# Land cover models

Bleaching in the Maldives

*Prof. Dr. Loic Pellissier*

*20 November, 2017*

## Contents

## 1 Introduction

The goal of this practical is to explore the use of satellite reflectance data in order to model land use. Specifically we will try and detect coral bleaching zones in the Maldives due to the bleaching event of 2016.

The basic idea is that a bleeching event at a certain site will increase the surface reflectance at this site. Apart from the health of the coral, there are many other factors that effect estimated surface reflectance at coral sites such as coral heterogeneity, water depth, tide, seaweed, river discharge and atmospheric conditions (Yamano and Tamura (2004)). Because we do not have information for these factors our approach will be to assume they are relatively stable by site. The methodology we will use will focus on detecting change to the normal cyclical regime of surface reflectance at a given site. We will then observe if there are any global spatial patterns to these changes that indicate the possibility of a global intervention such as a bleaching event.

We will use the programming language in R and reflectance data from the Landsat satellite. The following steps will be followed:

1. Download Top-of-Atmoshpere (TOA) reflectance and surface reflectance data for the chosen study area and study period.
2. Detect cloudy pixels by processing TOA reflectance data with FMASK algorithm in Matlab.
3. Load data into R and build a historical pixelwise database of reflectance that we can easily exploit.
4. Model the surface reflectance of each pixel as a function of the time of year and the tidal cycles. Use forward selection to choose the relevant seasonal and tidal factors.
5. Using surface reflectance model apply change point analysis independently to each pixel to determine the most likely date of a change occurring at the site of each pixel.
6. Determine the most likely date of bleaching event affecting the entire study area.
7. Produce a map of likely bleached areas within study area.

## 2 Landsat data

The Landsat program is a long-running satellite imagery acquisition project. The first satellite Landsat 1 of the programme was launched on July 23, 1972. The most recent, Landsat 8, was launched on February 11, 2013. Landsat 1 provided information for four spectral band with spatial resolutions of 80 metres . Landsat 8 data has eleven spectral bands with spatial resolutions ranging from 15 to 100 meters; the temporal resolution is 16 days. Landsat images are usually divided into scenes for easy downloading. Each Landsat scene is about 185 kilometers long and 185 kilometers wide although since scenes may be displayed obliqueley the image is padded with black pixels such that it a larger area.

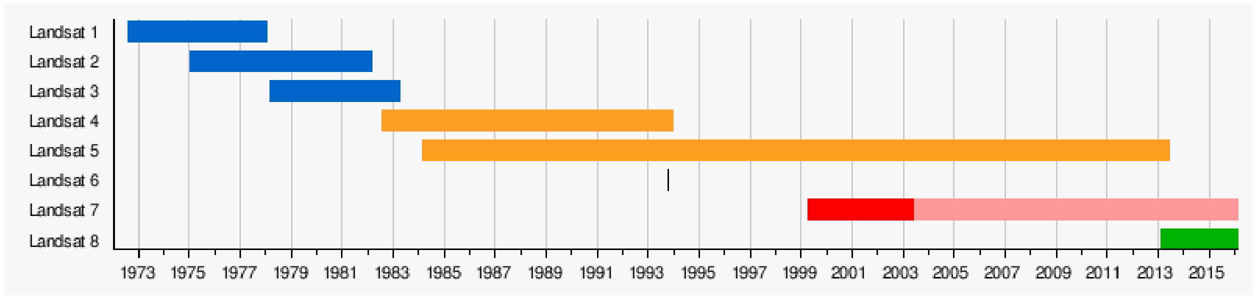


Figure 1: Landsat program timeline

The landsat 8 satellite has a sun-synchronous near-circular, near-polar, orbit with a 705 km altitude at the equator. Sun-synchronicity means it orbits the earth in such a way that it always passes over a given point of the planet???s surface at the same local solar time. Although, to be clear, it does not pass over all points at the same local solar time. This is useful since it allows for relatively constant illumination of a given point accross different time observations.

The landsat completely orbits the Earth every 98.9 minutes, completes over 14 orbits per day, and provides complete coverage of the Earth every 16 days. It has an equatorial crossing at 10:11 a.m. (+/???15 min) mean local time during the descending node (daytime). In this orbit, the Landsat 8 observatory follows a sequence of fixed ground tracks (also known as paths) defined by the second Worldwide Reference System (WRS-2). WRS-2 is a path/row coordinate system used to catalog all the science image data acquired from the Landsat 4 - 8 satellites. The following animation shows how the landsat 8 covers every part of the globe twice (once in daytime during the descending node and once in nighttime during the ascending node) by capturing 124 scenes daily and nightly along 14-15 paths.

```
# We first set the R working directory
setwd("/Volumes/Local/emilianodiazsalasp/Documents/ETH Zurich/USYS/scripts/bleachingMaldives/R")
```

When loading data we use routes with respect to the R working directory and with respect to the following folder structure.

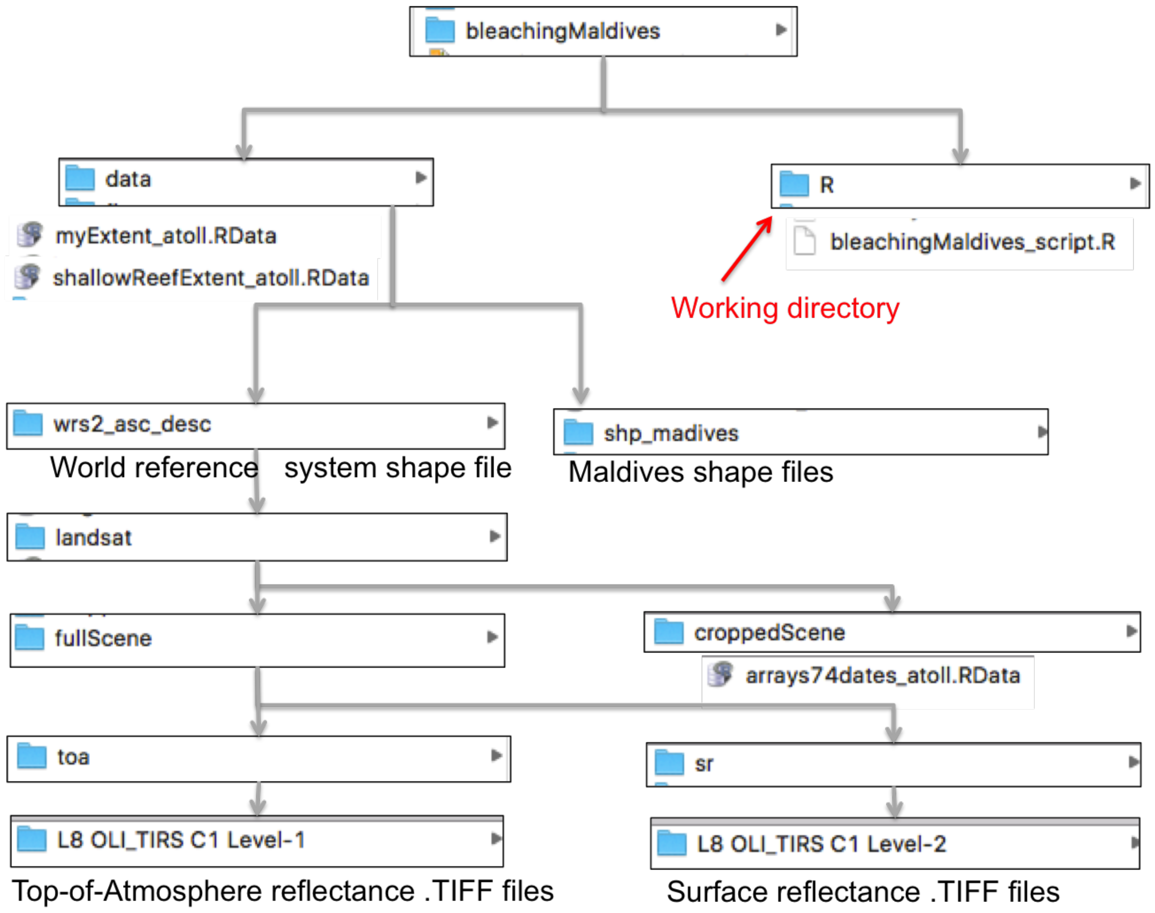

bleachingMaldives
data
myExtent_atoll.RData
shallowReefExtent_atoll.RData
R
bleachingMaldives_script.R
Working directory
wrs2_asc_desc
World reference system shape file
shp_madives
Maldives shape files
landsat
fullScene
croppedScene
arrays74dates_atoll.RData
toa
sr
L8 OLI_TIRS C1 Level-1
L8 OLI_TIRS C1 Level-2
Top-of-Atmosphere reflectance .TIFF files
Surface reflectance .TIFF files

Figure 2: Required folder structure

```
library(rgdal)
# reference system used by landsat
projLandsat <- "+proj=utm +zone=43 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
library(raster)
# Load World Reference System (2) shape file where landsat
# scenes are shown, identified by their path and row
wrs2 <- readOGR(dsn = "../data/wrs2_asc_desc", layer = "wrs2_asc_desc")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/wrs2_asc_desc", layer: "wrs2_asc_desc"
## with 57784 features
## It has 13 fields
## Integer64 fields read as strings:  PR_ PR_ID WRSPR
```

```
wrs2 <- wrs2[order(wrs2$DAYCLASS, wrs2$PATH, wrs2$ROW), ]
projection(wrs2)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
table(table(wrs2$PATH))  #248  scenes per path
```

```
##
## 248
## 233
```

```
table(table(wrs2$PATH, wrs2$MODE))  #124 daytime and nightime scenes per path
```

```
##
## 124
## 466
```

```
apply(as.array(table(wrs2$PATH, wrs2$DAYCLASS, wrs2$MODE)), c(2,
    3), function(col) sum(col > 0))  #14-15 paths per day
```

```
##
##       A  D
##    1  14 14
##    2  15 15
##    3  14 14
##    4  15 15
##    5  15 15
##    6  14 14
##    7  15 15
##    8  14 14
##    9  15 15
##    10 14 14
##    11 15 15
##    12 14 14
##    13 15 15
##    14 15 15
##    15 14 14
##    16 15 15
```

```
124 * 16 * 14.5 * 2  #approx.:  124 daytime scenes/path 14.5 paths/day * 16 days/cycle
```

```
## [1] 57536
```

```
# * 2 daytime & nightime scenes / daytime scenes = 57,536
# daytime & nightime scenes/cycle
nrow(wrs2)  #57,784
```

```
## [1] 57784
```

Figure 3: Animation of coverage of globe by landsat 8

We now show which landsat scenes provide daytime coverage of Switzerland.

```r
ch <- getData("GADM", country = "CHE", level = 0)
library(rgeos)
indx <- which(gIntersects(wrs2, ch, byid = T)[1, ])
wrs2.ch <- wrs2[indx, ]
wrs2.ch <- wrs2.ch[which(wrs2.ch$MODE == "D"), ]
plot(wrs2.ch, border = seq(length(unique(wrs2.ch$PATH)))[match(wrs2.ch$PATH,
    unique(wrs2.ch$PATH))])
plot(ch, bg = "light grey", add = T)
text(gCentroid(wrs2.ch, byid = T), labels = paste(wrs2.ch$PATH,
    wrs2.ch$ROW, sep = "-"), col = "purple", cex = 1)
```

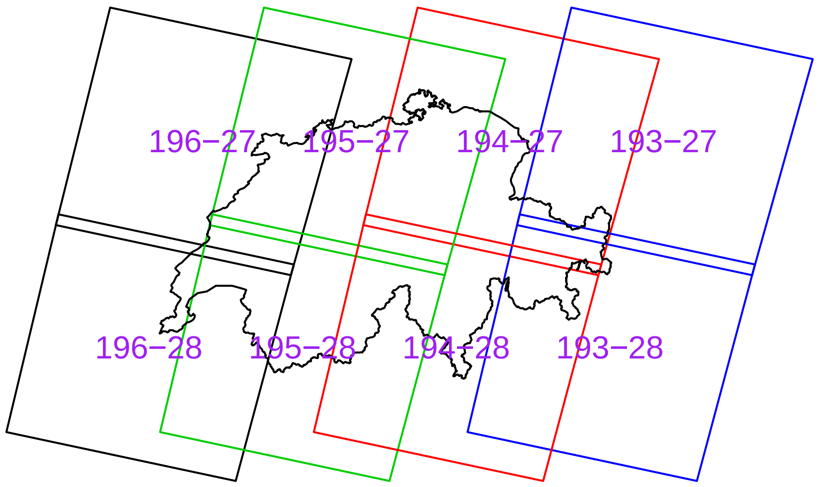


Figure 4: Map of Switzerland with Landsat scenes.

There are two sensors aboard landsat 8, the Operational Land Imager (OLI) recording radiances with frequencies between 0.43 and 1.38 micrometers in 9 bands, at resolutions of 15-30m, and the Thermal Infrared Sensor (TIRS) recording radiances with frequencies between 10.6 and 12.51 micrometers in 2 bands, at 100m resolution.

Both the OLI and TIRS sensors simultaneously image every scene, but are capable of independent use should a problem in either sensor arise. In normal operation the sensors view the Earth at nadir (directly overhead, view zenith angle=0), but special collections may be scheduled off-nadir.

