1000 95 175 250 ...
##  $ uptake   : num  16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
##  - attr(*, "formula")=Class 'formula'  language uptake ~ conc | Plant
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "outer")=Class 'formula'  language ~Treatment * Type
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "labels")=List of 2
##   ..$ x: chr "Ambient carbon dioxide concentration"
##   ..$ y: chr "CO2 uptake rate"
##  - attr(*, "units")=List of 2
##   ..$ x: chr "(uL/L)"
##   ..$ y: chr "(umol/m^2 s)"
```

ggplot2

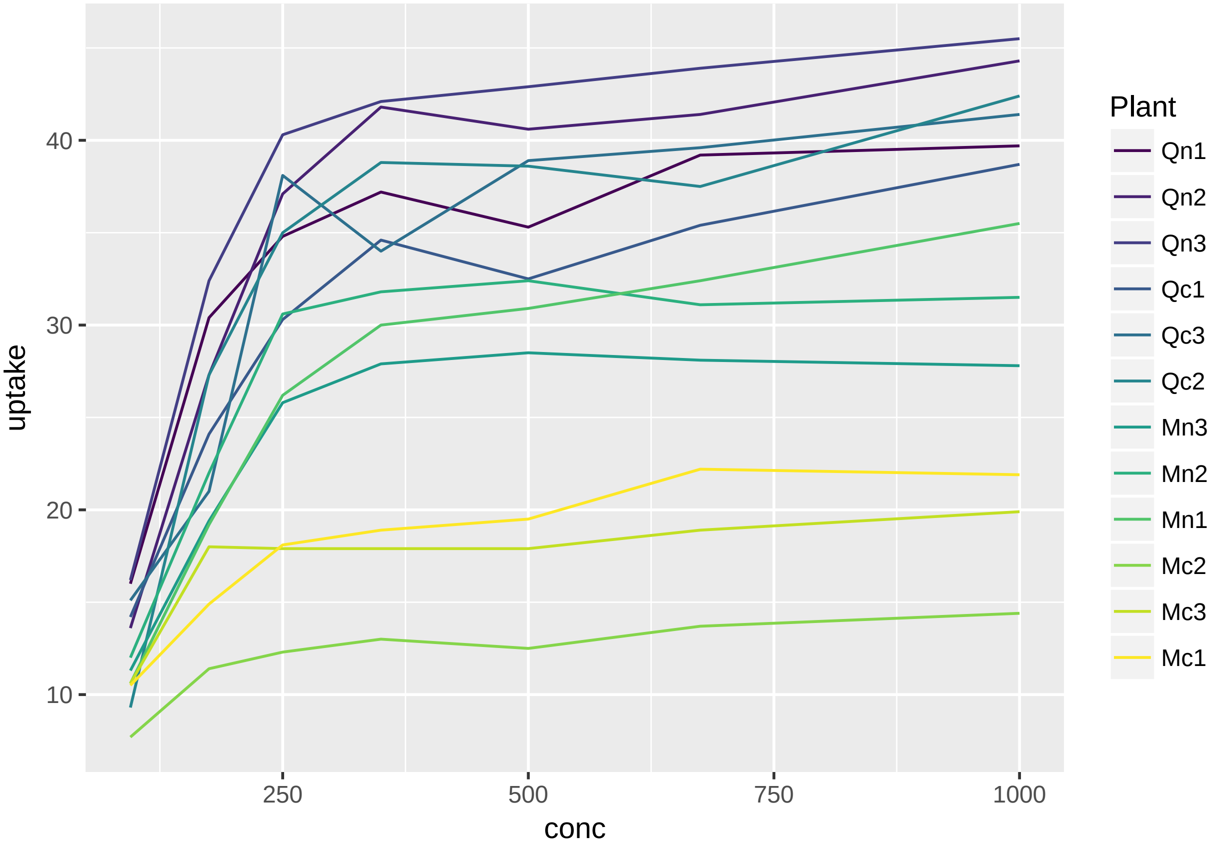
The package called `ggplot2` allows to visualize the data in data.frames. It follows a logic called *the grammar of graphics*. Start by loading the package.

```
require(ggplot2)
```

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

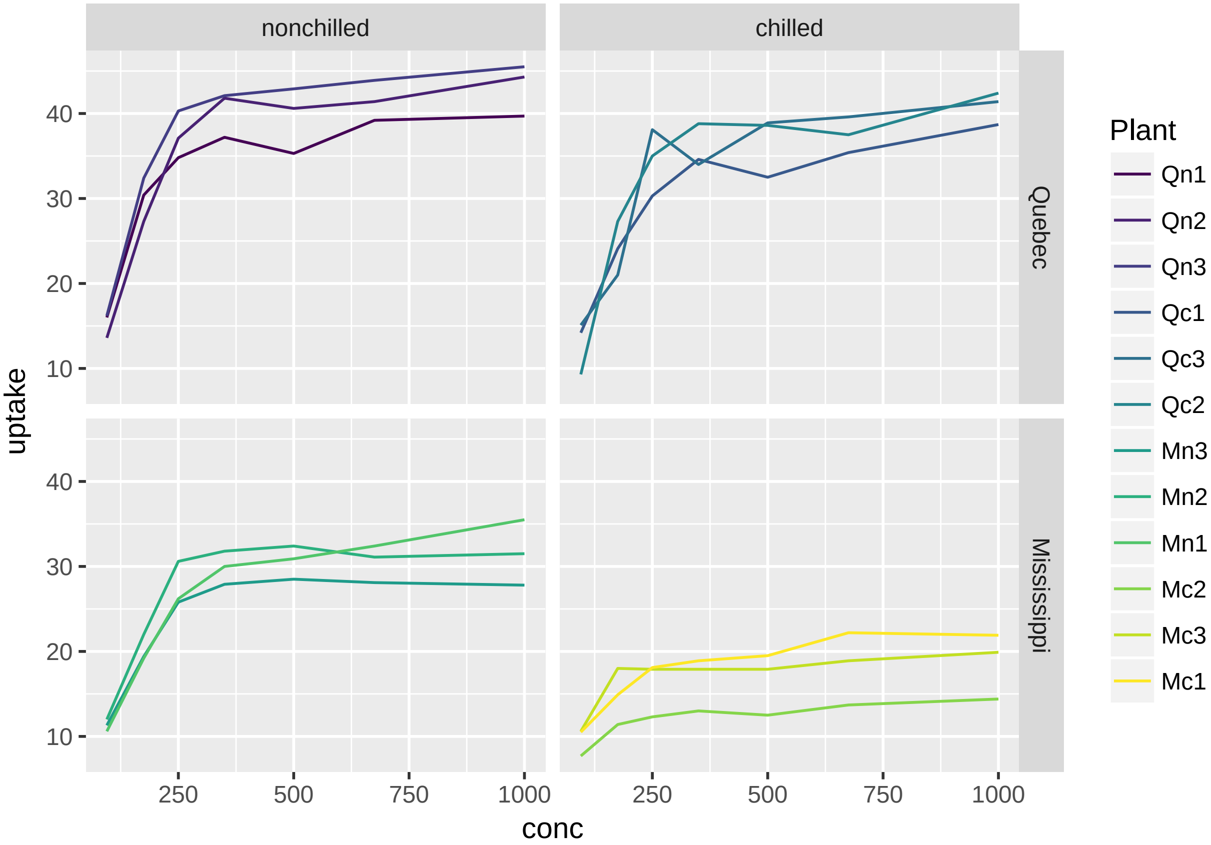
A basic principle is that variables of the dataframe must be explicitly mapped unto elements of the plots, known as *aesthetics*. This has to be done for every *geometry*, or object that needs to be plotted. As an example, the `conc` and `uptake` fields in `CO2` are here mapped to the `x` and `y` axis, while another axis `colour` is used to separate the information in the field `Plant`.

```
ggplot(CO2)+geom_line(aes(x=conc,y=uptake,colour=Plant))
```



With ggplot2 it is easy to analyze data with different facets, here for instance to bring out the extra information contained in the remaining fields of the dataframe (Type and Treatment).

```
ggplot(CO2)+
  geom_line(aes(x=conc,y=uptake,colour=Plant))+
  facet_grid(Type~Treatment)
```



Hopefully, more of the possibilities offered in `ggplot2` will become evident in the examples we will construct later on.

dplyr

The `dplyr` package is very useful to manipulate data in the data.frames. This will be used all along this tutorial. It is an essential part of the `tidyverse`. This package allows to nest a series of operations using the pipe operator `%>%`. This will become clearer later on.

The RStudio team has prepared a series of cheat sheets, among which there is one for `dplyr` and one for `ggplot2`, all available here: https://www.rstudio.com/resources/cheatsheets/.

Importing data

The first major step in this tutorial is to the necessary data. The exercise will focus two indicators of vegetation, namely NDVI and SIF. NDVI in a mathematical index calculated based on reflectance in the NIR and RED sections of the spectrum. The normalized difference of these bands (known as NDVI) is a good proxy for vegetation greenness. SIF stands for sun-induced fluorescence. It is a faint signal that can be derived from hyperspectral data and that is apparently strongly related to gross primary productivity (GPP) of vegetation.

SIF is potentially a very valuable signal but it is difficult to obtain since no dedicated sensor currently exists. The ESA Earth Explorer mission FLEX will be able to provide this signal but it is still being built. To have a single instrument with both SIF and NDVI, we use the dataset prepared by NASA from the GOME-2 instrument on-board of the MetOp-A instrument and available in this website: https://avdc.gsfc.nasa.gov/pub/data/satellite/MetOp/GOME_F/v27/MetOp-A/level3/.

The necessary images have been placed in a dedicated folder for this exercise. Find out where the data is on your machine and save the path as follows:

```
wpath <- '/Volumes/GREG/Work/Teaching/ESA_LTC_2017/Practical/DATA/'
setwd(wpath)
```

We will start by loading a single image defined by the following details.

```
time <- '20070601'
longname <- 'ret_f_nr5_nsvd12_v26_waves734_nolog.grid_SIF_v27'
fname <- paste(longname,time,'31.nc',sep='_')
```

Using the `ncdf4` package we can open the file and print the information that this command gives us.

```
require(ncdf4)
nc <- nc_open(filename = paste0(wpath,fname))
print(nc)
```

```
## File /Volumes/GregDrive/Work/Teaching/ESA_LTC_2017/Practical/DATA/ret_f_nr5_nsvd12_v26_waves734_nolog
##
##      9 variables (excluding dimension variables):
##         float SIF[longitude,latitude]
##             Title: SIF
##             Units: mW/m^2/nm/sr
##             Version: 1
##         float SIF_daily_average[longitude,latitude]
##             Title: SIF daily averaged based on clearsky PAR proxy
##             Units: mW/m^2/nm/sr
##             Version: 1
##         float SIF_std[longitude,latitude]
##             Title: SIF standard deviation
##             Units: mW/m^2/nm/sr
##             Version: 1
##         float Par_normalized_SIF[longitude,latitude]
##             Title: PAR_normalized-F F/cos(SZA) where F in mW/m^2/nm/sr at 737 nm
##             Units: none
##             Version: 1
##         float Par_normalized_SIF_std[longitude,latitude]
##             Title: PAR_normalized F standard deviation
##             Units: none
##             Version: 1
##         float NDVI[longitude,latitude]
##             Title: NDVI (not Rayleigh corrected)
##             Units: none
##             Version: 1
##         float NDVI_std[longitude,latitude]
##             Title: NDVI standard deviation
##             Units: none
##             Version: 1
##         float cos(SZA)[longitude,latitude]
##             Title: Cosine of solar zenith angle
##             Units: none
##             Version: 1
##         float Counts[longitude,latitude]
##             Title: Number of observations
##             Units: none
##             Version: 1
##
##      2 dimensions:
##         longitude  Size:720
##             Longitude: Longitude in degrees E
##             Units: Degrees E
##         latitude  Size:360
##             Latitude: Latitude in degrees N
##             Units: Degrees N
```

We are interested in the variables `SIF` and `NDVI`. How do we access them? One easy way to so if the data is spatialized is directly with the `raster` package.

```
require(raster)
```

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

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

```
##
## Attaching package: 'raster'
```

```
## The following object is masked from 'package:ggplot2':
##
##     calc
```

```
r <- raster(paste0(wpath,fname),varname='SIF')
```

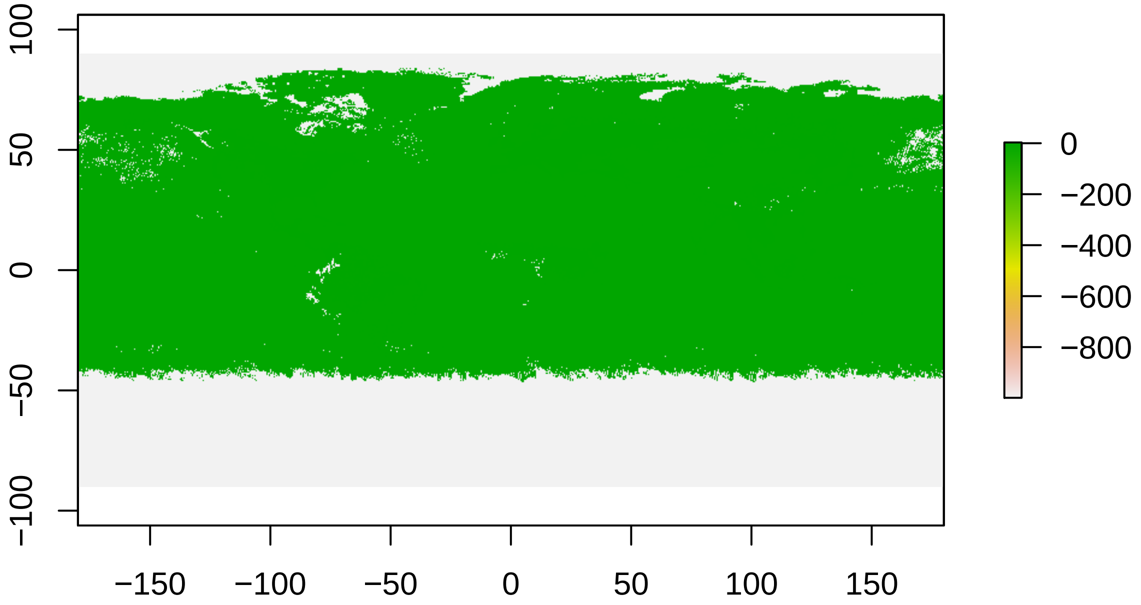
A raster object includes information of its spatial resolution and coordinate system, which can be accessed by just calling the object:

```
r
```

```
## class       : RasterLayer
## dimensions  : 360, 720, 259200  (nrow, ncol, ncell)
## resolution  : 0.5, 0.5  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : /Volumes/GregDrive/Work/Teaching/ESA_LTC_2017/Practical/DATA/ret_f_nr5_nsvd12_v26_waves
## names       : SIF
## zvar        : SIF
```

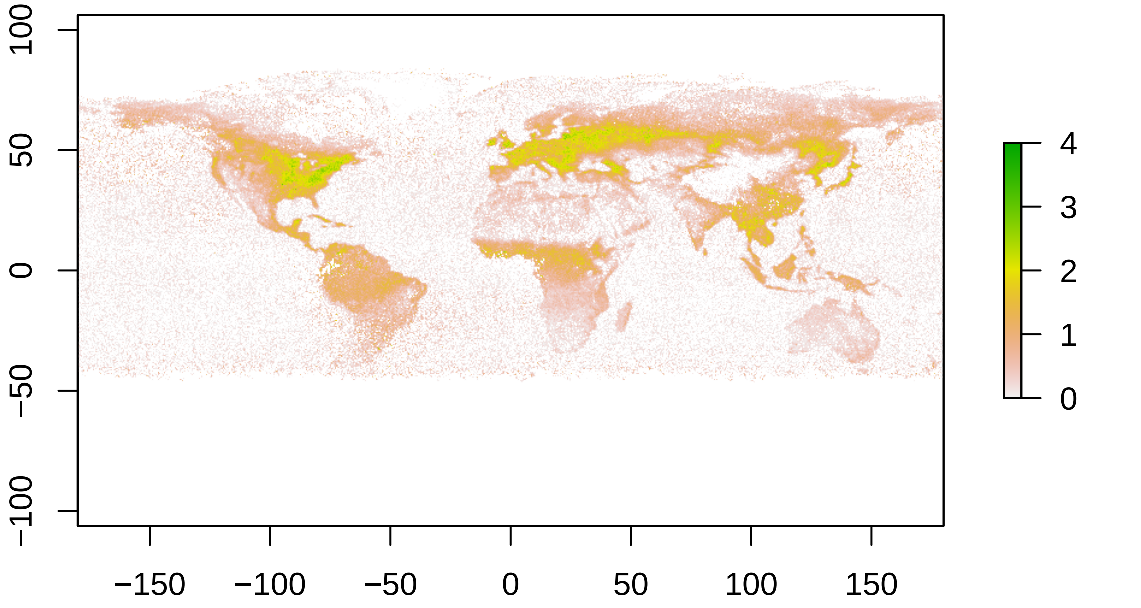
It can easily be plotted using the base R plots:

```
plot(r)
```



But the colour scale needs to be changed in order to see the data:

```
plot(r,zlim=c(0,4))
```



Using this raster format is very useful for spatial data. But here we want to extract the data directly from the netcdf and place it into a table. So we proceed using the `ncvar_get` command as follows:

```
sif <- ncvar_get(nc,varid = 'SIF',start = c(1,1),count = c(-1,-1))
ndvi <- ncvar_get(nc,varid = 'NDVI')
lon <- ncvar_get(nc,varid = 'longitude')
lat <- ncvar_get(nc,varid = 'latitude')
```

And now we place them in a data.frame:

```
df0 <- data.frame(lon=rep(lon,times=length(lat)),
                  lat=rep(lat,each=length(lon)),
                  sif=as.vector(sif),ndvi=as.vector(ndvi))
str(df0)
```

```
## 'data.frame':    259200 obs. of  4 variables:
##  $ lon : num  -180 -179 -179 -178 -178 ...
##  $ lat : num  89.8 89.8 89.8 89.8 89.8 ...
##  $ sif : num  -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 ...
##  $ ndvi: num  -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 ...
```

Apparently we need to filter out unwanted data. Some are set as `-999`. But first we will isolate the terrestrial part of the world by constructing a water mask from a land cover dataset.

We here use another netcdf describing continuously the presences of land cover types. It comes from the ESA CCI land cover product (viewable and downloadable here: http://maps.elie.ucl.ac.be/CCI/viewer/). The netcdf was generated using their dedicated User Tool and contains fractions of land covers, including one for water.

```r
fname <-paste0(wpath,'ESACCI-LC-L4-LCCS-Map-300m-P1Y-aggregated-0.050000Deg-2010-v2.0.7.nc')
PFT <- nc_open(filename = fname)
```

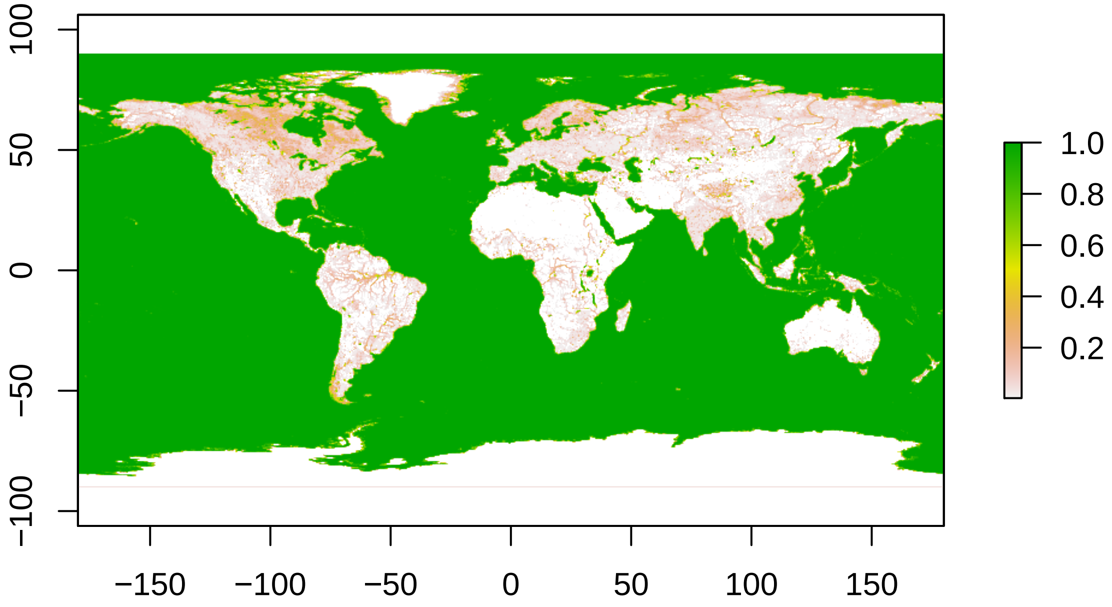
Here we will open it using the `raster` package.

```r
r.05 <- raster(fname,varname='WAT')
r.05
```

```
## class       : RasterLayer
## dimensions  : 3600, 7200, 25920000  (nrow, ncol, ncell)
## resolution  : 0.05, 0.05  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : /Volumes/GregDrive/Work/Teaching/ESA_LTC_2017/Practical/DATA/ESACCI-LC-L4-LCCS-Map-300m
## names       : WAT
## zvar        : WAT
```

so that we can easily aggregate it from its 0.05 decimal degree spatial resolution to the 0.5 in the SIF and NDVI dataset.

```r
r.50 <- aggregate(r.05,fact=10)
plot(r.50)
```



We set the no data values `NA` into 0 and built a new data.frame.

```
r.50[is.na(r.50)] <- 0
dfW <- data.frame(lat=rep(-seq(-89.75,89.75,0.5),each=ncol(r.50)),
                  lon=rep(seq(-179.75,179.75,0.5),times=nrow(r.50)),
                  wat=as.vector(r.50))
```

To join the two dataframes, we will use a practical join functionality of the `dplyr` package.