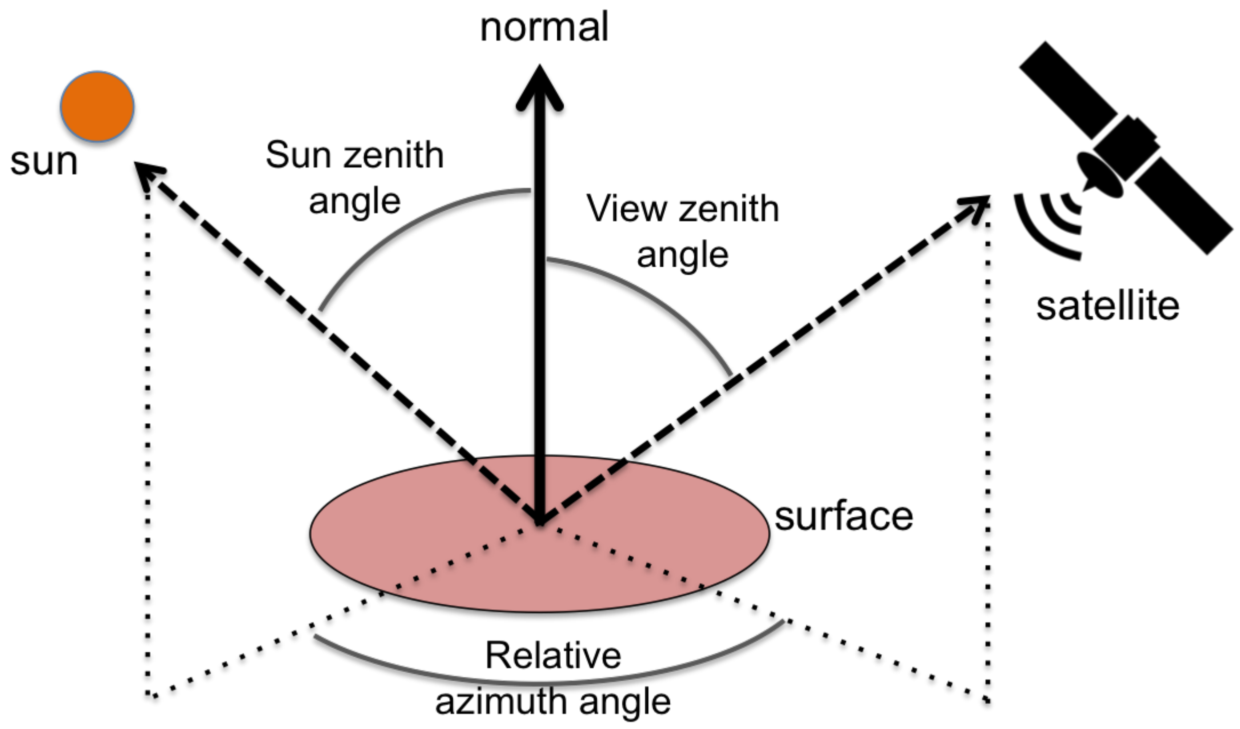


Figure 5: Sun sensor geometry

The TIRS and OLI collect level 0 brightness data , store it using the WRS2 coordinate system and send it to the EROS Center via a ground network consisting of stations around the world. It is then geometrically and radiometrically calibrated to turn it into the Level-1 absolute radiance data that is available for download. Geometric calibration involves linking radiance data to a geographic system in the most accurate way possible. Radiometric calibration involves performing operations on the radiance data itslef to correct for measurement inaccuracies. This data has been converted to absolute radiance so is unitless (16-bit Digital Numbers or DNs).

The collection number associated to Level-1 landsat data refers to how the data was collected in order to characterize its quality. So far there are only two collection numbers: *pre-collection* which is not *tiered* according to quality and collection 1 which is. The collection category or *tier*, refers to the quality of the parameters used to carry out geometric and radiometric correction. There are three tiers: tier 1, tier 2, real time in order of quality.

Level-1 absolute radiance data (in DNs) can be converted to spectral radiance using scaling factors (*gain* and *bias* associated to each scene) or to top-of-atmosphere reflectence using scaling factors (*gain* and *bias* associated to each scene) and the solar elevation/zenith angle (associated to each pixel).

To obtain Level-2 surface reflectance data, Level-1 top-of-atmosphere reflectance must be atmospherically corrected. Ground level surface reflectance is estimated by correcting for atmospheric scattering and absorption. Correction involves many different variables including atmospheric intrinsic reflectance, gaseous transmission, atmospheric transmission, spherical albedo, surface pressure, ozone, water vapor and aerosol optical thickness.

More information on landsat 8 data can be found at https://landsat.usgs.gov/landsat-8-l8-data-users-handbook and https://landsat.usgs.gov/frequently-asked.

## 2.1 Description

### 2.1.1 Study area

Landsat reflectance data are downloaded by *scene* which in the case of the Maldives extend a 230km$^2$ area (including padding pixels) . Since the resolution of the images is 30m this means each scene has around $(230000/30)^2 \approx 60$ million pixels. We will use 74 landsat images from the 2013-05 to 2017-05 period. For each pixel we have one 16 bit integer reflectance value for each of nine bands (wavelength intervals) plus a 16 bit integer value for the quality band. In other words for each pixel and date we will have a vector of ten 16 bit integer values. For 74 dates and 4 bytes per integer value this means we have 74 dates $*$ 60 million pixels/date $*$ 10 integers/pixel $*$ 4 bytes/integer $\approx$ 177 gigabytes. For data storage and processing restrictions we chose a subarea of approximately 5.36 km$^2$ such that the amount of data will be approximately $5.36^2/230^2 * 177000 \approx = 96$ megabytes.

In this subsection we explore the chosen 5.36km$^2$ area in the context of the wider maldives area and the landsat scenes.

```
# reference system used by landsat
projLandsat <- "+proj=utm +zone=43 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
# Load maldives shape file using readOGR function from rgdal
# package
maldives.shp <- readOGR(dsn = "../data/shp_madives", layer = "Maldive_inter")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/shp_madives", layer: "Maldive_inter"
## with 5306 features
## It has 25 fields
```

```
projection(maldives.shp)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
# Keep only paths 144-146 and row 55-60 corresponding to the
# maldives scenes
wrs2 <- wrs2[which(wrs2$PATH %in% seq(144, 146) & wrs2$ROW %in%
    seq(55, 60)), ]
wrs2 <- wrs2[which(wrs2$MODE == "D"), ]  #descending corresponds to daytime
dim(maldives.shp)  # 5306 polygons
```

```
## [1] 5306   25
```

```
# The maldives shape file includes a classification of the
# maldives according to the type of atoll and reef of each
# polygon. Show the distribution of atoll type and reef type
# for the maldives.
table(maldives.shp$RB_ATTRIB)/nrow(maldives.shp) * 100
```

```
##
## barrier atoll-bank   patch atoll-bank
##           50.88579           49.11421
```

```
table(maldives.shp$RB_DEPTH_A)/nrow(maldives.shp) * 100
```

```
##
##           deep_reef        shallow_reef variable_depth_reef
##            1.356954           81.134565           17.508481
```

```
# Load 2 objects of type extent which have coordinates of the
# chosen study area Load first extent - first zoom
load(file = "../data/shallowReefExtent_atoll.RData")
# Transform into a SpatialPolygons object in order to plot
# and calculate area
(ext <- shallowReefExtent)
```

```
## class       : Extent
## xmin        : 73.30199
## xmax        : 73.59743
## ymin        : 3.780065
## ymax        : 4.142287
```

```
pts <- expand.grid(x = ext[1:2], y = ext[3:4])
pts <- pts[c(1, 2, 4, 3, 1), ]
P1 <- Polygon(pts)
Ps1 <- list(Polygons(list(P1), ID = 1))
rect <- SpatialPolygons(Ps1, proj4string = CRS(projection(wrs2)))
sqrt(gArea(spTransform(rect, CRSobj = projLandsat)))/1000  #36.25 km^2
```

```
## [1] 36.25075
```

```
# Load second extent - second zoom
load(file = "../data/myExtent_atoll.RData")
(ext <- myExtent)
```

```
## class       : Extent
## xmin        : 73.43658
## xmax        : 73.48313
## ymin        : 3.879924
## ymax        : 3.930201
```

```
pts <- expand.grid(x = ext[1:2], y = ext[3:4])
pts <- pts[c(1, 2, 4, 3, 1), ]
P1 <- Polygon(pts)
Ps1 <- list(Polygons(list(P1), ID = 1))
rect2 <- SpatialPolygons(Ps1, proj4string = CRS(projection(wrs2)))
sqrt(gArea(spTransform(rect2, CRSobj = projLandsat)))/1000  #5.36 km^2
```

```
## [1] 5.361264
```

The next three plots show progressive levels of detail of the study area. We first show the Maldives islands labelling the 18 landsat scenes which cover them. We also show the 36km$^2$ area, in landsat scene with $path = 145$ and $row = 57$, within which our study area is located.

```
plot(wrs2)
plot(maldives.shp, add = T, border = c("blue", "green", "red")[match(maldives.shp$RB_DEPTH_A,
    c("deep_reef", "shallow_reef", "variable_depth_reef"))])
text(gCentroid(wrs2, byid = T), labels = paste(wrs2$PATH, wrs2$ROW,
    sep = "-"), col = "purple", cex = 0.5)
plot(rect, add = T, border = "black")
plot(rect2, add = T, border = "black")
```

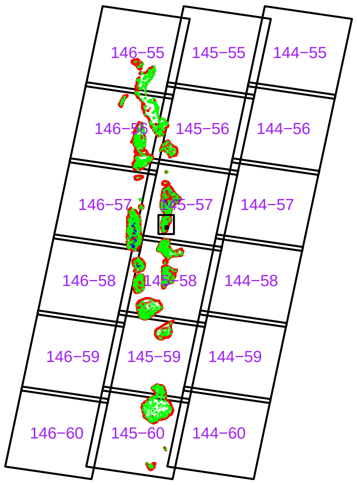


Figure 6: Map of Maldives with Landsat scenes.

The following plot shows the 36km$^2$ area and the 5.4km$^2$ study area within it.

```
plot(crop(maldives.shp, rect), border = c("blue", "green", "red")[match(maldives.shp$RB_DEPTH_A,
    c("deep_reef", "shallow_reef", "variable_depth_reef"))])
plot(rect, add = T, border = "black")
plot(rect2, add = T, border = "black")
```

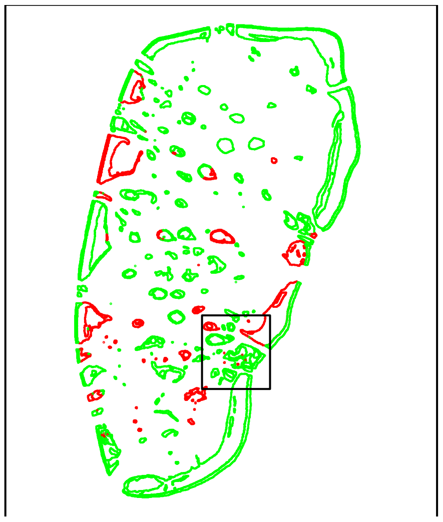


Figure 7: Map of Maldives with Landsat scenes. First close-up.

Finally the following plot shows a our 5.4km$^2$ study-area in detail.

```
plot(crop(maldives.shp, rect2), border = c("blue", "green", "red")[match(maldives.shp$RB_DEPTH_A,
    c("deep_reef", "shallow_reef", "variable_depth_reef"))])
plot(rect2, add = T, border = "black")
```

Figure 8: Map of Maldives with Landsat scenes. Second close-up.

#### 2.1.2 Downloading Landsat data

We use two levels of reflectance data to detect coral bleaching: top-of-atmosphere and surface reflectance data. In section 2.1.3 we explore the different processing levels of reflectance data. We provide all the data necessary to carry out all analyses. However, in this section we show how to download reflectance data to aid in other reflectance based studies or in order to update analyses with data recorded after May of 2017.

##### 2.1.2.1 Top of atmosphere reflectance

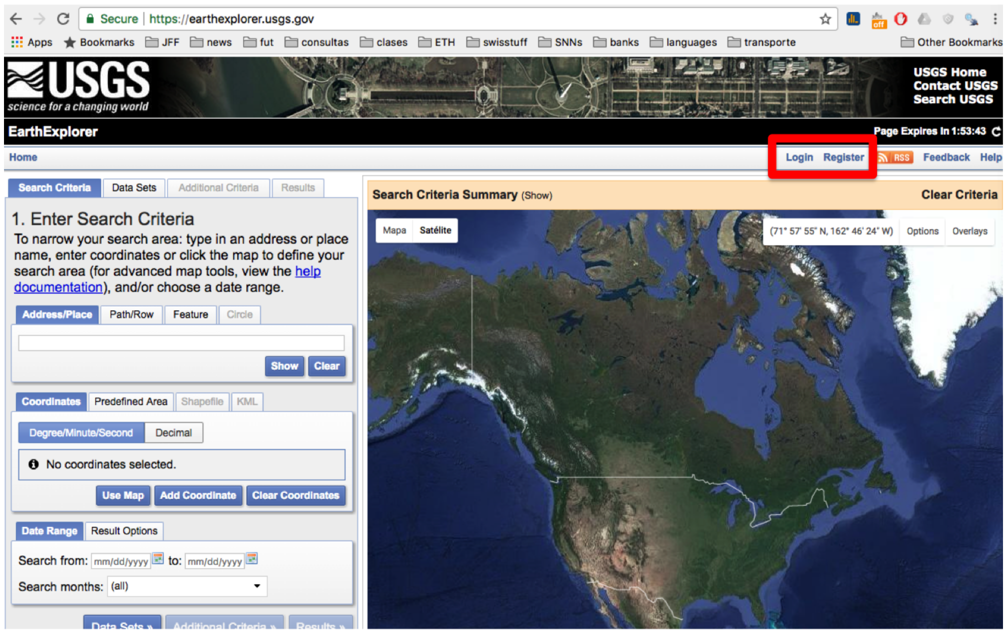
Top-of-atmosphere reflectance data can be ordered from the earth explorer website run by the U.S. Geological Survey (https://earthexplorer.usgs.gov/) .

To download top-of-atmosphere reflectance for many dates in a bulk fashion, we first need to download and install the Bulk Download Application (BDA). Information on how to download, install and use the BDA application can be found at https://www.usgs.gov/media/videos/eros-earthexplorer-how-do-a-bulk-download.

Once we have downloaded the bulk download application we can order and download top-of-atmosphere reflectance data by carrying out the following steps:

1. Go to earth explorer website. https://earthexplorer.usgs.gov/

2. Create account and login


USGS
science for a changing world
USGS Home
Contact USGS
Search USGS
EarthExplorer
Page Expires In 1:53:43
Home
Login Register
RSS Feedback Help
Search Criteria
Data Sets
Additional Criteria
Results
Search Criteria Summary (Show)
Clear Criteria
Mapa Satélite
Options Overlays
Address/Place Path/Row Feature Circle
Show Clear
Coordinates Predefined Area Shapefile KML
Degree/Minute/Second Decimal
Use Map Add Coordinate Clear Coordinates
Date Range Result Options
Data Sets » Additional Criteria » Results »

https://earthexplorer.usgs.gov

1. Enter Search Criteria

To narrow your search area: type in an address or place name, enter coordinates or click the map to define your search area (for advanced map tools, view the help documentation), and/or choose a date range.

No coordinates selected.