```
require(dplyr)
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:raster':
##
##     intersect, select, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df1 <- left_join(df0, dfW, by = c("lon", "lat"))
head(df1)
```

```
##       lon   lat  sif ndvi wat
## 1 -179.75 89.75 -999 -999   1
## 2 -179.25 89.75 -999 -999   1
## 3 -178.75 89.75 -999 -999   1
## 4 -178.25 89.75 -999 -999   1
## 5 -177.75 89.75 -999 -999   1
## 6 -177.25 89.75 -999 -999   1
```

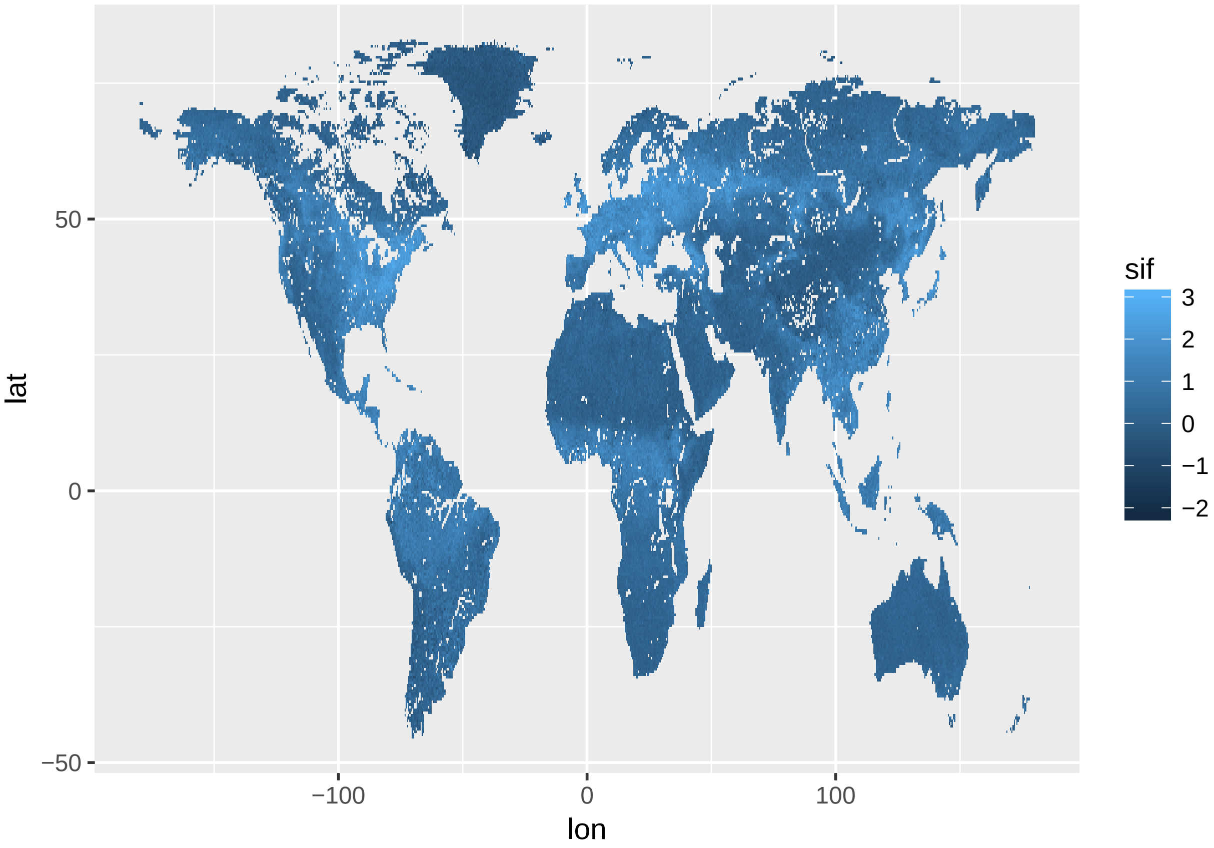
We can then filter out values set to -999 in the GOME-2 data and including more than 20% of water.

```
df2 <- filter(df1,sif!=-999,wat<0.2)
head(df2)
```

```
##      lon   lat         sif         ndvi       wat
## 1 -75.25 82.75 -0.18288906 -0.026392393 0.1920500
## 2 -74.75 82.75  0.05060377 -0.012912206 0.1142066
## 3 -72.25 82.75 -0.33275706 -0.012454201 0.1392164
## 4 -70.75 82.75  0.07926224  0.008922078 0.1861559
## 5 -70.25 82.75  0.11424994 -0.026155502 0.1018513
## 6 -69.75 82.75 -0.29370731 -0.011980801 0.1983110
```

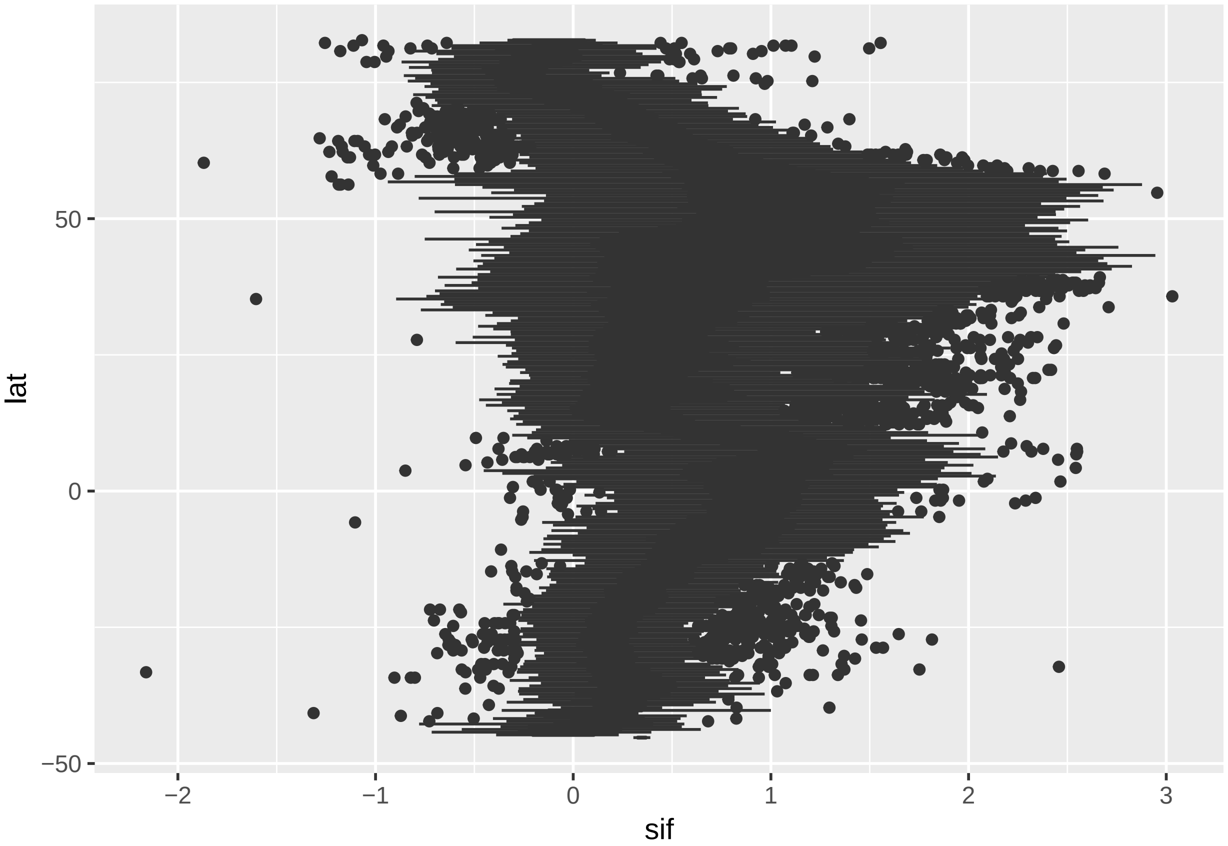
And now we plot the data with `ggplot2`

```
ggplot(df2) +
  geom_raster(aes(x=lon,y=lat,fill=sif))
```



Another way to visualize the data is to use boxplots over a latitudinal gradient:

```
ggplot(df2) +
  geom_boxplot(aes(x=lat,y=sif,group=lat))+
  coord_flip()
```



This plots shows how the higher values of SIF are in the tropics and the mid-latidudes of the Northern hemisphere, where it we expect higher GPP due to the presence of more productive ecosystems.

We still can add info on the date (even if for the moment we only have one moment in time)

```
df2$year=as.numeric(substr(time,1,4))
df2$month=month.abb[as.numeric(substr(time,5,6))]
head(df2)
```

```
##      lon   lat         sif         ndvi       wat year month
## 1 -75.25 82.75 -0.18288906 -0.026392393 0.1920500 2007   Jun
## 2 -74.75 82.75  0.05060377 -0.012912206 0.1142066 2007   Jun
## 3 -72.25 82.75 -0.33275706 -0.012454201 0.1392164 2007   Jun
## 4 -70.75 82.75  0.07926224  0.008922078 0.1861559 2007   Jun
## 5 -70.25 82.75  0.11424994 -0.026155502 0.1018513 2007   Jun
## 6 -69.75 82.75 -0.29370731 -0.011980801 0.1983110 2007   Jun
```

And now we extent these operations to the entire dataset. To do so we start by defining a function in which every step is taken:

```
get_data <- function(fname){

  nc <- nc_open(filename = fname)

  sif <- ncvar_get(nc,varid = 'SIF',start = c(1,1),count = c(-1,-1))
  ndvi <- ncvar_get(nc,varid = 'NDVI')
  lon <- ncvar_get(nc,varid = 'longitude')
  lat <- ncvar_get(nc,varid = 'latitude')

  time <- substr(x = basename(fname), start = 50, stop=57)
  nc_close(nc)

  df0 <- data.frame(lon=rep(lon,times=length(lat)),
                    lat=rep(lat,each=length(lon)),
                    sif=as.vector(sif),
                    ndvi=as.vector(ndvi),
                    year=as.numeric(substr(time,1,4)),
                    month=month.abb[as.numeric(substr(time,5,6))])


  df1 <- left_join(df0, dfW, by = c("lon", "lat")) %>%
    filter(sif!=-999,wat<0.2) %>%
    select(-wat)

}
```

And then we can apply this function on all files in the folder.

```
lf <- list.files(path=wpath,pattern=longname,full.names = T)
dfLIST <- lapply(X = lf,FUN = get_data)
df <- do.call(rbind,dfLIST)
```

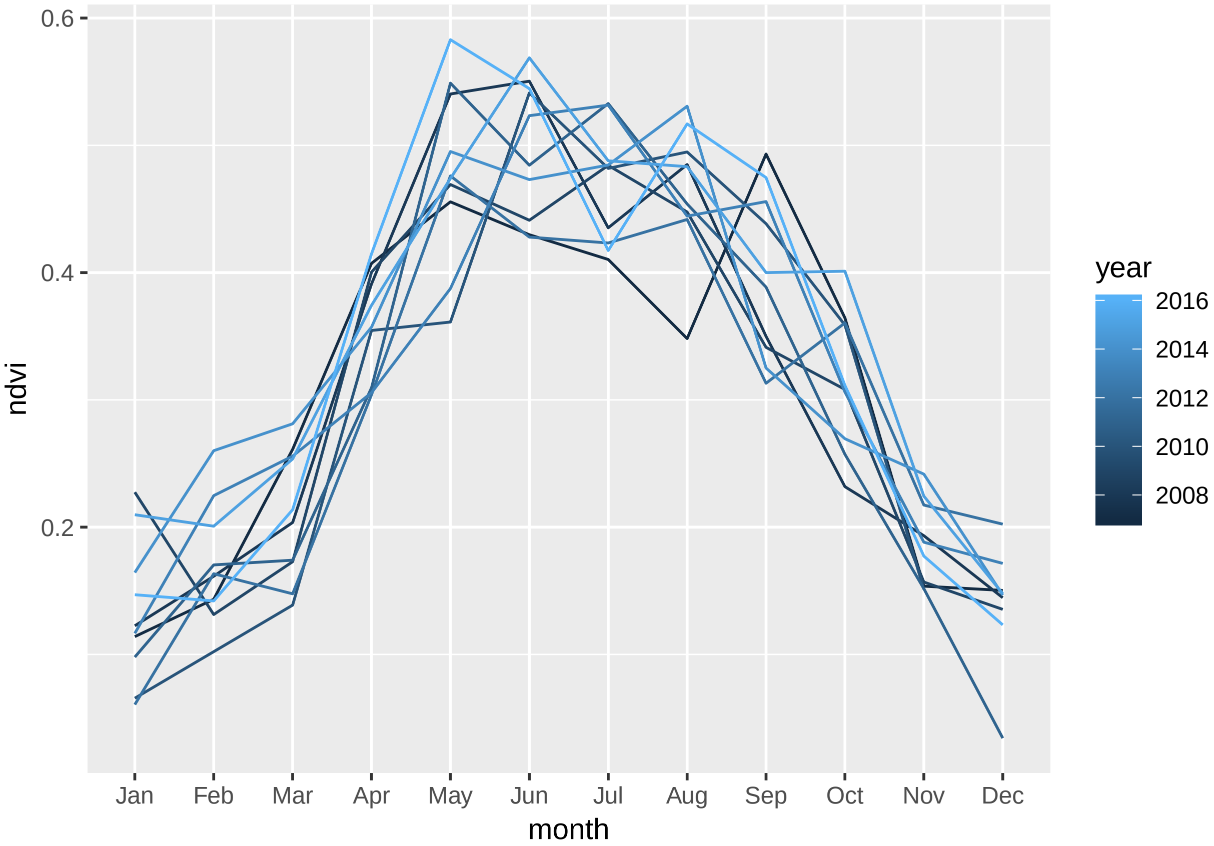
Explore data in the temporal dimension

First let us explore the data from a time series over single point. Using the `filter` function we can quickly extract the data by specifying the coordinates.

```
dfsub <- filter(df,lat==47.75, lon==19.25)
```

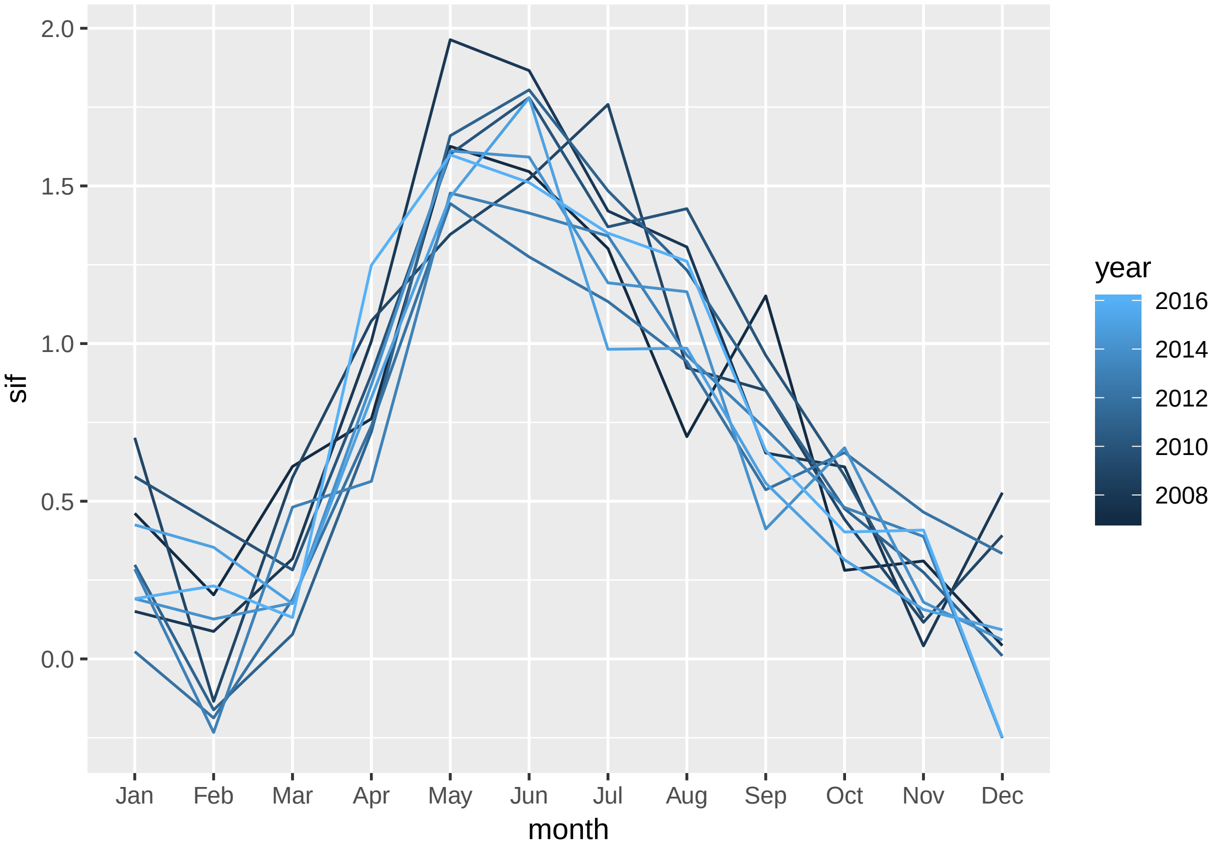
This is the plot for `ndvi`:

```
ggplot(dfsub)+geom_line(aes(x=month, y=ndvi, group=year,colour=year))
```



And this is the plot for `sif`:

```
ggplot(dfsub)+geom_line(aes(x=month, y=sif, group=year,colour=year))
```



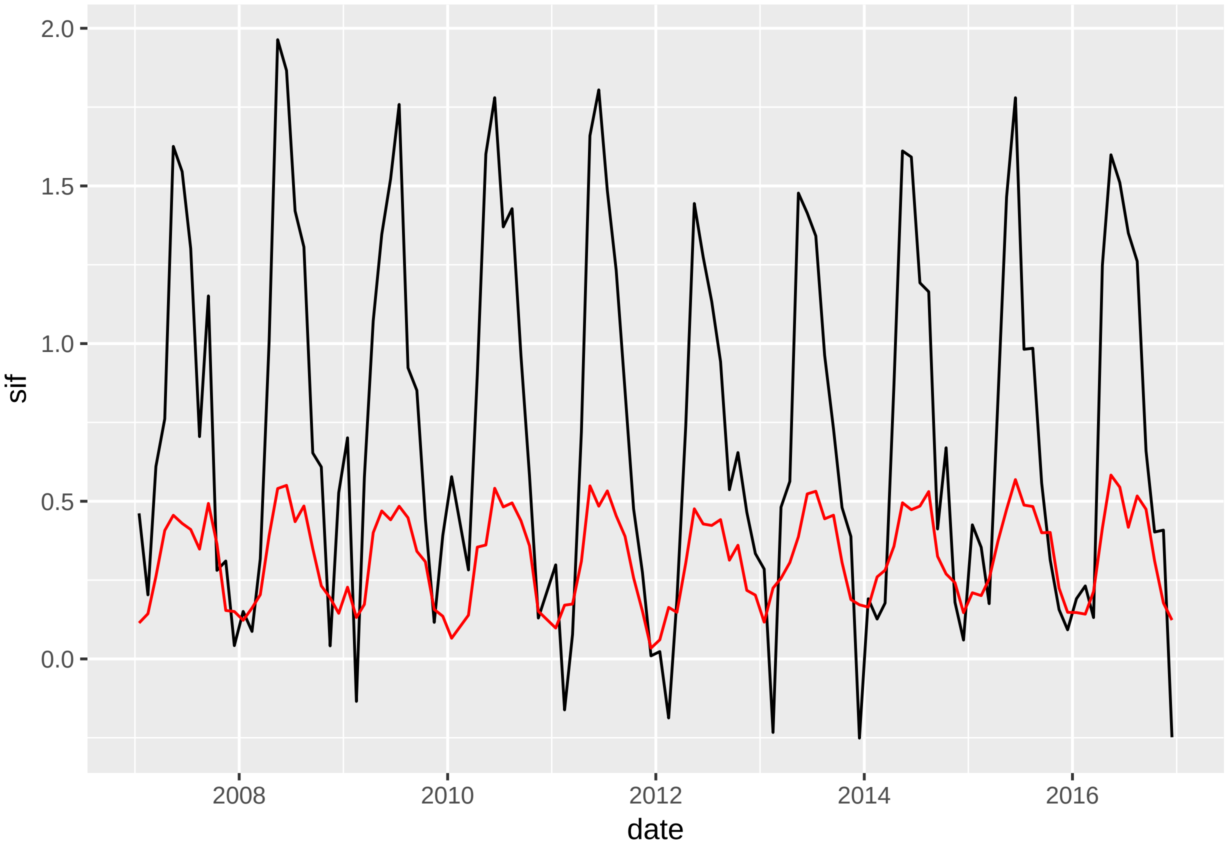
A first glance at the two plots already shows how these two indicators have a different shape. Indeed, they are not responding to the same physiological processes. NDVI reacts to how green the leaves are, while SIF is supposed to respond more directly to how productive these leaves are.

For some purposes it may be interesting to visualize this data in a sequence and next to each other (i.e. with NDIV and SIF overlapping). First, we need a continuous date field, so lets build one.

```
dfsub$date <- as.Date(paste(dfsub$year,dfsub$month,15), format="%Y %b %d")
```

Now we can plot both time series on on top of the other using two different line geometries:

```
ggplot(dfsub)+
  geom_line(aes(x=date, y=sif))+
  geom_line(aes(x=date, y=ndvi), colour='red')
```



However, NDVI and SIF are not in the same units, which explains the diffeences in amplitude. To compare these two signals in relative terms we can standardize them, that is, to remove their mean and divide by their standard deviation.

```
dfsub1 <- dfsub %>%
  mutate(z_sif=(sif-mean(sif,na.rm=T))/sd(sif,na.rm=T),
         z_ndvi=(ndvi-mean(ndvi,na.rm=T))/sd(ndvi,na.rm=T))
```

Now we can actaully gather the values in the same column using the following function:

```
require(tidyr)
```

```
## Loading required package: tidyr
```

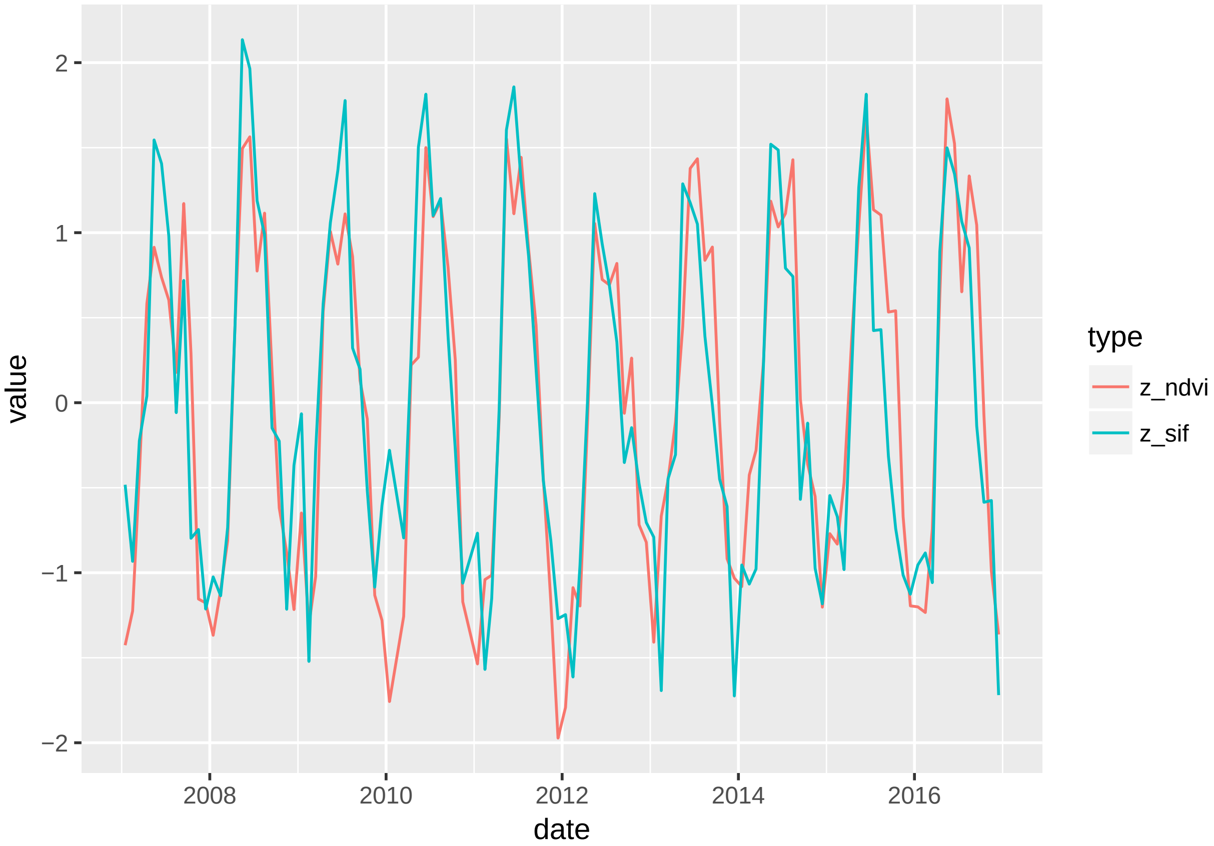
```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:raster':
##
##     extract
```