Search from: mm/dd/yyyy to: mm/dd/yyyy

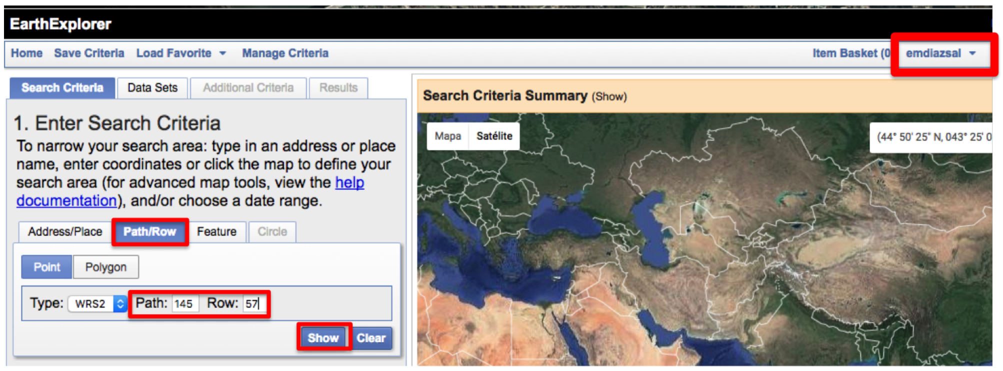
Search months: (all)

(71° 57' 55" N, 162° 46' 24" W)

Figure 9: Create account and login

3. Enter search criteria:

   a. **Path/Row**: Enter path= 145, row= 57 and click on *show*.



1. Enter Search Criteria

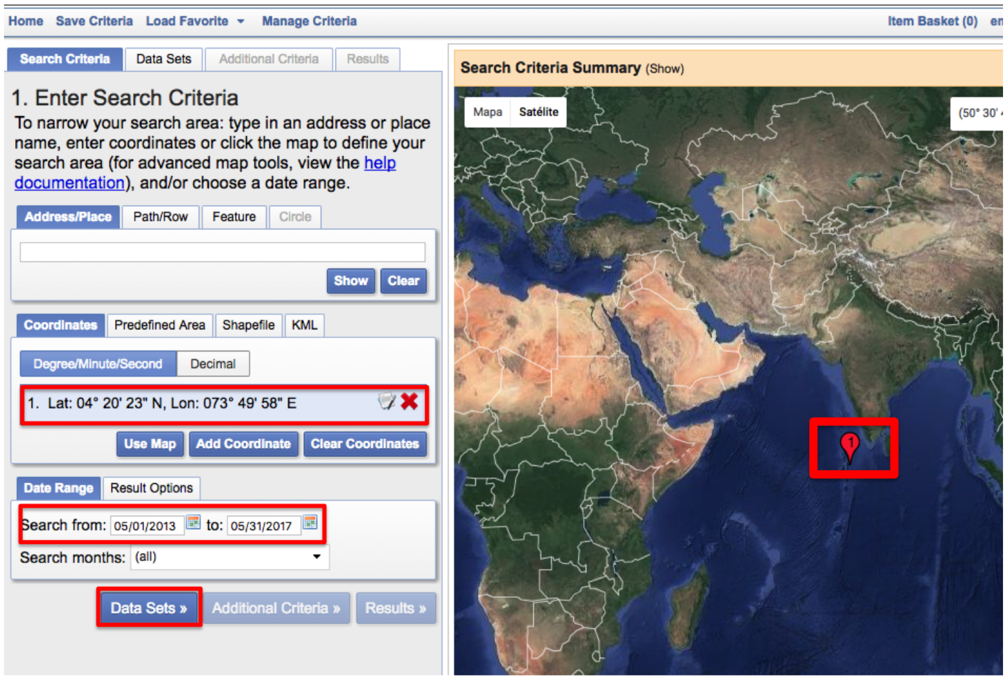
To narrow your search area: type in an address or place name, enter coordinates or click the map to define your search area (for advanced map tools, view the help documentation), and/or choose a date range.

Type: WRS2 Path: 145 Row: 57

(44° 50' 25" N, 043° 25' 0

Figure 10: Enter location

   b. **Date range**:Enter date ranges 2013-05-01 to 2017-05-31 and click on *Data sets*.



1. Enter Search Criteria

To narrow your search area: type in an address or place name, enter coordinates or click the map to define your search area (for advanced map tools, view the help documentation), and/or choose a date range.

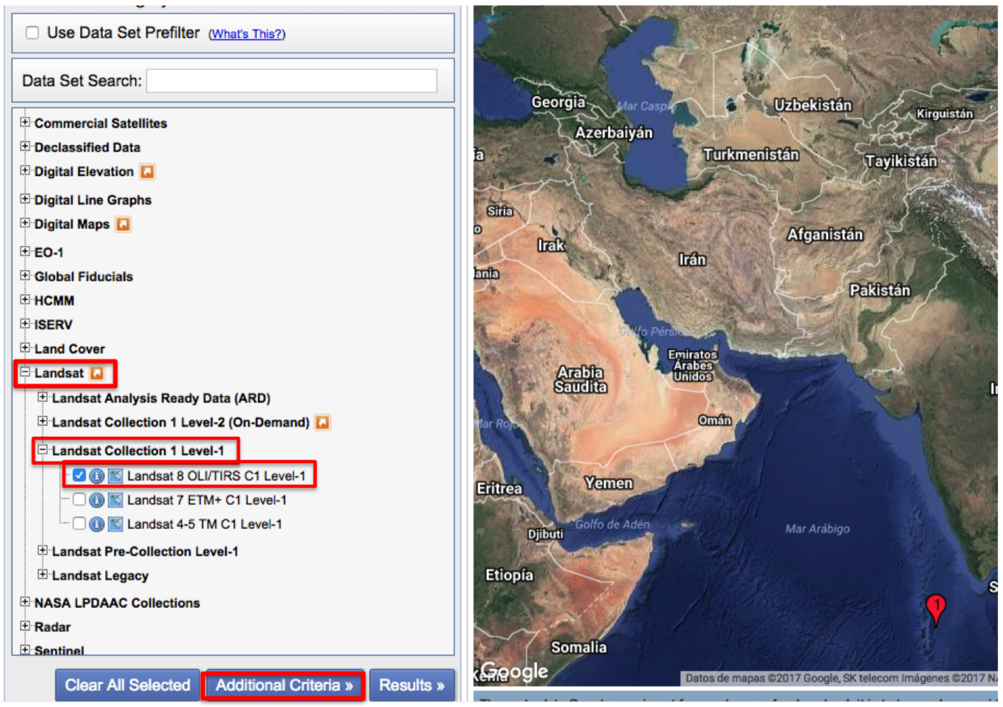
1. Lat: 04° 20' 23" N, Lon: 073° 49' 58" E

Search from: 05/01/2013 to: 05/31/2017

Search months: (all)

Figure 11: Enter dates

4. Select data sets:
   a. Select Landsat
   b. Select Landsat Collection 1 Level-1
   c. Select Landsat 8 OLI/TIRS C1 Level-1
   d. Click on *Additional Criteria*.



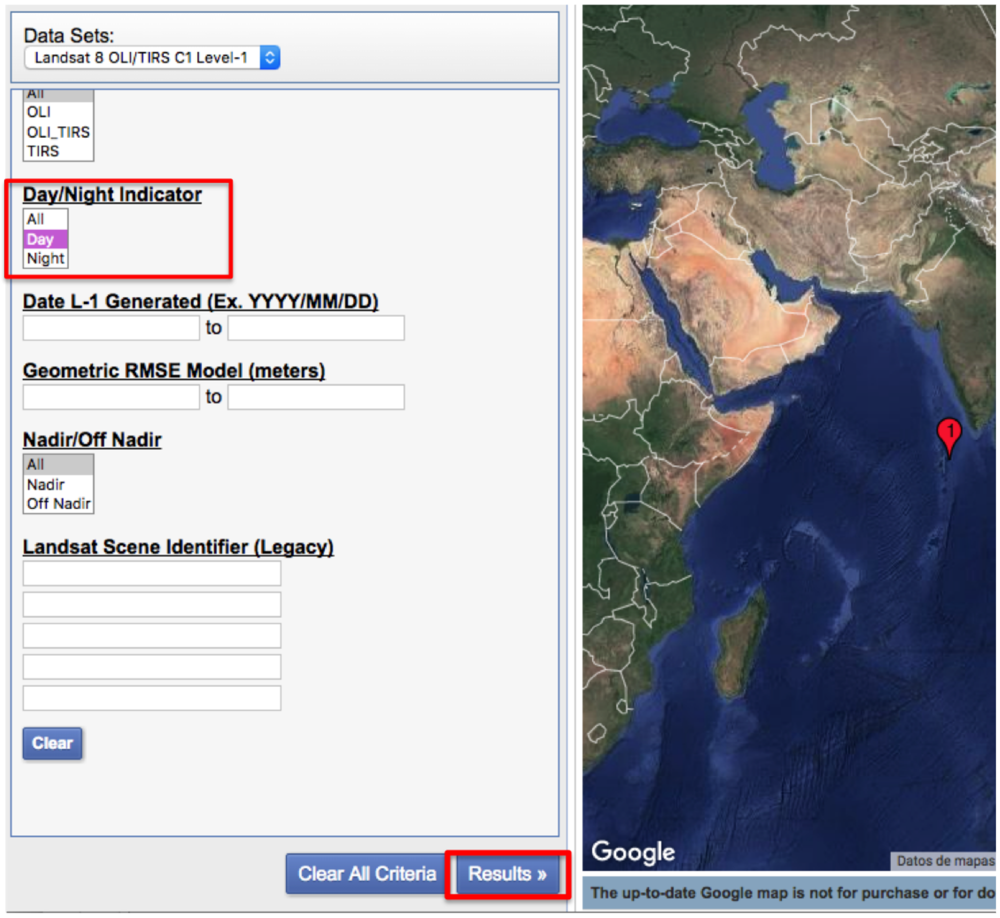
Use Data Set Prefilter (What's This?)

Data Set Search:

- Commercial Satellites
- Declassified Data
- Digital Elevation
- Digital Line Graphs
- Digital Maps
- EO-1
- Global Fiducials
- HCMM
- ISERV
- Land Cover
- Landsat
  - Landsat Analysis Ready Data (ARD)
  - Landsat Collection 1 Level-2 (On-Demand)
  - Landsat Collection 1 Level-1
    - [✓] Landsat 8 OLI/TIRS C1 Level-1
    - Landsat 7 ETM+ C1 Level-1
    - Landsat 4-5 TM C1 Level-1
  - Landsat Pre-Collection Level-1
  - Landsat Legacy
- NASA LPDAAC Collections
- Radar
- Sentinel

Figure 12: Select datasets

5. Additional Criteria: Under *Day/Night Indicator* select *Day* and click on *Results*.


Data Sets: Landsat 8 OLI/TIRS C1 Level-1

All
OLI
OLI_TIRS
TIRS

Day/Night Indicator
All
Day
Night

Date L-1 Generated (Ex. YYYY/MM/DD) to

Geometric RMSE Model (meters) to

Nadir/Off Nadir
All
Nadir
Off Nadir

Landsat Scene Identifier (Legacy)

Figure 13: Select additional criteria

6. Results:
   a. Select *Show Result Controls*
   b. Select *Add All Results From Current Page to Bulk Download*
   c. Repeat for all remaining results pages.
   d. Click on *View Item Basket.*

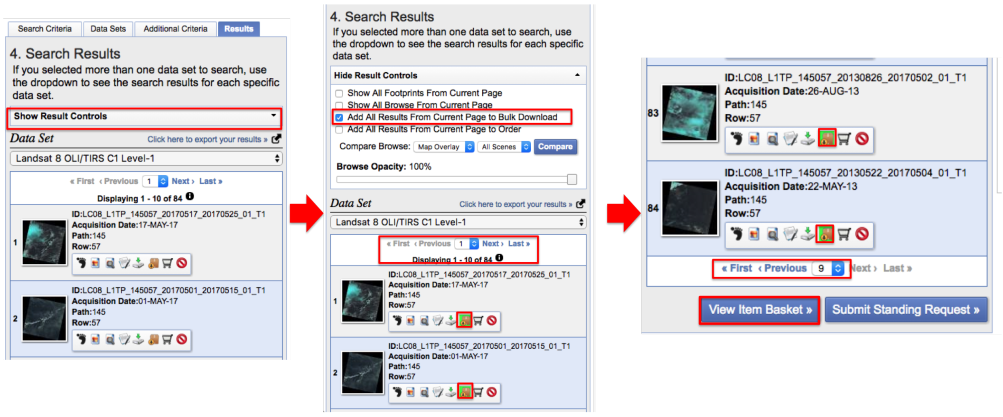

Search Criteria
Data Sets
Additional Criteria
Results
4. Search Results
If you selected more than one data set to search, use the dropdown to see the search results for each specific data set.
Show Result Controls
Data Set
Click here to export your results »
Landsat 8 OLI/TIRS C1 Level-1
« First ‹ Previous 1 Next › Last »
Displaying 1 - 10 of 84
ID:LC08_L1TP_145057_20170517_20170525_01_T1
Acquisition Date:17-MAY-17
Path:145
Row:57
ID:LC08_L1TP_145057_20170501_20170515_01_T1
Acquisition Date:01-MAY-17
Path:145
Row:57
Hide Result Controls
Show All Footprints From Current Page
Show All Browse From Current Page
Add All Results From Current Page to Bulk Download
Add All Results From Current Page to Order
Compare Browse: Map Overlay All Scenes Compare
Browse Opacity: 100%
ID:LC08_L1TP_145057_20130826_20170502_01_T1
Acquisition Date:26-AUG-13
Path:145
Row:57
ID:LC08_L1TP_145057_20130522_20170504_01_T1
Acquisition Date:22-MAY-13
Path:145
Row:57
« First ‹ Previous 9 Next › Last »
View Item Basket »
Submit Standing Request »

Figure 14: Pick results

7. Checkout: Click on *Proceed to Checkout*
8. Download order with Bulk Download Installer
   a. Open Bulk Download Installer
   b. Log in with user and password created in step 1b.
   c. Select order
   d. Click on *Begin download*.

.

Technically the data downloaded in this way is not top-of-atmosphere reflectance but absolute radiance. However, absolute radiance can be converted to top-of-atmosphere using scaling factors (*gain* and *bias* associated to each scene) and the solar elevation/zenith angle (associated to each pixel) which are available in the `ANG.txt` solar illumination and sensor viewing coefficient angle file (see https://landsat.usgs.gov/solar-illumination-and-sensor-viewing-angle-coefficient-file) and the `MTL.txt` meta parameter file associated to each scene. We don't work directly with top-of-atmosphere reflectance and use it only to detect clouds using the fmask algorithm in matlab. However, this algorithm is coded to take in the absolute radiance, angle and meta-parameter files and then convert absolute radiance to top-of-atmosphere reflectance internally so we don't carry out this conversion.

Information on converting Level-1 absolute radiance data to spectral radiance or top-of-atmosphere reflectance can be found at https://landsat.usgs.gov/landsat-8-l8-data-users-handbook-section-5.

#### 2.1.2.2 Surface reflectance

Surface reflectance data can also be ordered from the earth explorer website run by the U.S. Geological Survey (https://earthexplorer.usgs.gov/). To order surface reflectance data repeat steps 1-7 from section 2.1.2.1 only replacing step 4 with the following step and skipping step 5 altogether:

4. Select data sets:
   a. Select Landsat
   b. Select Landsat Collection 1 **Level-2** (On-Demand)
   c. Select Landsat 8 OLI/TIRS C1 **Level-2**
   d. Click on *Results*.

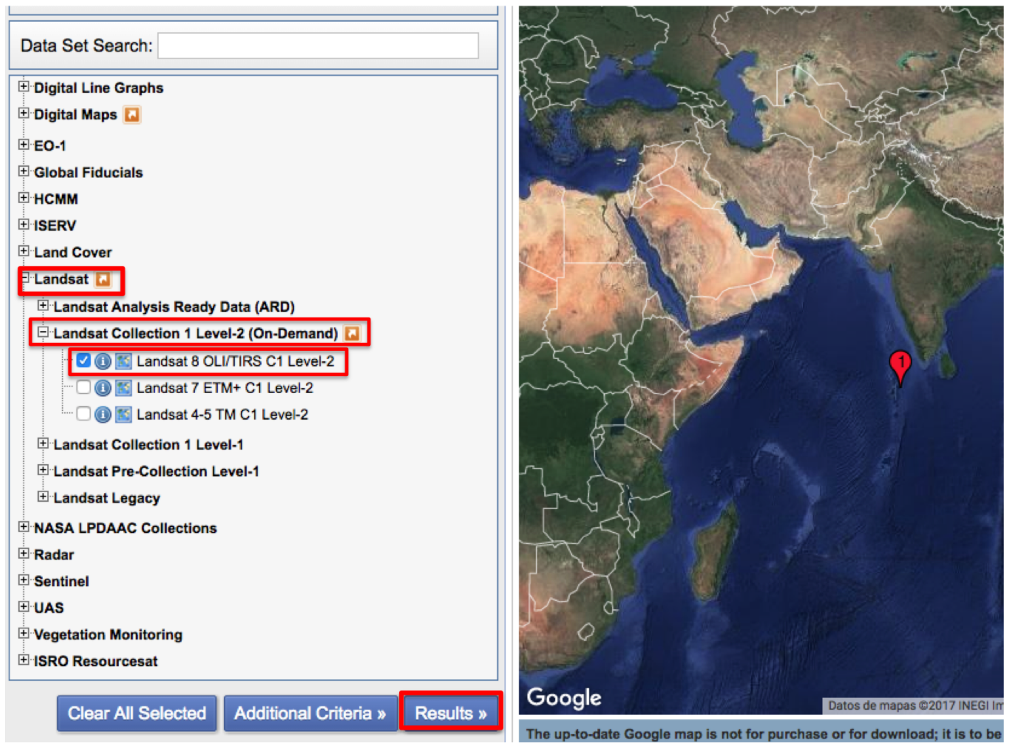

Data Set Search:
Digital Line Graphs
Digital Maps
EO-1
Global Fiducials
HCMM
ISERV
Land Cover
Landsat
Landsat Analysis Ready Data (ARD)
Landsat Collection 1 Level-2 (On-Demand)
Landsat 8 OLI/TIRS C1 Level-2
Landsat 7 ETM+ C1 Level-2
Landsat 4-5 TM C1 Level-2
Landsat Collection 1 Level-1
Landsat Pre-Collection Level-1
Landsat Legacy
NASA LPDAAC Collections
Radar
Sentinel
UAS
Vegetation Monitoring
ISRO Resourcesat
Clear All Selected
Additional Criteria »
Results »
Google
Datos de mapas ©2017 INEGI
The up-to-date Google map is not for purchase or for download; it is to be

Figure 15: Select datasets

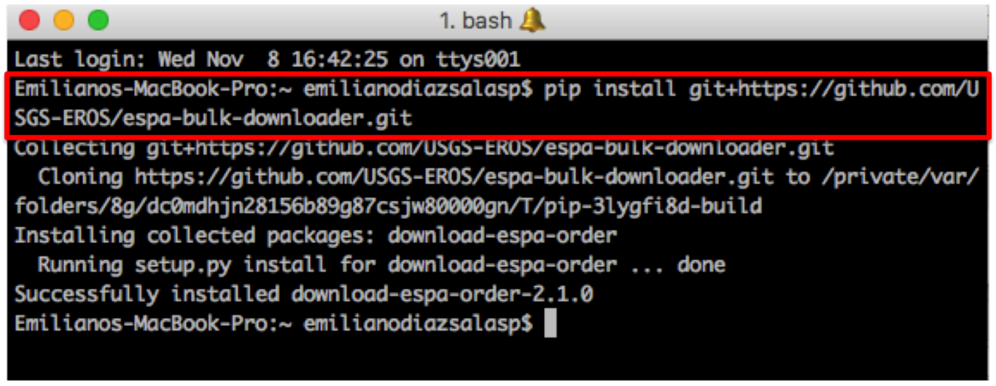
Notice that it is almost the same as before except that here we are ordering Level-2 data (surface reflectance) instead of Level-1 data (top-of-atmosphere reflectance) and we click on *Results* instead of *Additional Criteria*. For the differences between the different levels of reflectance data see section 2.1.3.

Unfortunately, to download top-of-atmosphere reflectance for many dates in a bulk fashion we cannot use the Bulk Download Application of section 2.1.2.1. Instead we must use the ESPA Bulk Download Client. The ESPA Bulk Download Client is a python-based tool that retrieves all completed scenes per user and/or order and places them into a target directory.

To use this client python must be installed on your computer. Once python is installed you can download and use the ESPA Bulk Download Client by carrying out the following steps:

1. Open Computer Command Prompt
2. Install `download_espa_order.py` python package by running following code:

```
pip install git+https://github.com/USGS-EROS/espa-bulk-downloader.git|
```

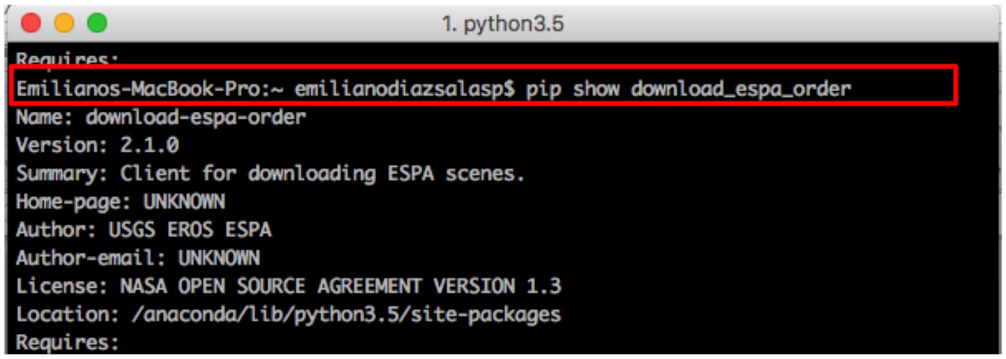


```
Last login: Wed Nov  8 16:42:25 on ttys001
Emilianos-MacBook-Pro:~ emilianodiazsalasp$ pip install git+https://github.com/U
SGS-EROS/espa-bulk-downloader.git
Collecting git+https://github.com/USGS-EROS/espa-bulk-downloader.git
  Cloning https://github.com/USGS-EROS/espa-bulk-downloader.git to /private/var/
folders/8g/dc0mdhjn28156b89g87csjw80000gn/T/pip-3lygfi8d-build
Installing collected packages: download-espa-order
  Running setup.py install for download-espa-order ... done
Successfully installed download-espa-order-2.1.0
Emilianos-MacBook-Pro:~ emilianodiazsalasp$
```

Figure 16: Install ESPA bulk downloader

3. Check that it was succesfully installed with following code:

```
pip show download_espa_order
```



```
Requires:
Emilianos-MacBook-Pro:~ emilianodiazsalasp$ pip show download_espa_order
Name: download-espa-order
Version: 2.1.0
Summary: Client for downloading ESPA scenes.
Home-page: UNKNOWN
Author: USGS EROS ESPA
Author-email: UNKNOWN
License: NASA OPEN SOURCE AGREEMENT VERSION 1.3
Location: /anaconda/lib/python3.5/site-packages
Requires:
```

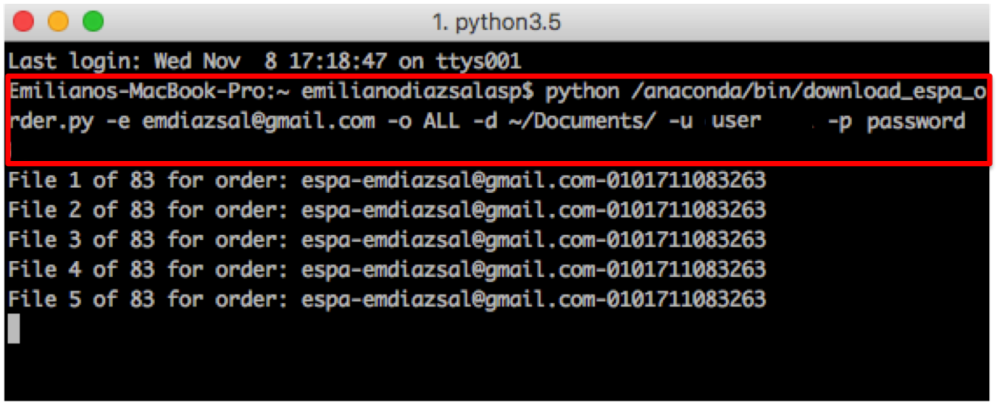
Figure 17: Check for ESPA bulk downloader

4. Start download with following code:

```
python /anaconda/bin/download_espa_order.py -e user@email.com
-o ALL -d ~/Directory/ -u user -p password
```

where

- `user@email.com`, `user` and `password` are the email, user name and password of the USGS account created in step 1b of subsection 2.1.2.1 and
- `~/Directory/` is the directory where the files should be downloaded.



```
Last login: Wed Nov  8 17:18:47 on ttys001
Emilianos-MacBook-Pro:~ emilianodiazsalasp$ python /anaconda/bin/download_espa_o
rder.py -e emdiazsal@gmail.com -o ALL -d ~/Documents/ -u  user     -p password

File 1 of 83 for order: espa-emdiazsal@gmail.com-0101711083263
File 2 of 83 for order: espa-emdiazsal@gmail.com-0101711083263
File 3 of 83 for order: espa-emdiazsal@gmail.com-0101711083263
File 4 of 83 for order: espa-emdiazsal@gmail.com-0101711083263
File 5 of 83 for order: espa-emdiazsal@gmail.com-0101711083263
```

Figure 18: Download surface reflectance order

The surface reflectance data downloaded in this way is in scaled reflectance units (0.0001 scale factor). More information on level-2 products, file names, content, atmospheric correction process, etc. can be found at https://landsat.usgs.gov/sites/default/files/documents/lasrc_product_guide.pdf and https://landsat.usgs.gov/landsat-surface-reflectance-data-products.

More information on how to download surface reflectance can be found at https://landsat.usgs.gov/landsat-surface-reflectance-data-products under section *data access*.

#### 2.1.2.3 Cloud detection using Fmask

In order to detect bleaching events we will monitor historical surface reflectance patterns. However, in order to do so reliably we must first filter out cloud and shadow observations. We will first use the top-of-atmosphere reflectance to detect the presence of clouds or shadows in order to filter out noisy surface reflectance observations affected by clouds. We will use a cloud detection algorithm called fmask (Zhu and Woodcock (2012)) which is available in Matlab. We note that the quality band included in the top-of-atmosphere and surface reflectance products already include cloud flags derived from a cloud detection algorithm called cfmask (see https://landsat.usgs.gov/what-cfmask) which is an adaptation made by the USGS of the fmask algorithm. However, as we will see, when comparing the results of Fmask and CFmask we noticed they were not identical. Comparing both cloud flags against RGB top-of-atmosphere RGB images we considered fmask to detect clouds better so we used the fmask flag to filter out cloudy observations. Although the data provided already includes the fmask cloud filter, we include instructions on how to run fmask to aid in other reflectance based studies or in order to update Maldives bleeching analysis with data published after May of 2017.

The fmask matlab files may be downloaded from https://github.com/prs021/fmask although we also provide them. Additionally we provide the matlab file `autoFmask_bulk` which allows us to call fmask for different landsat scenes. The following steps describe how to implement the fmask algorithm on the top-of-atmosphere reflectence files downloaded in section 2.1.2.1.