```
dfsub2 <- gather(dfsub1,type,value,8:9)
```

And plot using the colour aestheric to separate both signals:

```
ggplot(dfsub2)+
  geom_line(aes(x=date, y=value, colour=type))
```



We can now attack the entire dataset and try to build seasonal plots for the mean, max and min values of each signal.

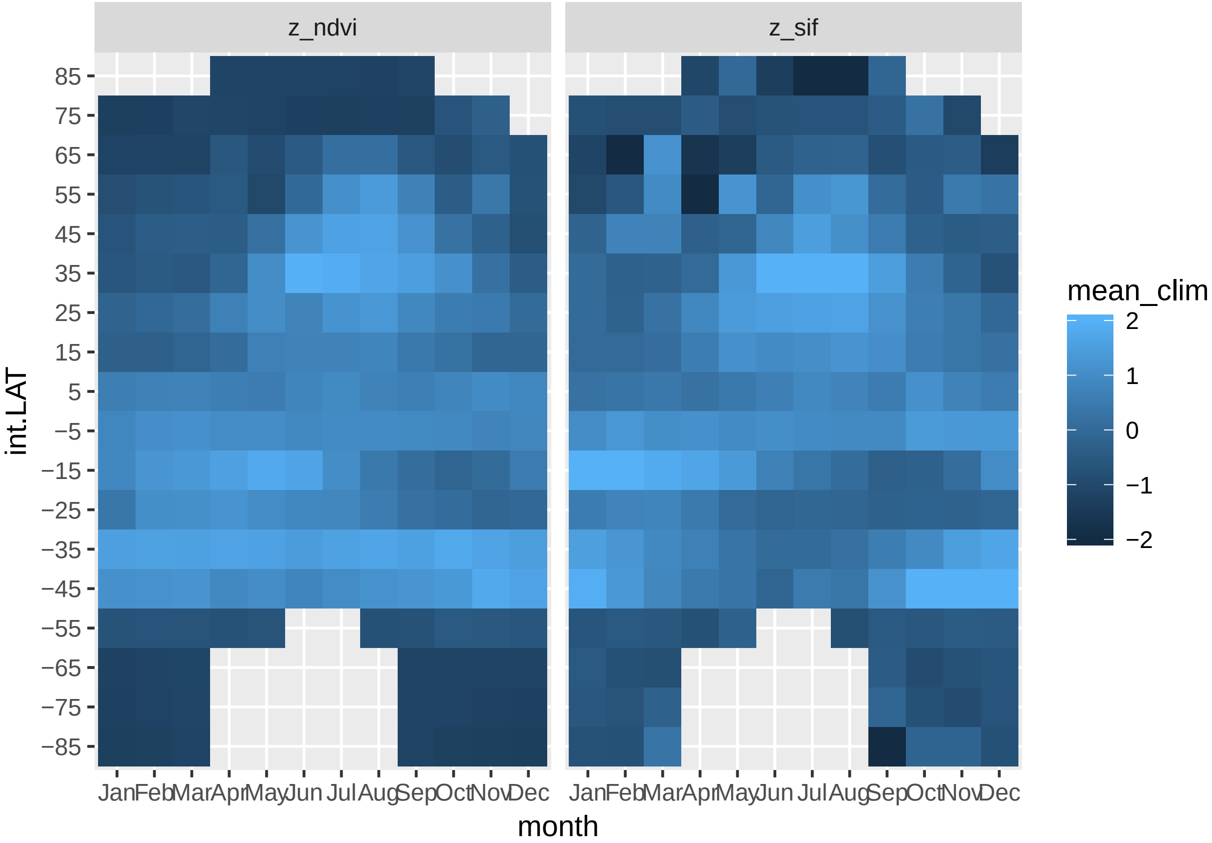
```
df1 <- df %>%
  mutate(data=as.Date(paste(year,month,15), format="%Y %b %d")) %>%
  mutate(z_sif=(sif-mean(sif,na.rm=T))/sd(sif,na.rm=T),
         z_ndvi=(ndvi-mean(ndvi,na.rm=T))/sd(ndvi,na.rm=T))

dfSum <- df1 %>%
  gather(type,value,8:9) %>%
  group_by(month,lat,lon,type) %>%
  summarize(mean_clim = mean(value,na.rm=T),
            max_clim = max(value,na.rm=T),
            min_clim = min(value,na.rm=T))
```

To help visualization we will define bin/intervals in the latitude scale:

```
brks.LAT <- seq(-90,90, by=10)
dfSum$int.LAT <- cut(dfSum$lat, brks.LAT, labels=brks.LAT[-1] - diff(brks.LAT)/2)

require(scales)
```

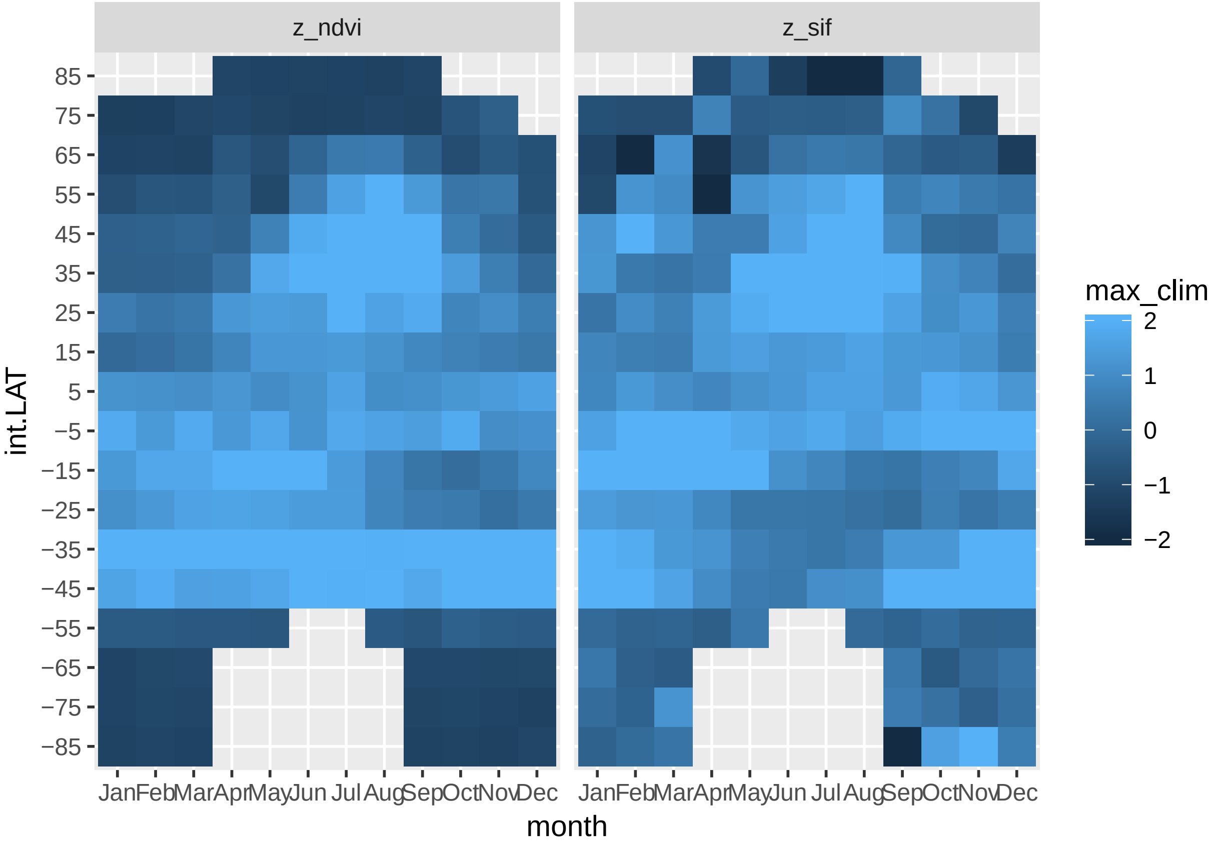
```
## Loading required package: scales
```

```
ggplot(dfSum) +
  geom_raster(aes(x=month,y=int.LAT,fill=mean_clim))+
  facet_wrap(~type) +
  scale_fill_continuous(limits=c(-2,2), oob=squish)
```

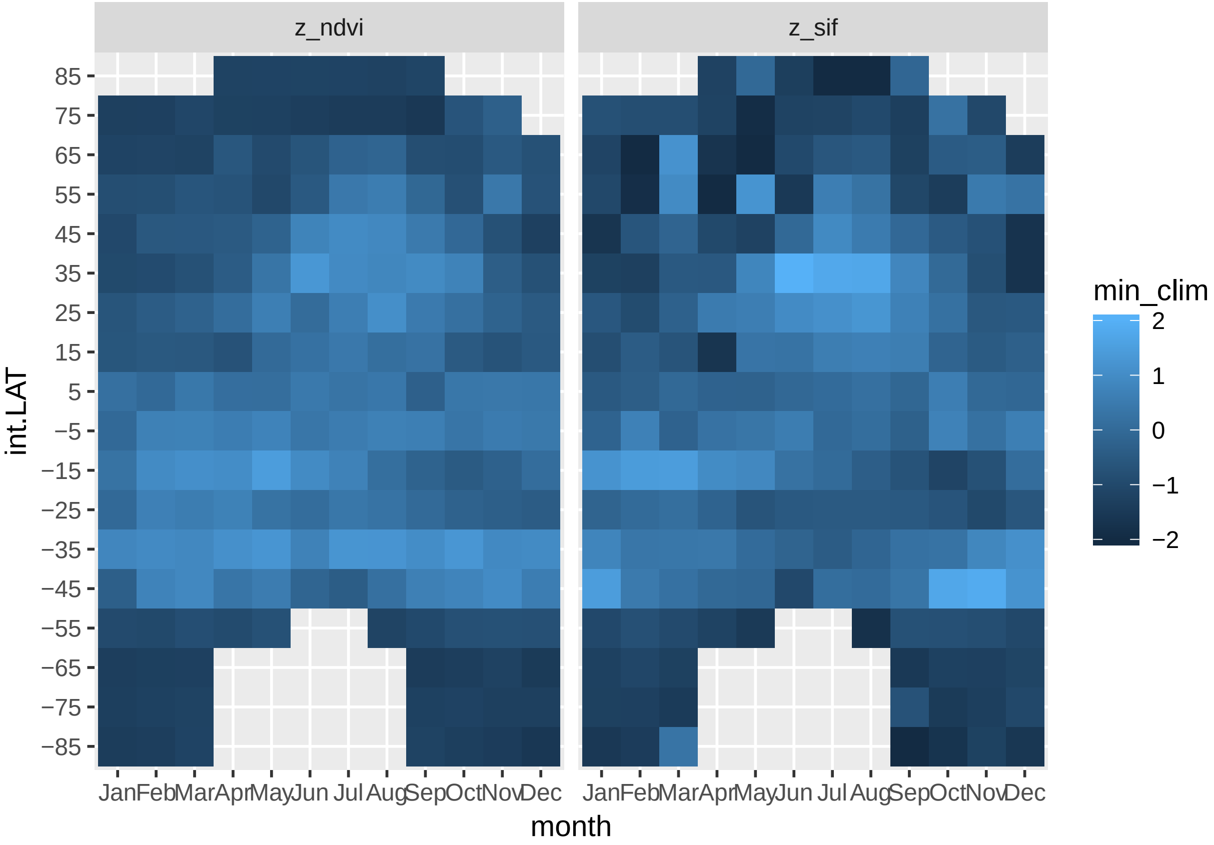


The mean standardized signal is quite similar, but how about the max and the min:

```
ggplot(dfSum) +
  geom_raster(aes(x=month,y=int.LAT,fill=max_clim))+
  facet_wrap(~type) +
  scale_fill_continuous(limits=c(-2,2), oob=squish)
```



```
ggplot(dfSum) +
  geom_raster(aes(x=month,y=int.LAT,fill=min_clim))+
  facet_wrap(~type) +
  scale_fill_continuous(limits=c(-2,2), oob=squish)
```



Mapping temporal correlation and agreeement

Another technique to explore the two datasets involves mapping the temporal correlation, or Pearson coefficient of correlation (ρ), between both time series at each pixel. This can be easily calculated as follows:

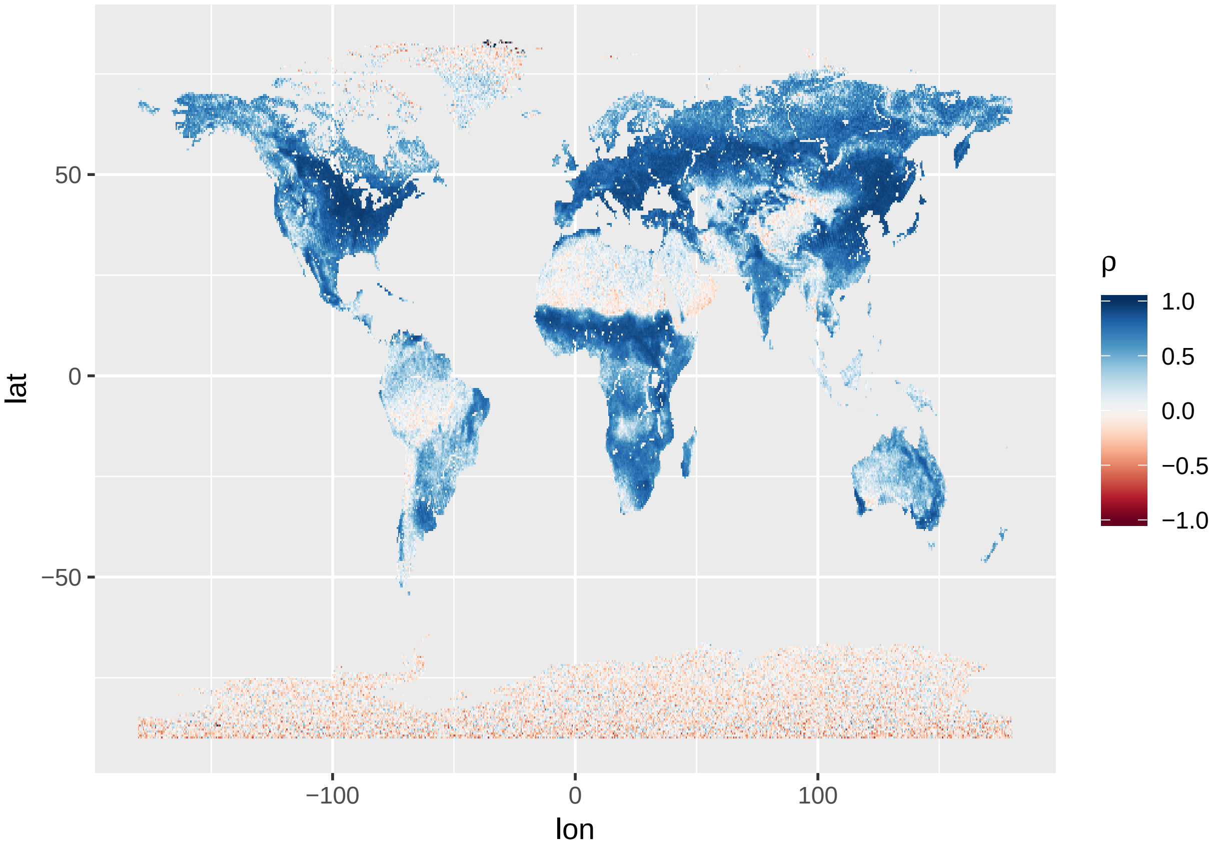
```
dfSpCorr <- df %>%
  group_by(lat,lon) %>%
  summarize(corr=cor(sif,ndvi))
```

And the resulting plot is:

```
require(RColorBrewer)
```

```
## Loading required package: RColorBrewer
```

```
col.pal=brewer.pal(11,'RdBu')
ggplot(dfSpCorr) +
  geom_raster(aes(x=lon,y=lat,fill=corr)) +
    scale_fill_gradientn(bquote(rho),limits=c(-1,1), colours = col.pal, oob=squish)
```



We can see in this map that many areas have high correlation between both time series. These areas are typically those with vegetated landscapes, as should be expected, as there greenness and GPP should be related. Areas of negative correlation exist in deserts.

Correlation conveniently ignores any bias between the two signals and hence the difference in units is irrelevant. Sometimes it is necessary to measure agreement of datasets, that is, having an idea beyond correlation if there are no additive or multiplicative bias in the data.

In order to assess the agreement between dataset Y and dataset X, we can use an index of agreement (λ) that varies from 0, indicating *no agreement* to 1 *full agreement*. It has the advantage of being directly related to *rho*, but it additionally indicates if there is a bias (either additive or multiplicative). For more info, see the paper Duveiller et al. 2016. It is defined as:

$$\lambda = 1 - \frac{n^{-1}\sum_{i=1}^{n}(X_i - Y_i)^2}{\sigma_X^2 + \sigma_Y^2 + (\bar{X} + \bar{Y})^2 + \kappa}$$

where $\kappa = 0$ if $\rho > 0$ and $\kappa = 2|\sum_{i=1}^{n}(X - \bar{X})(Y - \bar{Y})|$ otherwise. The logic between this is index is that the numerator represents the root mean squared deviation (RMSD) and the denominator is the maximal deviation the dataset can take. In a way, it combines the info of the bias, the RMSD and ρ in a single metric. This index of agreement also has the advantage of being symmetric (i.e. X or Y are exchangeable in the definition above, which in turns means neither is *a priori* considered as more of a reference that the other).

First let us write a function for λ:

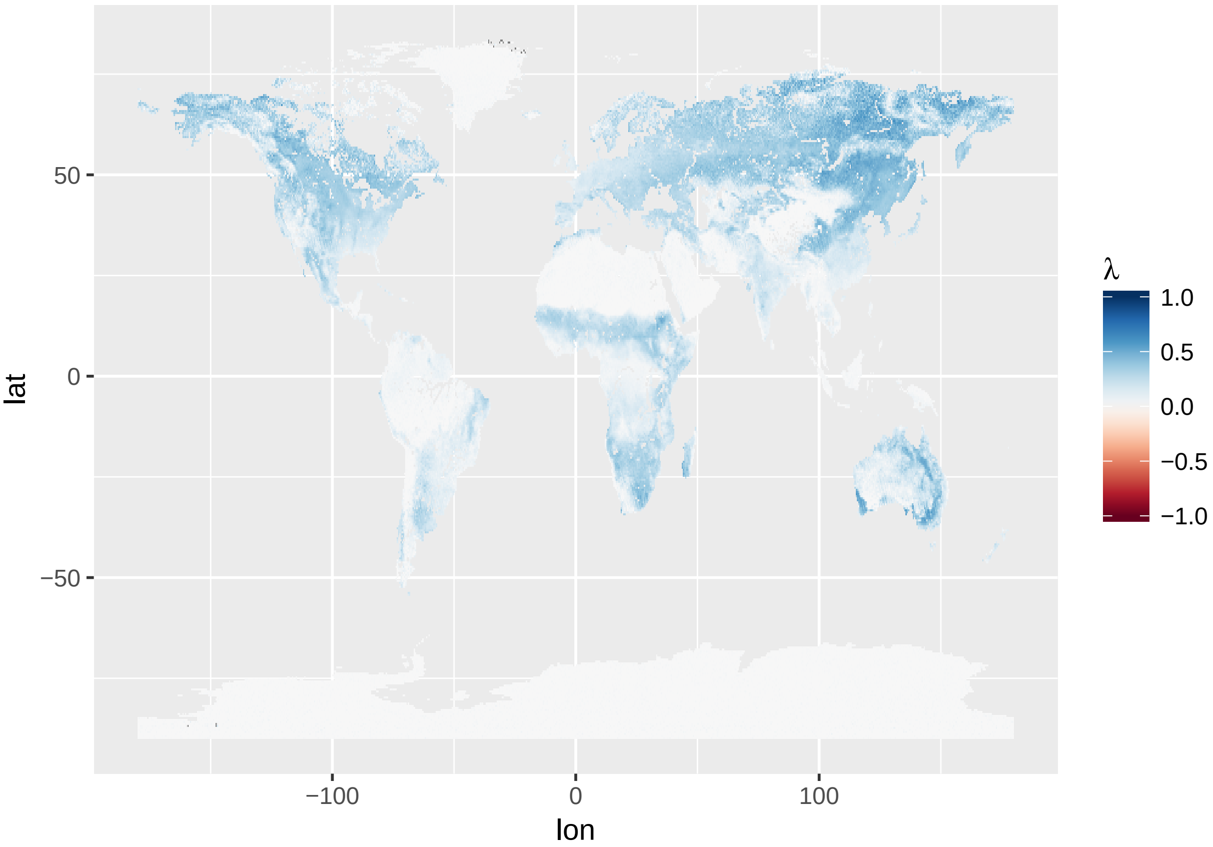
```
Lambda <- function(x,y) {
  if(length(x)<=1){return(NA)}
  Xm <- mean(x)
  Ym <- mean(y)
  n <- length(x)
  K <- sign(cor(x,y)<0) * 2*abs(sum((x-Xm)*(y-Ym)))
  N <- sum((x-y)^2)
  D <- sum((x-Xm)^2) + sum((y-Ym)^2) + n*(Xm - Ym)^2 + K
  Index <- 1 - N/D
  return(Index)
}
```

And we can calculate it for all time series of both the standardized and raw signals.