1. Download fmask matlab files and `autoFmask_bulk` matlab file to your computer.

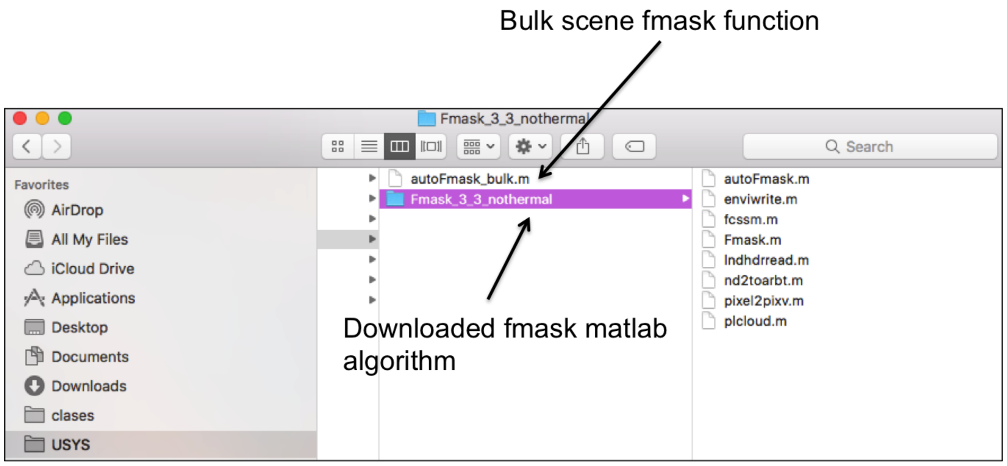
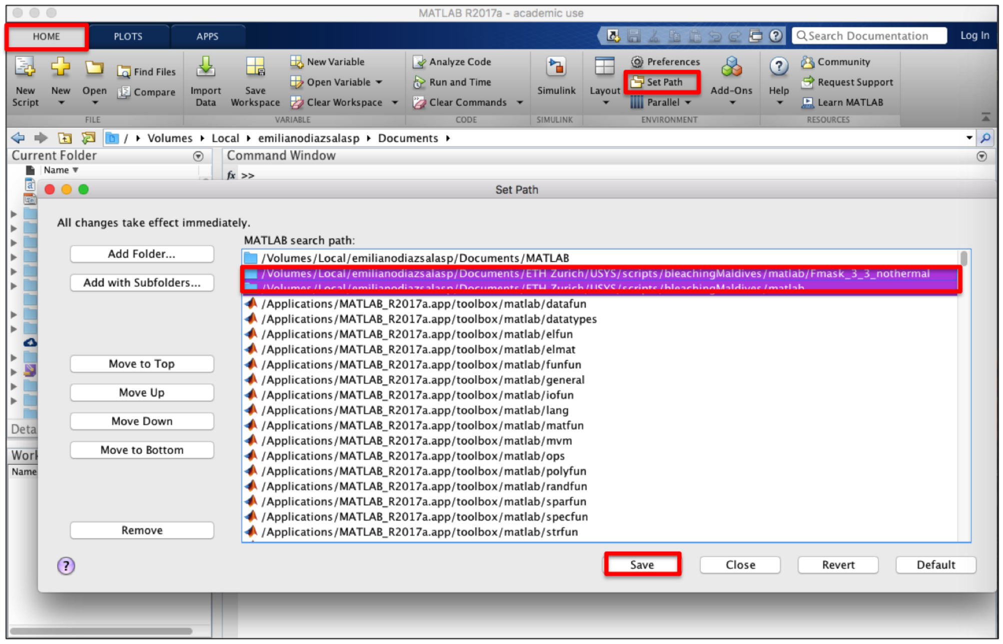


Figure 19: Download fmask matlab files

2. Open matlab and add fmask matlab folder and folder where `autoFmask_bulk` are to path (Home > Set Path).



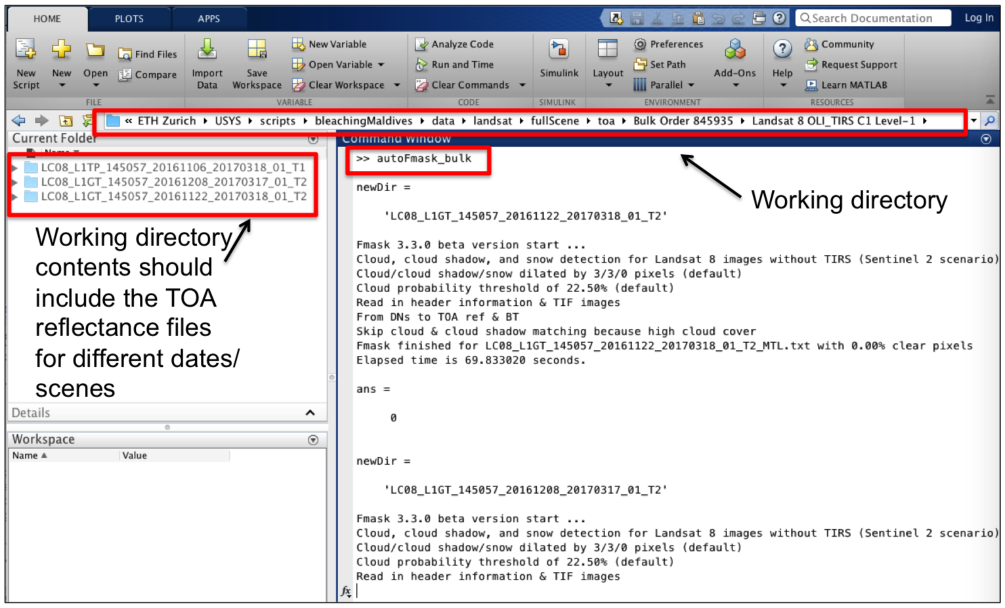
All changes take effect immediately.

MATLAB search path:

```
/Volumes/Local/emilianodiazsalasp/Documents/MATLAB
/Volumes/Local/emilianodiazsalasp/Documents/ETH Zurich/USYS/scripts/bleachingMaldives/matlab/Fmask_3_3_nothermal
/Volumes/Local/emilianodiazsalasp/Documents/ETH Zurich/USYS/scripts/bleachingMaldives/matlab
/Applications/MATLAB_R2017a.app/toolbox/matlab/datafun
/Applications/MATLAB_R2017a.app/toolbox/matlab/datatypes
/Applications/MATLAB_R2017a.app/toolbox/matlab/elfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/elmat
/Applications/MATLAB_R2017a.app/toolbox/matlab/funfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/general
/Applications/MATLAB_R2017a.app/toolbox/matlab/iofun
/Applications/MATLAB_R2017a.app/toolbox/matlab/lang
/Applications/MATLAB_R2017a.app/toolbox/matlab/matfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/mvm
/Applications/MATLAB_R2017a.app/toolbox/matlab/ops
/Applications/MATLAB_R2017a.app/toolbox/matlab/polyfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/randfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/sparfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/specfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/strfun
```

Figure 20: Set path of matlab files

3. Change working directory to folder where the TOA reflectance for different dates is located.
4. Type `autoFmask_bulk` into matlab command window


Working directory
Working directory contents should include the TOA reflectance files for different dates/ scenes
Current Folder
Command Window
Details
Workspace

```
« ETH Zurich ▸ USYS ▸ scripts ▸ bleachingMaldives ▸ data ▸ landsat ▸ fullScene ▸ toa ▸ Bulk Order 845935 ▸ Landsat 8 OLI_TIRS C1 Level-1 ▸
```

```
LC08_L1TP_145057_20161106_20170318_01_T1
LC08_L1GT_145057_20161208_20170317_01_T2
LC08_L1GT_145057_20161122_20170318_01_T2
```

```
>> autoFmask_bulk

newDir =

    'LC08_L1GT_145057_20161122_20170318_01_T2'

Fmask 3.3.0 beta version start ...
Cloud, cloud shadow, and snow detection for Landsat 8 images without TIRS (Sentinel 2 scenario)
Cloud/cloud shadow/snow dilated by 3/3/0 pixels (default)
Cloud probability threshold of 22.50% (default)
Read in header information & TIF images
From DNs to TOA ref & BT
Skip cloud & cloud shadow matching because high cloud cover
Fmask finished for LC08_L1GT_145057_20161122_20170318_01_T2_MTL.txt with 0.00% clear pixels
Elapsed time is 69.833020 seconds.

ans =

     0


newDir =

    'LC08_L1GT_145057_20161208_20170317_01_T2'

Fmask 3.3.0 beta version start ...
Cloud, cloud shadow, and snow detection for Landsat 8 images without TIRS (Sentinel 2 scenario)
Cloud/cloud shadow/snow dilated by 3/3/0 pixels (default)
Cloud probability threshold of 22.50% (default)
Read in header information & TIF images
```

Figure 21: Change working directory and call fmask in bulk

### 2.1.3 Levels of reflectance data

Explain the different levels of data

### 2.1.4 Spectral bands

The landsat 8 level-1 product (top-of-atmosphere) provides information for the following bands:

| Band | Name | type | Wavelength (micrometers) | Resolution | data type | units | missing flag | min | max |
|---|---|---|---|---|---|---|---|---|---|
| BQA | Quality Code | Quality | | 30m | UINT16 | flag | | 0 | 65535 |
| B1 | Ultra blue (coastal/aerosol) | reflectance | 0.435-0.451 | 30m | INT16 | DN | | 0 | 65535 |
| B2 | Blue | | 0.452-0.512 | 30m | INT16 | DN | | 0 | 65535 |
| B3 | Green | | 0.533-0.590 | 30m | INT16 | DN | | 0 | 65535 |
| B4 | Red | | 0.636-0.673 | 30m | INT16 | DN | | 0 | 65535 |
| B5 | Near Infrared (NIR) | | 0.851-0.879 | 30m | INT16 | DN | | 0 | 65535 |
| B6 | Shortwave Infrared (SWIR) 1 | | 1.566-1.651 | 30m | INT16 | DN | | 0 | 65535 |
| B7 | Shortwave Infrared (SWIR) 2 | | 2.107-2.294 | 30m | INT16 | DN | | 0 | 65535 |
| B8 | Panchromatic | | 0.503-0.676 | 15m | INT16 | DN | | 0 | 65535 |
| B9 | Cirrus | | 1.363-1.384 | 30m | INT16 | DN | | 0 | 65535 |
| B10 | Thermal infrared (TIRS) 1 | | 10.60-11.19 | 100m | INT16 | DN | | 0 | 65535 |
| B11 | Thermal infrared (TIRS) 2 | | 11.50-12.51 | 100m | INT16 | DN | | 0 | 65535 |

Table 1: Landsat 8 level-1 bands

The landsat 8 level-2 product (surface reflectance) provides information for the following bands:

| Band | Name | type | Wavelength (micrometers) | Resolution | data type | units | missing flag | min | max | scaling factor |
|---|---|---|---|---|---|---|---|---|---|---|
| pixel_qa | Level-2 quality Code | quality | | 30m | UINT16 | flag | 1 | 0 | 32768 | |
| sr_aerosol | Aerosol QA | | | 30m | UINT8 | flag | | 0 | 255 | |
| radsat_qa | Radiometric saturation QA | | | 30m | UINT16 | flag | 1 | 0 | 3839 | |
| B1 | Ultra blue (coastal/aerosol) | reflectance | 0.435-0.451 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B2 | Blue | | 0.452-0.512 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B3 | Green | | 0.533-0.590 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B4 | Red | | 0.636-0.673 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B5 | Near Infrared (NIR) | | 0.851-0.879 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B6 | Shortwave Infrared (SWIR) 1 | | 1.566-1.651 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |
| B7 | Shortwave Infrared (SWIR) 2 | | 2.107-2.294 | 30m | INT16 | SR | -9999 | 0 | 10000 | 0.0001 |

Table 2: Landsat 8 level-2 bands

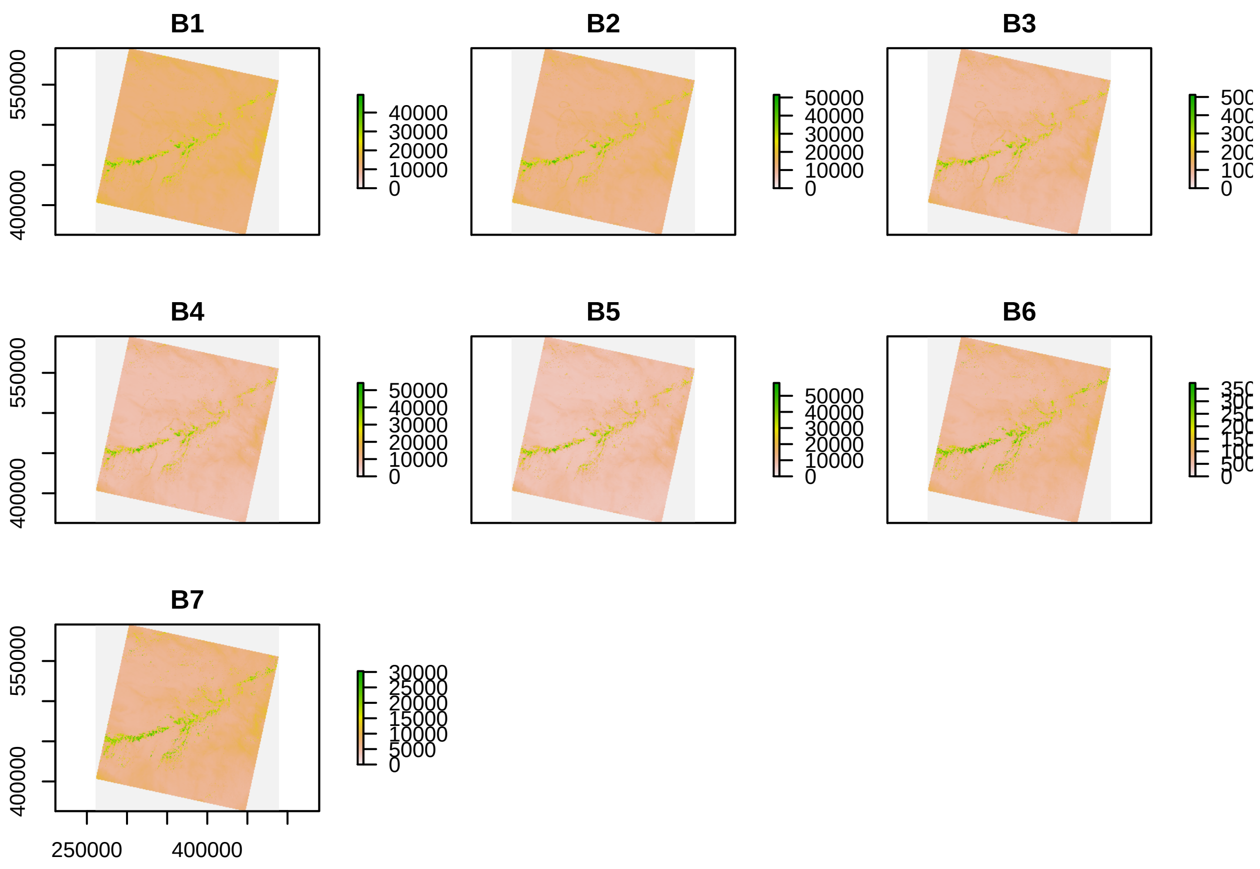
We will explain both the level-1 and level-2 quality bands in section 2.1.5.