```
dfSpCorr1 <- df1 %>%
  group_by(lat,lon) %>%
  summarize(IoAg=Lambda(sif,ndvi),
            IoAg_z=Lambda(z_sif,z_ndvi),
            corr_z=cor(z_sif,z_ndvi))
```

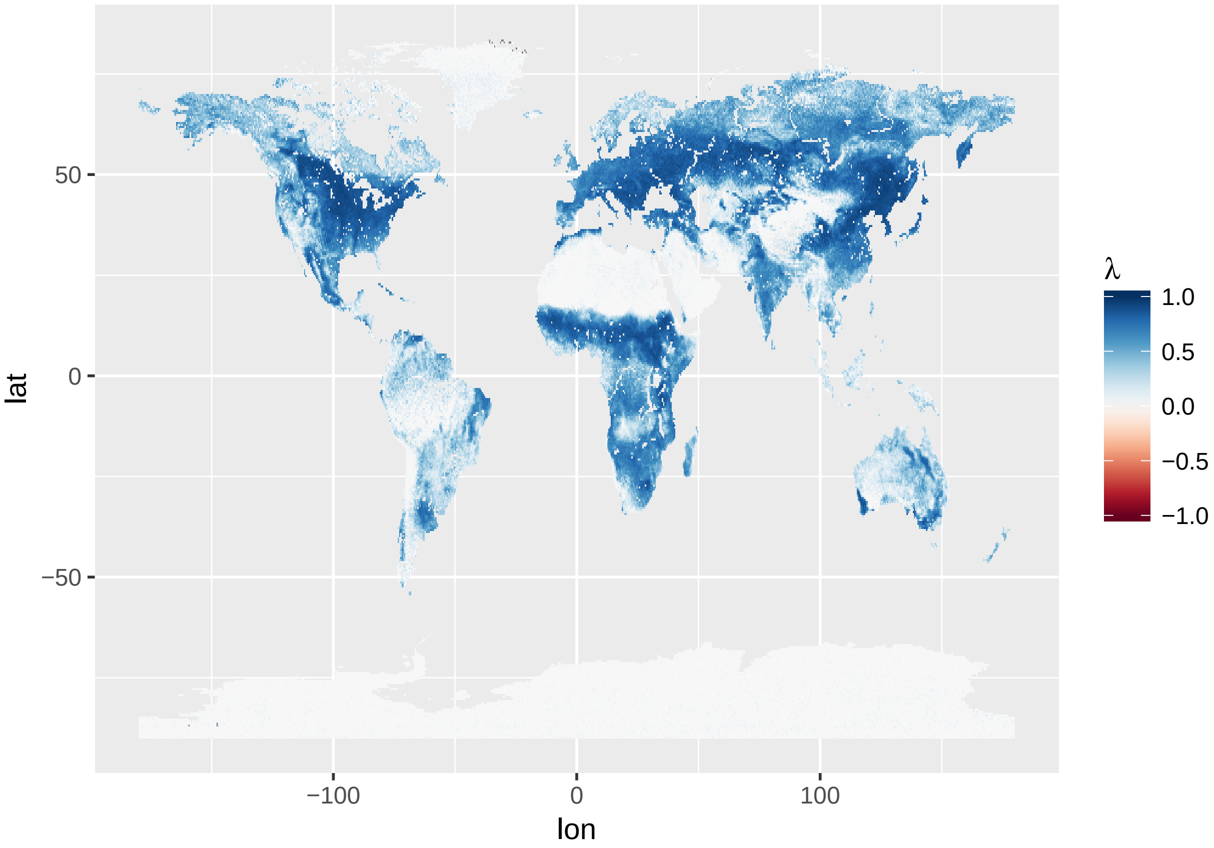
The following plot shows how the agreement of the raw (unstandardized) signals, plotted in the same colour scale as the correlation. The following values are much lower because the units of SIF and NDIV are still different.

```
ggplot(dfSpCorr1) +
  geom_raster(aes(x=lon,y=lat,fill=IoAg)) +
    scale_fill_gradientn(bquote(lambda),limits=c(-1,1), colours = col.pal, oob=squish)
```



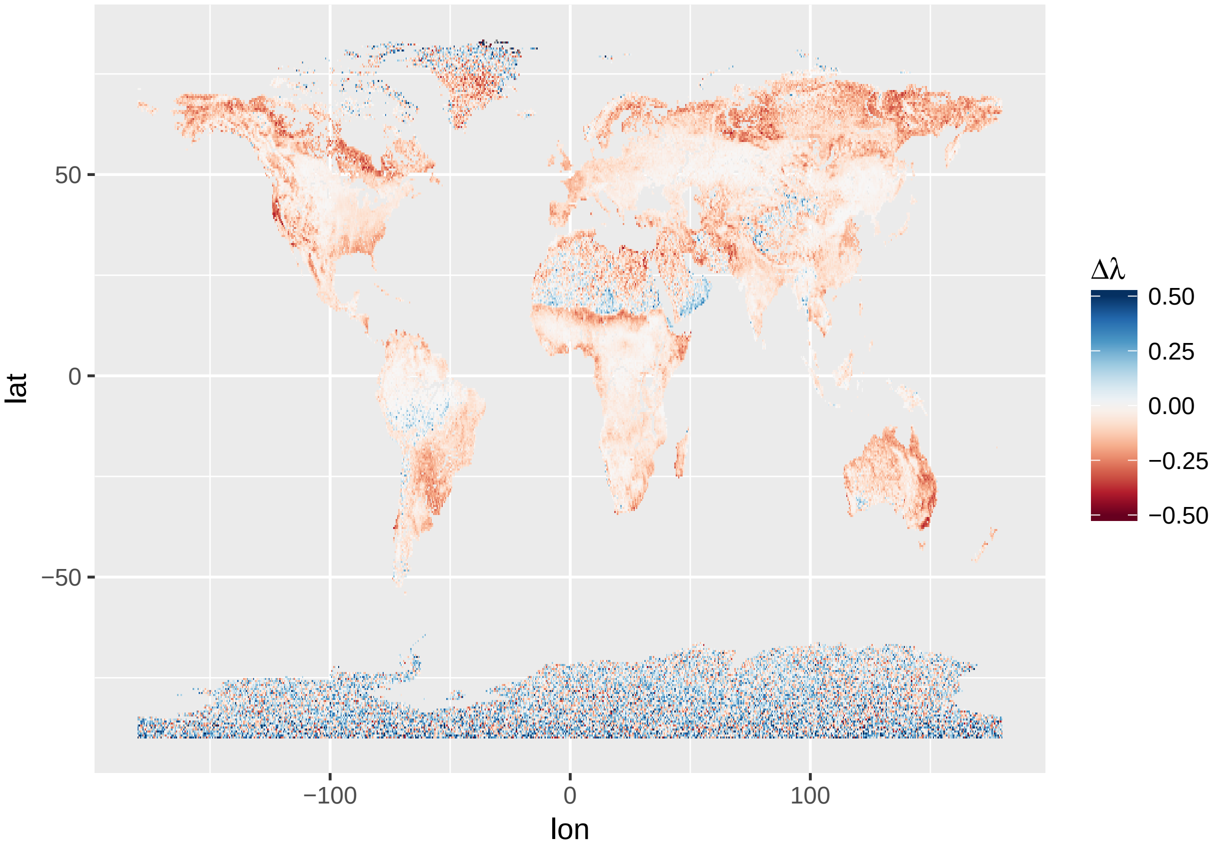
After standardization, the agreement increases:

```
ggplot(dfSpCorr1) +
  geom_raster(aes(x=lon,y=lat,fill=IoAg_z)) +
    scale_fill_gradientn(bquote(lambda),limits=c(-1,1), colours = col.pal, oob=squish)
```



But it is different from the correlation, as shown in this difference plot:

```
ggplot(dfSpCorr1) +
  geom_raster(aes(x=lon,y=lat,fill=IoAg_z-corr_z)) +
    scale_fill_gradientn(bquote(paste(Delta,lambda)),limits=c(-0.5,0.5), colours = col.pal, oob=squish)
```



Summarizing data in climate space

In climate science, it is often helpful to visualize data beyond the standard Cartesian space in what is known as *climate space*. This space is defined using climate variables in the y and x axes instead of latitude and longitude. Some useful variables are mean temperature and accumulated precipitation.

To plot our data in climate space we will use climate data from the CRU. It can be obtained here:http://dx.doi.org/10.5072/edf8febfdaad48abb2cbaf7d7e846a86. For this example we will only use mean data over the 2001 to 2010 period (even if these are not exactly synchronous with the GOME-2 data).

```
nc <- nc_open(paste0(wpath,'cru_ts4.00.2001.2010.pre.dat.nc'))
pre <- ncvar_get(nc,varid = 'pre')
lon <- ncvar_get(nc,varid = 'lon')
lat <- ncvar_get(nc,varid = 'lat')
nc_close(nc)
nc <- nc_open(paste0(wpath,'cru_ts4.00.2001.2010.tmp.dat.nc'))
tmp <- ncvar_get(nc,varid = 'tmp')
nc_close(nc)
```

We then calculate and gather the mean temperature `tmp` and accumulated precipitation `pre` in a new dataframe `dfClim`.

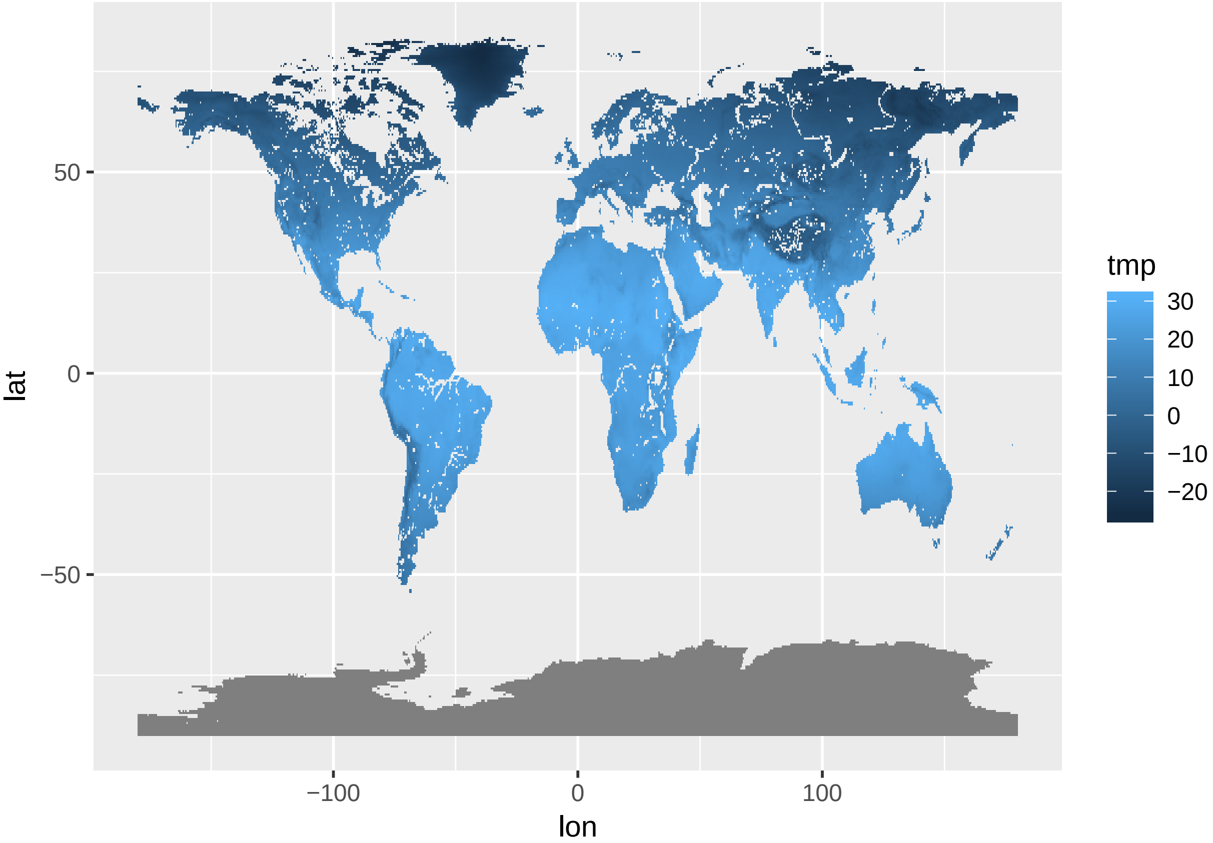
```
dfClim <- data.frame(lon=rep(lon,times=length(lat)),
                     lat=rep(lat,each=length(lon)),
                     tmp=as.vector(apply(tmp,MARGIN=c(1,2), FUN=mean,na.rm=T)),
                     pre=as.vector(apply(pre,MARGIN=c(1,2), FUN=sum,na.rm=T))/10)
```

We can join `dfClim` to the previous one with the mean seasonal values `dfSum`