We now load into R the top of atmosphere and surface reflectance data for one date.

```
# We'll construct the folder and file strings for each band
# following the structure of files used First the file name
# endings which indicate the relevant band
bands_TOAR <- c("B1", "B2", "B3", "B4", "B5", "B6", "B7", "BQA")
bands_SR <- c("sr_band1", "sr_band2", "sr_band3", "sr_band4",
    "sr_band5", "sr_band6", "sr_band7", "pixel_qa", "radsat_qa")
# Now the route of the order folder
orderFolder_TOAR <- "../data/landsat/fullScene/toa"
orderFolder_SR <- "../data/landsat/fullScene/sr"
# Next the product folder
product_TOAR <- "L8 OLI_TIRS C1 Level-1"
product_SR <- "L8 OLI_TIRS C1 Level-2"
# Next the date folders
dateFolder_TOAR <- "LC08_L1TP_145057_20170501_20170515_01_T1"
dateFolder_SR <- "LC081450572017050101T1-SC20170629205213_20170515_TP"
dateFolder_SR_2 <- paste(substr(dateFolder_SR, 1, 4), paste("L1",
    substr(dateFolder_SR, 50, 51), sep = ""), substr(dateFolder_SR,
    5, 10), substr(dateFolder_SR, 11, 18), substr(dateFolder_SR,
    41, 48), substr(dateFolder_SR, 19, 20), substr(dateFolder_SR,
    21, 22), sep = "_")
# Now we form the file name strings with date folders and
# band file endings
files_TOAR <- paste(dateFolder_TOAR, "_", bands_TOAR[-c(8)],
    ".TIF", sep = "")
files_SR <- paste(dateFolder_SR_2, "_", bands_SR[-c(8, 9)], ".tif",
    sep = "")
file_TOAR_QA <- paste(dateFolder_TOAR, "_", bands_TOAR[8], ".TIF",
    sep = "")
file_SR_cloudQA <- paste(dateFolder_SR_2, "_", bands_SR[8], ".tif",
    sep = "")
file_SR_radsatQA <- paste(dateFolder_SR_2, "_", bands_SR[9],
    ".tif", sep = "")
fileFmask <- paste(dateFolder_TOAR, "_MTLFmask", sep = "")
file_TOAR_MTL <- paste(dateFolder_TOAR, "_MTL.txt", sep = "")
file_SR_MTL <- paste(dateFolder_SR_2, "_MTL.txt", sep = "")
# Now we form the folder & file name with the order, product,
# date folder strings and the file strings
files_TOAR <- paste(orderFolder_TOAR, product_TOAR, dateFolder_TOAR,
    files_TOAR, sep = "/")
files_SR <- paste(orderFolder_SR, product_SR, dateFolder_SR,
    files_SR, sep = "/")
file_TOAR_QA <- paste(orderFolder_TOAR, product_TOAR, dateFolder_TOAR,
    file_TOAR_QA, sep = "/")
file_SR_cloudQA <- paste(orderFolder_SR, product_SR, dateFolder_SR,
    file_SR_cloudQA, sep = "/")
file_SR_radsatQA <- paste(orderFolder_SR, product_SR, dateFolder_SR,
    file_SR_radsatQA, sep = "/")
fileFmask <- paste(orderFolder_TOAR, product_TOAR, dateFolder_TOAR,
    fileFmask, sep = "/")
file_TOAR_MTL <- paste(orderFolder_TOAR, product_TOAR, dateFolder_TOAR,
    file_TOAR_MTL, sep = "/")
file_SR_MTL <- paste(orderFolder_SR, product_SR, dateFolder_SR,
    file_SR_MTL, sep = "/")

# Now we use the stack function from the raster package to
# load the reflectance data into a stack file
scene_TOAR.r <- stack(files_TOAR)
names(scene_TOAR.r) <- bands_TOAR[-c(8)]
scene_SR.r <- stack(files_SR)
names(scene_SR.r) <- bands_TOAR[-c(8)]
```

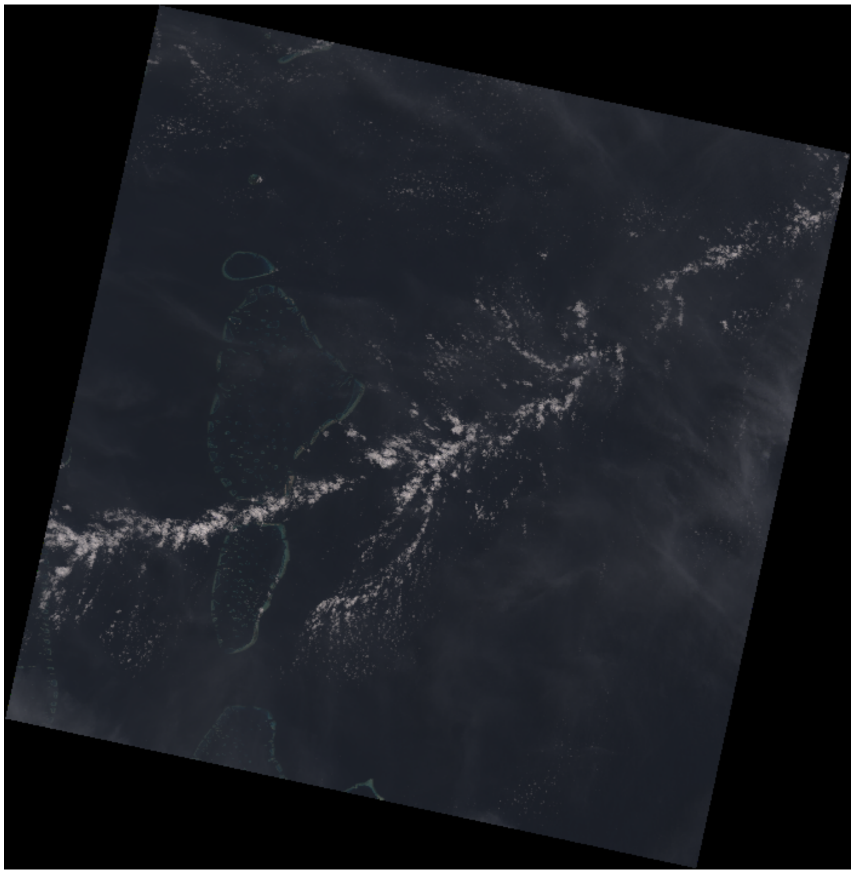
Lets look at all the top-of-atmosphere bands separately.

```
plot(scene_TOAR.r)
```



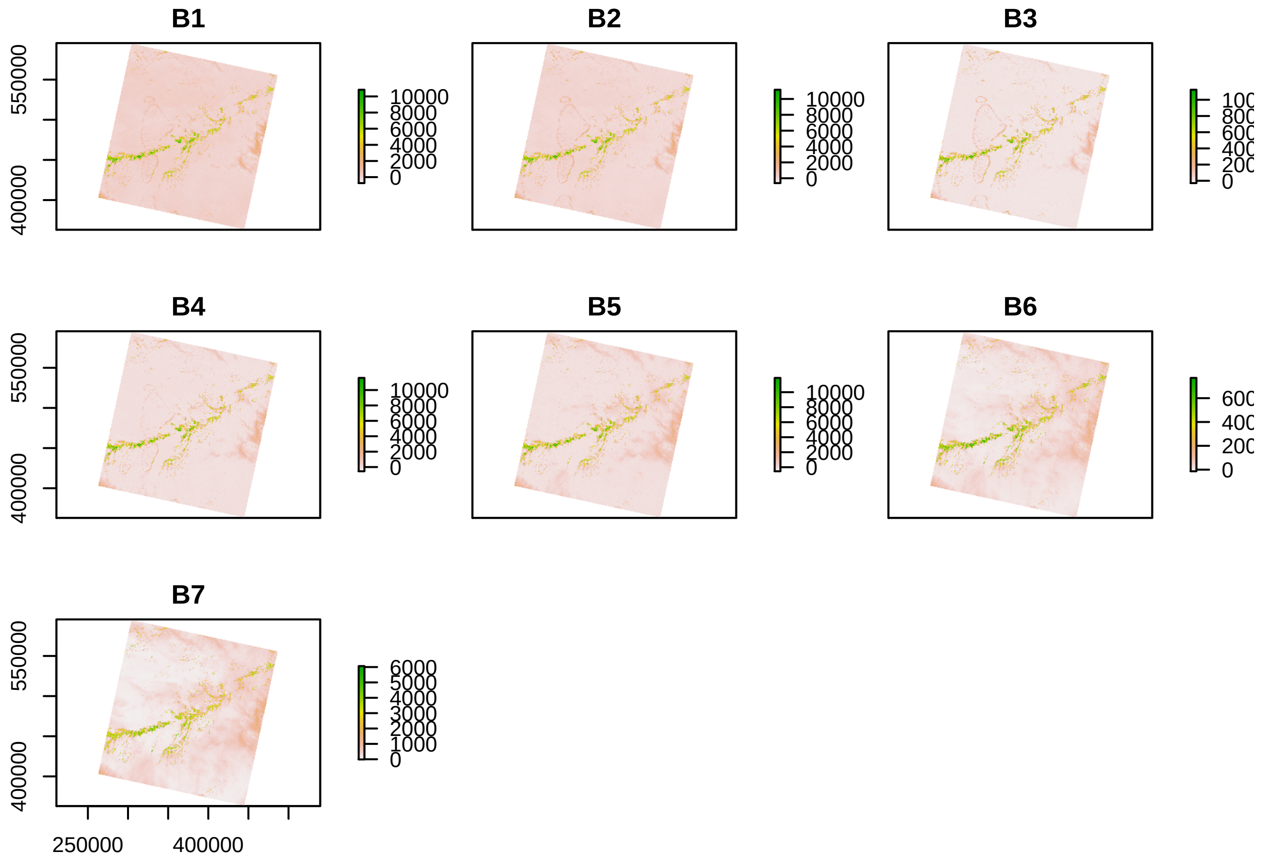
We can combine the red, green and blue bands to obtain a normal RGB image.

```
plotRGB(scene_TOAR.r, 4, 3, 2, scale = 65535)
```



We do the same for the surface reflectance data.

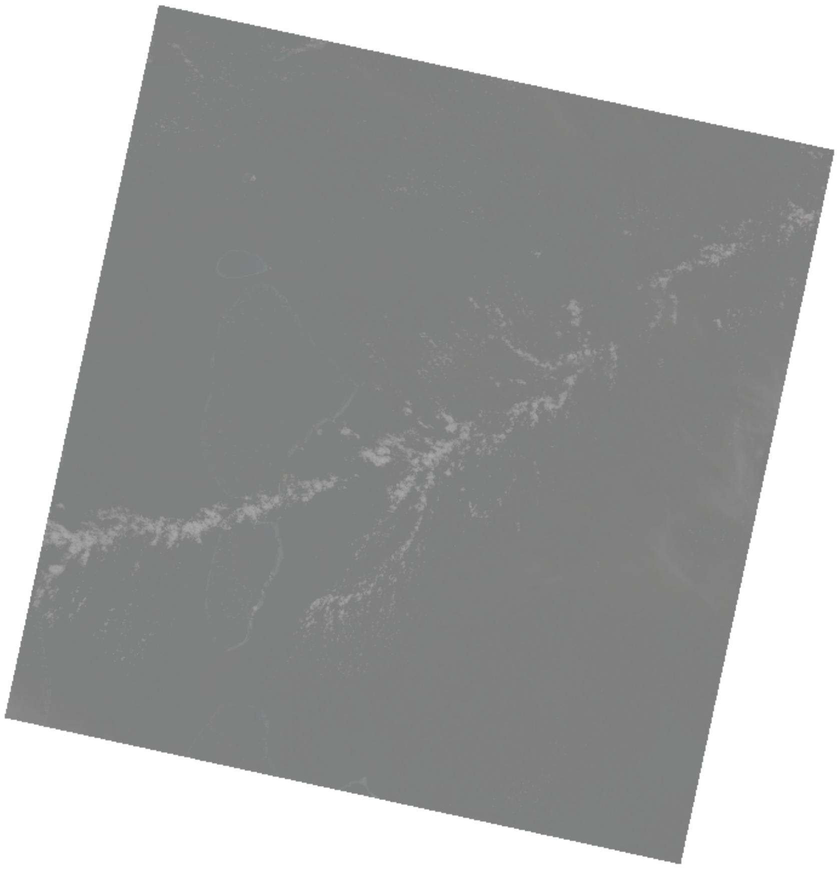
```
plot(scene_SR.r)
```



To combine surface reflectance red, green and blue bands into a normal RGB image we must first re-scale the data.

```
# minValue(raster) much faster than min(value)
minSR <- min(minValue(scene_SR.r[[c(4, 3, 2)]]))
maxSR <- max(maxValue(scene_SR.r[[c(4, 3, 2)]]))
# rescaling to 0-1
scaleSR <- function(x) (x - minSR)/(maxSR - minSR)

# calc(raster, function) much faster than function(raster)
scene_SR_2.r <- calc(scene_SR.r[[c(4, 3, 2)]], scaleSR)
```

```
plotRGB(scene_SR_2.r, 1, 2, 3, scale = 1)
```

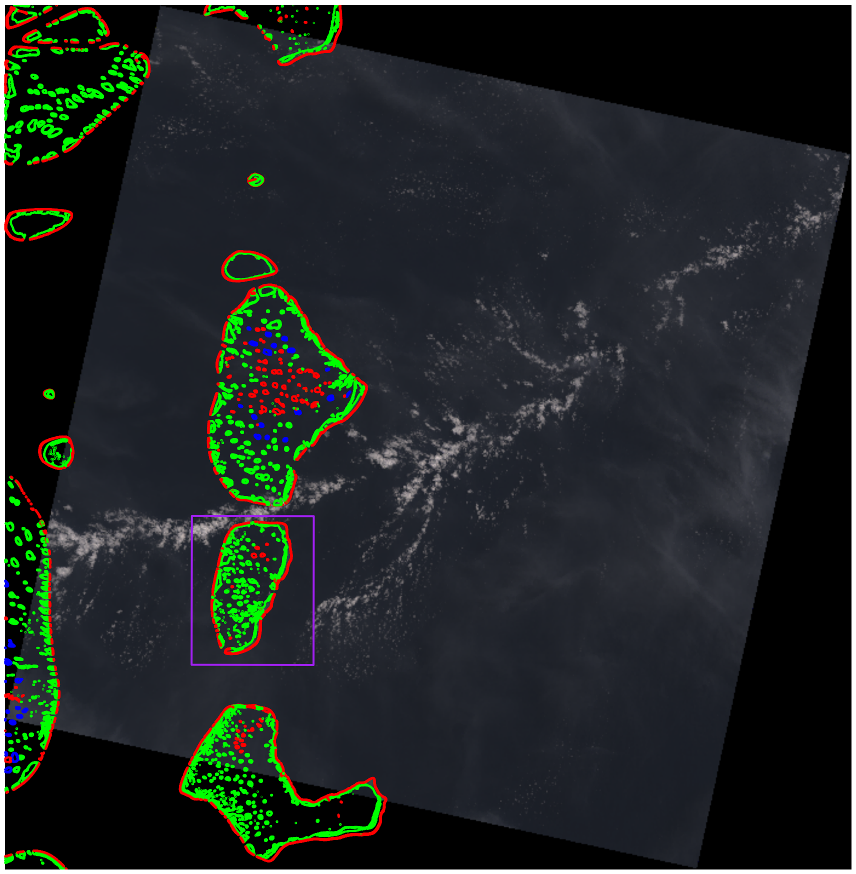


We finish this subsection by plotting the landsat scene and overlaying the Maldives atoll shapes.

```r
# Transform the malidves shape object collection into the
# geographic reference system of landsat info
maldives.shp <- spTransform(maldives.shp, CRSobj = CRS(projection(scene_TOAR.r)))
# Crop the maldives shape object collection to landsat scene
maldives.shp <- crop(maldives.shp, extent(scene_TOAR.r))
# We also transform the 36 km^2 rectangle object into landsat
# coordinates
rect <- spTransform(rect, CRSobj = CRS(projection(scene_TOAR.r)))
sqrt(gArea(rect))/1000  # around 36km^2 atoll scene
```

```
## [1] 36.25075
```

```r
plotRGB(scene_TOAR.r, 4, 3, 2, scale = 65535)
plot(maldives.shp, add = T, border = c("blue", "green", "red")[match(maldives.shp$RB_DEPTH_A,
    c("deep_reef", "shallow_reef", "variable_depth_reef"))])
plot(rect, add = T, border = "purple")
```