```
dfc <- left_join(dfSum,dfClim,by=c('lon','lat'))
```

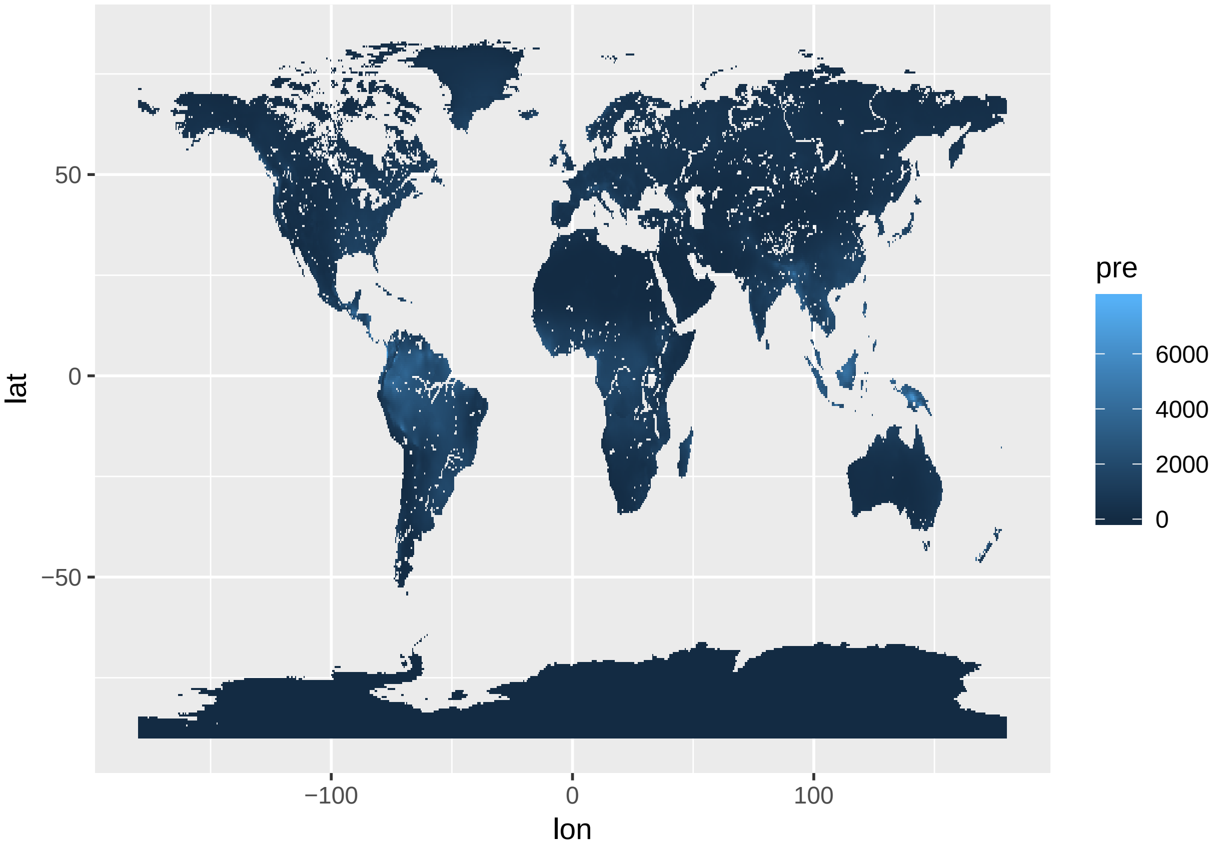
The result can be plotted to control that everything is alright. This is a map of the mean temperature:

```
ggplot(dfc)+geom_raster(aes(x=lon,y=lat,fill=tmp))
```



And the accumulated precipitation:

```
ggplot(dfc)+geom_raster(aes(x=lon,y=lat,fill=pre))
```

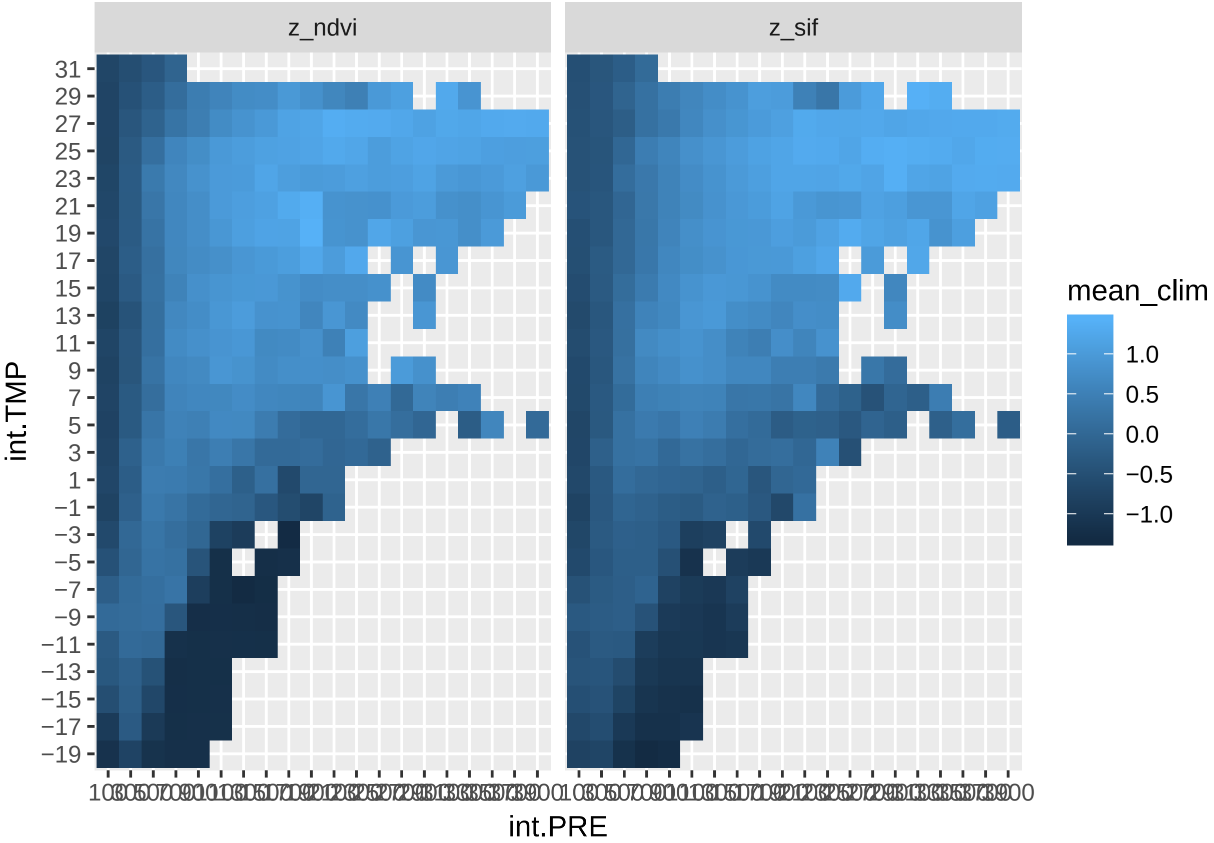


We can define intervals or bins for the climate variables to facilitate visualization, as we previously did for the latitude classes.

```
brks.TMP <- seq(-20,32, by=2) #; lims.TMP <- c(-20,32)
brks.PRE <- seq(0,4000, by=200)
dfc$int.TMP <- cut(dfc$tmp, brks.TMP, labels=brks.TMP[-1] - diff(brks.TMP)/2)
dfc$int.PRE <- cut(dfc$pre, brks.PRE, labels=brks.PRE[-1] - diff(brks.PRE)/2)
```

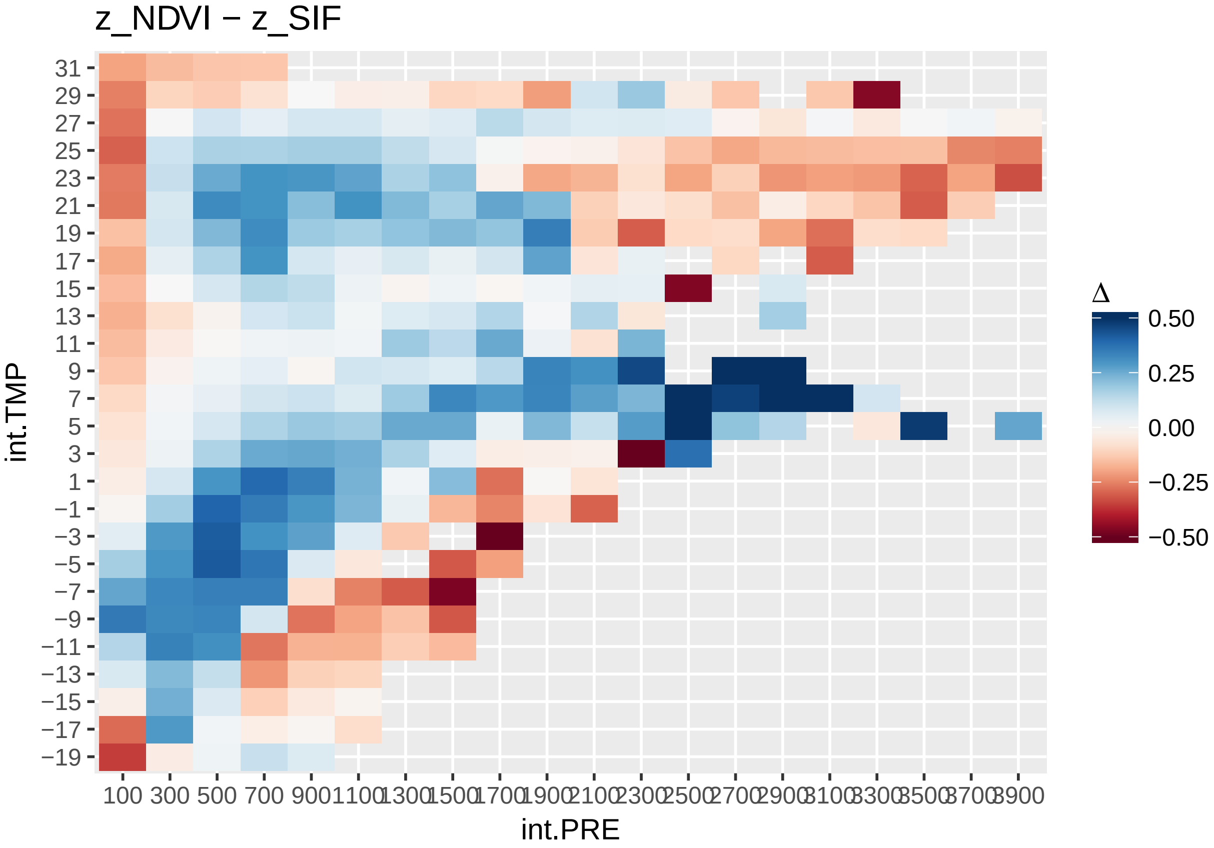
Now we summarize the mean climatogical value of the standardized index for each bin in the climate space and make a plot.

```
dfc2 <- dfc %>%
  group_by(int.TMP,int.PRE,type) %>%
  summarize(mean_clim = mean(mean_clim,na.rm=T)) %>%
  filter(!is.na(int.TMP),!is.na(int.PRE))

ggplot(dfc2)+
  geom_raster(aes(x=int.PRE,y=int.TMP,fill=mean_clim)) +
  facet_wrap(~type)
```



The results look very similar. To spot the differences it is often better to calculate a difference. For that we need to spread the two variables back into separate columns and then plot the differences.

```
dfc3 <- spread(dfc2,type,mean_clim)

ggplot(dfc3)+geom_raster(aes(x=int.PRE,y=int.TMP,fill=z_ndvi-z_sif))+
  scale_fill_gradientn(bquote(Delta),limits=c(-0.5,0.5), colours = col.pal, oob=squish) +
  ggtitle('z_NDVI - z_SIF')
```



The output shows how the largest positive difference (i.e. std. NDVI larger than std. SIF) are in humid and temperate areas, while the largest negative differences are in cold and warm regions, predominantly those which are humid. Dryer areas show less marked differences, but extremes generally have higher values of std. SIF.