### 2.1.5 Quality flags

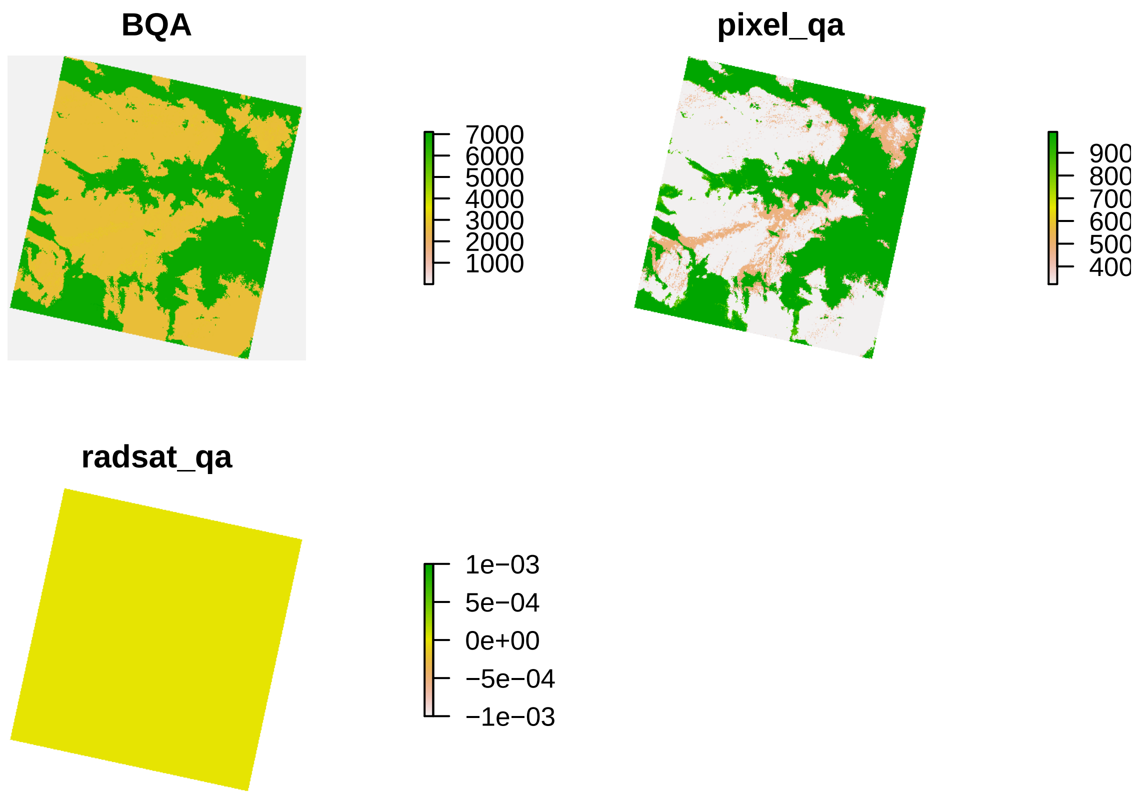
We will use the level-1 Quality Assurance band `BQA` and the level-2 quality bands `pixel_qa` and `radsat_qa`. The `BQA` encodes flags concerning the presence of cloud, shadow, snow and water affecting the quality of the level-1 measurements for a given pixel. The Level-2 Pixel Quality Assurance band (pixel_qa) is calculated using information from the Level-1 Quality Assurance band, specifically Cloud Confidence, Cloud Shadow and Snow/Ice flags. Additionally water values are re-calculated, and high-confidence cloud pixels are dilated. (see section 7.2 from https://landsat.usgs.gov/sites/default/files/documents/lasrc_product_guide.pdf). The Radiometric Saturation Quality (radsat_qa) band is a bit packed representation of which sensor bands were saturated during data capture, yielding unusable data. The level-2 download described in section 2.1.2.2 also includes an Internal Surface Reflectance Aerosol Quality (sr_aerosol) band which provides low-level detail about the factors that may have influenced atmospheric correction. We do not work with this quality band.

Lets first load the quality bands into a raster object in R .

```
scene_TOAR_QA.r <- raster(file_TOAR_QA)
scene_SR_cloudQA.r <- raster(file_SR_cloudQA)
scene_SR_radsatQA.r <- raster(file_SR_radsatQA)
```

We now plot them together with to get a sense of the information they provide.

```
plot(stack(list(BQA = scene_TOAR_QA.r, pixel_qa = scene_SR_cloudQA.r,
    radsat_qa = scene_SR_radsatQA.r)), box = F, axes = F)
```



Comparing them to the RGB image of the scene we can see they contain cloud information and that there appear to be no saturation problems. However, we have not yet decoded this information into easily interpretable flags. We will do this later in this section. We first load the fmask band to R . The fmask algorithm in matlab produces High Dynamic Range image file (.hdr) which we load into R using the `read.ENVI` function from the `caTools` package.

```
library(caTools)
fmask.r <- raster(read.ENVI(fileFmask, headerfile = paste(fileFmask,
    ".hdr", sep = "")), crs = CRS(projection(scene_TOAR.r)),
    template = scene_TOAR.r)
# the value '255' is missing value in fmask
values(fmask.r)[which(values(fmask.r) == 255)] <- NA
table(values(fmask.r))
```

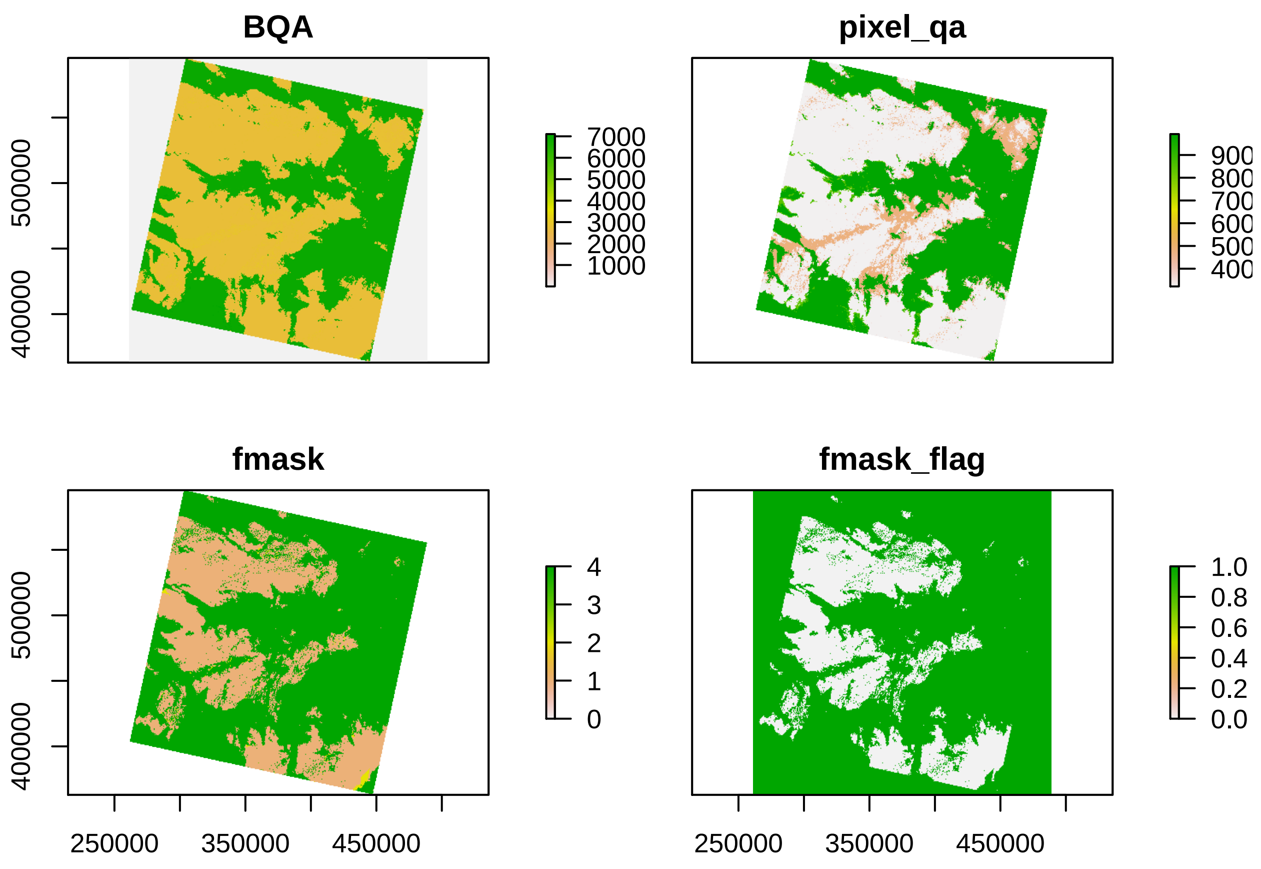
```
##
##        0        1        2        3        4
##     1762 13817798   246991     6257 27381416
```

The following table shows the coding for the fmask flag:

| Value | Meaning |
|---|---|
| 0 | land |
| 1 | clear water |
| 2 | cloud shadow |
| 3 | snow |
| 4 | cloud |
| 255 | outside scene |

We now plot the fmask quality band showing land and water pixels (0 or 1) and compare to the landsat quality bands:

```r
plot(stack(list(BQA = scene_TOAR_QA.r, pixel_qa = scene_SR_cloudQA.r,
    fmask = fmask.r, fmask_flag = !(fmask.r %in% c(0, 1)))))
```



As we can see, the fmask band is slightly different to the landsat quality bands. By now we have loaded all the pixelwise information for the chosen landsat scene and date. The scenewise information is stored in the MTL metafile. We use the `readMeta` from the `RStoolbox` package to read this file which has a special format. In particular we are interested in obtaining the exact hour at which the scene was captured since we will use this information later on.

```r
library(RStoolbox)
MTL_TOAR <- readMeta(file_TOAR_MTL, raw = FALSE)
MTL_TOAR$ACQUISITION_DATE
```

```
## [1] "2017-05-01 05:18:43 GMT"
```

```r
MTL_SR <- readMeta(file_SR_MTL, raw = FALSE)
MTL_SR$ACQUISITION_DATE
```

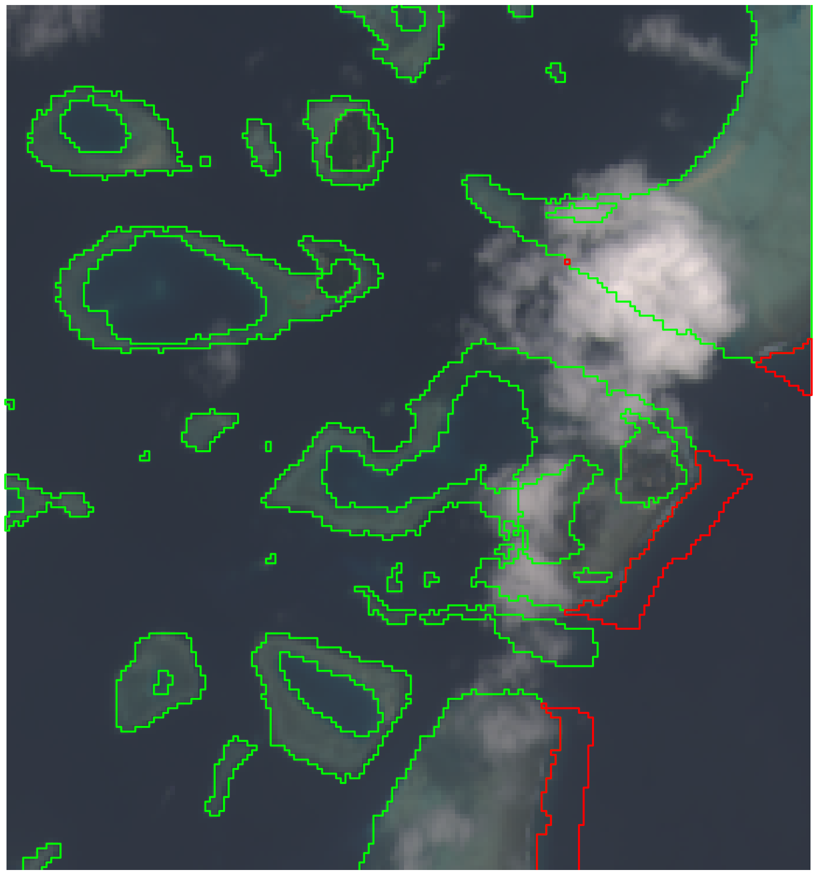
```
## [1] "2017-05-01 05:18:43 GMT"
```

Since decoding the quality flags will be a resource intensive task we first *crop* the landsat scene to the 5.35 km$^2$ study area we chose in section 2.1.1.

```r
# need to convert 5.35km^2 extent to landsat coordinates
# before cropping
ext <- myExtent
```

```
pts <- expand.grid(x = ext[1:2], y = ext[3:4])
pts <- pts[c(1, 2, 4, 3, 1), ]
P1 <- Polygon(pts)
Ps1 <- list(Polygons(list(P1), ID = 1))
rect <- SpatialPolygons(Ps1, proj4string = CRS(projection(wrs2)))
rect <- spTransform(rect, CRSobj = CRS(projection(scene_TOAR.r)))
myExtent <- extent(rect)

# Crop all level-1 top-of-atmosphere, level-2 surface
# refletance, quality bands and maldives shape object to
# 5.35km^2 extent
scene_TOAR.r <- crop(scene_TOAR.r, myExtent)
scene_SR.r <- crop(scene_SR.r, myExtent)
scene_TOAR_QA.r <- crop(scene_TOAR_QA.r, myExtent)
scene_SR_cloudQA.r <- crop(scene_SR_cloudQA.r, myExtent)
fmask.r <- crop(fmask.r, myExtent)
scene_SR_radsatQA.r <- crop(scene_SR_radsatQA.r, myExtent)
maldives.shp <- crop(maldives.shp, myExtent)
```

```
plotRGB(scene_TOAR.r, 4, 3, 2)
plot(maldives.shp, add = T, border = c("blue", "green", "red")[match(maldives.shp$RB_DEPTH_A,
    c("deep_reef", "shallow_reef", "variable_depth_reef"))])
```



Since we have now loaded a raster object of our chosen study area we have available a raster grid for this area and we can rasterize the maldives shape object collection using the `rasterize` function from the `raster` package.

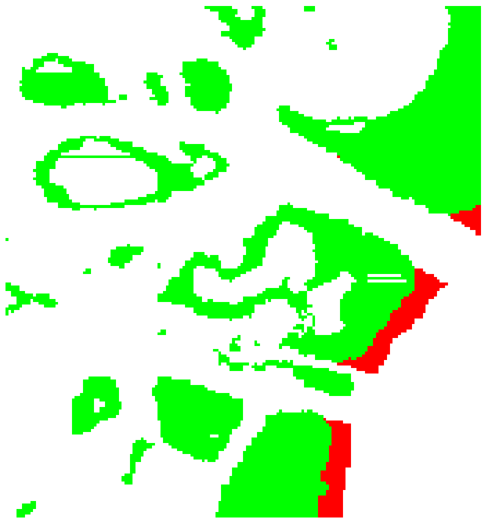
```
# rasterize maldives shape object collection using 5.35km^2
# study area landsat raster grid
coral.r <- rasterize(maldives.shp, scene_TOAR.r, field = as.numeric(maldives.shp$RB_DEPTH_A))

table(values(coral.r))/ncell(coral.r) * 100  # 26.75% shallow reef, 2.19% variable depth reef
```

```
##
##         2         3
## 26.750471  2.190446
```

```
sum(is.na(values(coral.r)))/ncell(coral.r) * 100  # 71.06% non-reef
```

```
## [1] 71.05908
```

```
plot(coral.r, legend = F, box = F, axes = F, col = c("green",
    "red"))
```



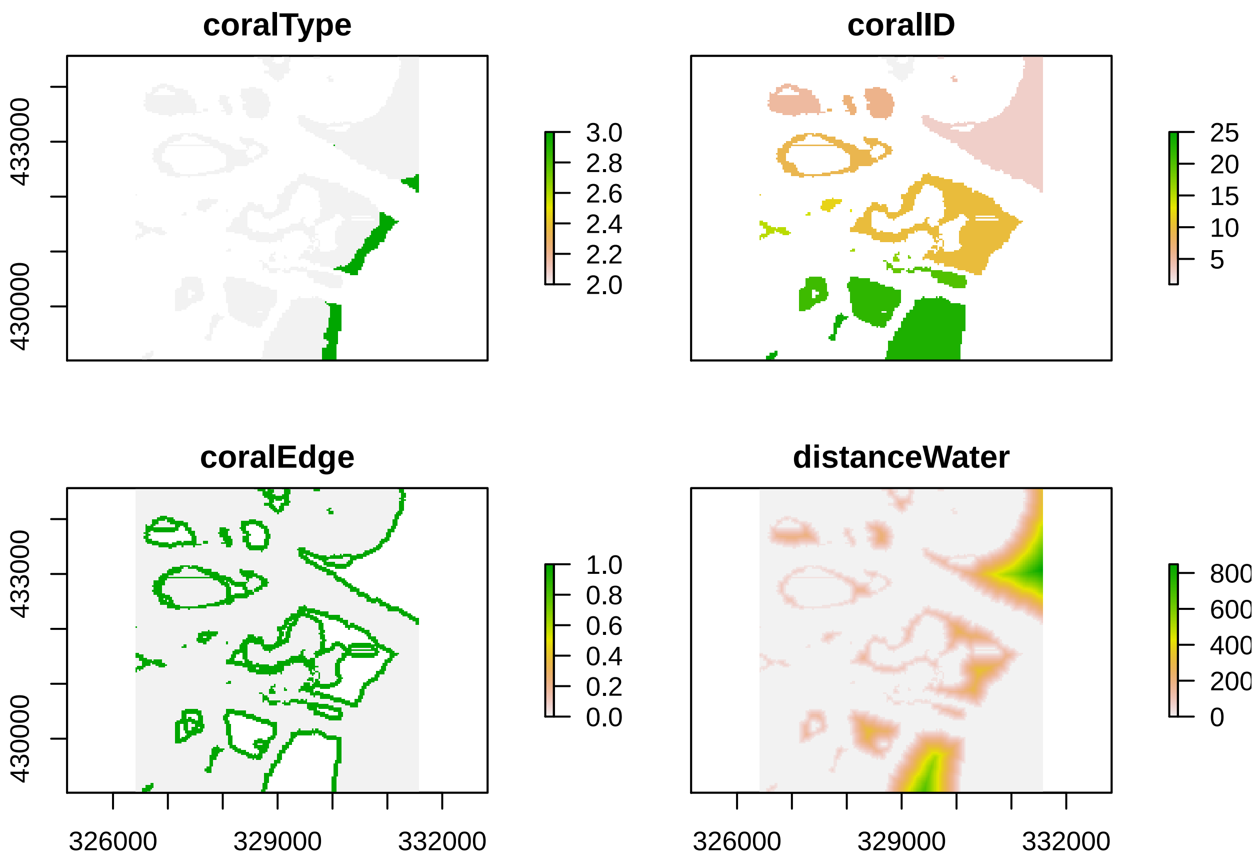
We now generate some more rasters which might help us to detect special bleaching patterns.

```
# Generate a water/non-reef raster.
water.r <- is.na(coral.r)
values(water.r)[which(values(water.r) == 0)] <- NA
# Generate a raster which shows nearest distance to a
# non-reef pixel of each pixel
waterBuffer.r <- buffer(water.r, width = 30 * 3, doEdge = T)
# Generate an indicator raster showing coral-reef edges
coralThickEdge.r <- waterBuffer.r & is.na(water.r)
# We stack original coral reef raster with newly generated
# rasters
coral.r <- stack(coral.r, clump(coral.r), coralThickEdge.r, distance(water.r))
```

```
## Loading required namespace: igraph
```

```
names(coral.r) <- c("coralType", "coralID", "coralEdge", "distanceWater")
```

```
plot(coral.r)
```



We will now decode the landsat quality bands to obtain easily interpretable quality flags. The level-1 quality band `BQA` can be decoded with the following table taken from https://landsat.usgs.gov/collectionqualityband. We note that table 5.1 from https://landsat.usgs.gov/sites/default/files/documents/Landsat8DataUsersHandbook.pdf has conflicting information but the first source seems to be consitent with the data.

| Inverse position in 16-bit | variable encoded | values | description |
|---|---|---|---|
| 0 | designated fill | 0<br>1 | |
| 1 | terrain occlusion | 0<br>1 | no<br>yes |
| 2-3 | radiometric saturation | 00<br>01<br>10<br>11 | No bands contain saturation<br>1-2 bands contain saturation<br>3-4 bands contain saturation<br>5 or more bands contain saturation |
| 4 | cloud | 0<br>1 | no<br>yes |
| 5-6 | cloud confidence | 00<br>01<br>10<br>11 | Not Determined<br>Low (0-33%)<br>Medium (34-66%)<br>High (67-100%) |
| 7-8 | cloud shadow confidence | 00<br>01<br>10<br>11 | Not Determined<br>Low (0-33%)<br>Medium (34-66%)<br>High (67-100%) |
| 9-10 | snow/ice confidence | 00<br>01<br>10<br>11 | Not Determined<br>Low (0-33%)<br>Medium (34-66%)<br>High (67-100%) |
| 11-12 | cirrus confidence | 00<br>01<br>10<br>11 | Not Determined<br>Low (0-33%)<br>Medium (34-66%)<br>High (67-100%) |

Table 4: BQA quality flags

We now write a function `unravelQA` in R to implement this decoding.

```
# Helper function which transforms integer to k-bits (default
# 16 bits) This function will be used by decoding functions.
first_k_bits <- function(int, k = 16, reverse = T) {
```

```r
    integer_vector <- as.integer(intToBits(int))[1:k]
    if (reverse)
        integer_vector <- rev(integer_vector)
    return(paste(as.character(integer_vector), collapse = ""))
}

unravelQA <- function(sceneQA.r) {

    # transform integer values to 16 bits
    ints <- sapply(values(sceneQA.r), first_k_bits, reverse = F)
    # designated fill
    fill.r <- sceneQA.r
    values(fill.r) <- as.numeric(as.factor(substr(ints, 1, 1))) -
        1
    # terrain occlusion
    terrainOcclusion.r <- sceneQA.r
    values(terrainOcclusion.r) <- as.numeric(as.factor(substr(ints,
        2, 2))) - 1
    # radiometric saturation
    sat.r <- sceneQA.r
    values(sat.r) <- as.numeric(as.factor(substr(ints, 3, 4))) -
        1
    # cloud flag
    cloud.r <- sceneQA.r
    values(cloud.r) <- as.numeric(as.factor(substr(ints, 5, 5))) -
        1
    # cloud confidence
    cloudConf.r <- sceneQA.r
    values(cloudConf.r) <- as.numeric(as.factor(substr(ints,
        6, 7))) - 1
    # cloud shadow confidence
    cloudShadowConf.r <- sceneQA.r
    values(cloudShadowConf.r) <- as.numeric(as.factor(substr(ints,
        8, 9))) - 1
    # snow/ice confidence
    snowIceConf.r <- sceneQA.r
    values(snowIceConf.r) <- as.numeric(as.factor(substr(ints,
        10, 11))) - 1
    QA.r <- stack(fill.r, terrainOcclusion.r, sat.r, cloud.r,
        cloudConf.r, cloudShadowConf.r, snowIceConf.r)
    names(QA.r) <- c("Designated Fill", "Terrain occlusion",
        "Radiometric Saturation", "Cloud", "Cloud Confidence",
        "Cloud Shadow Confidence", "Snow/Ice Confidence")
    return(QA.r)
}
```

The level-2 quality band `pixel_qa` can be decoded with the following table taken from the table 7.2 from https://landsat.usgs.gov/sites/default/files/documents/lasrc_product_guide.pdf.

| Inverse position in 16-bit | variable encoded | values | description |
|---|---|---|---|
| 0 | designated fill | 0<br>1 | |
| 1 | clear | 0<br>1 | no<br>yes |
| 2 | water | 0<br>1 | no<br>yes |
| 3 | cloud shadow | 0<br>1 | no<br>yes |
| 4 | snow | 0<br>1 | no<br>yes |
| 5 | cloud | 0<br>1 | no<br>yes |
| 6-7 | cloud confidence | 00<br>01<br>10<br>11 | none<br>low<br>medium<br>high |
| 8-9 | cirrus confidence | 00<br>01<br>10<br>11 | not set<br>low from band 9<br>medium from band 9<br>high from band 9 |
| 10 | terrain occlusion | 0<br>1 | no<br>yes |

Table 5: pixel_qa quality flags

We now write a function `unravelQA_l2_cld` in R to implement this decoding.

```
unravelQA_l2_cld <- function(sceneQA.r) {

    # transform integer values to 16 bits
    ints <- sapply(values(sceneQA.r), first_k_bits, reverse = F)

    # designated fill
    fill.r <- sceneQA.r
    values(fill.r) <- as.numeric(as.factor(substr(ints, 1, 1))) -
        1
    # clear pixel flag
    clear.r <- sceneQA.r
    values(clear.r) <- as.numeric(as.factor(substr(ints, 2, 2))) -
        1
    # water flag
    water.r <- sceneQA.r
    values(water.r) <- as.numeric(as.factor(substr(ints, 3, 3))) -
        1
    # shadow flag
    shadow.r <- sceneQA.r
    values(shadow.r) <- as.numeric(as.factor(substr(ints, 4,
        4))) - 1
    # snow flag
    snow.r <- sceneQA.r
    values(snow.r) <- as.numeric(as.factor(substr(ints, 5, 5))) -
        1
    # cloud flag
    cloud.r <- sceneQA.r
    values(cloud.r) <- as.numeric(as.factor(substr(ints, 6, 6))) -
        1
    # cloud confidence
    cloudConfidence.r <- sceneQA.r
    values(cloudConfidence.r) <- as.numeric(as.factor(substr(ints,
        7, 8))) - 1
    # cirrus confidence
    cirrusConfidence.r <- sceneQA.r
    values(cirrusConfidence.r) <- as.numeric(as.factor(substr(ints,
        9, 10))) - 1
    # terrain occlusion flag
    terrainOcclusion.r <- sceneQA.r
    values(terrainOcclusion.r) <- as.numeric(as.factor(substr(ints,
        11, 11))) - 1

    QA.r <- stack(fill.r, clear.r, water.r, shadow.r, snow.r,
        cloud.r, cloudConfidence.r, cirrusConfidence.r, terrainOcclusion.r)
    names(QA.r) <- c("designated fill", "clear pixel", "water",
        "shadow", "snow", "cloud", "cloud confidence", "cirrus confidence",
        "terrain occlusion")
    return(QA.r)
}
```

The level-2 quality band `radsat_qa` can be decoded with the following table taken from the table 7.5 from https://landsat.usgs.gov/sites/default/files/documents/lasrc_product_guide.pdf.

| Inverse position in 16-bit | variable encoded | values | description |
|---|---|---|---|
| 0 | data fill flag | 0<br>1 | valid data<br>invalid data |
| 1 | band 1 saturation | 0<br>1 | no<br>yes |
| 2 | band 2 saturation | 0<br>1 | no<br>yes |
| 3 | band 3 saturation | 0<br>1 | no<br>yes |
| 4 | band 4 saturation | 0<br>1 | no<br>yes |
| 5 | band 5 saturation | 0<br>1 | no<br>yes |
| 6 | band 6 saturation | 0<br>1 | no<br>yes |
| 7 | band 7 saturation | 0<br>1 | no<br>yes |
| 8 | not used | 0<br>1 | |
| 9 | band 9 saturation | 0<br>1 | no<br>yes |
| 10 | band 10 saturation | 0<br>1 | no<br>yes |
| 11 | band 11 saturation | 0<br>1 | no<br>yes |

Table 6: radsatl_qa quality flags

We now write a function `unravelQA_l2_rad` in R to implement this decoding.

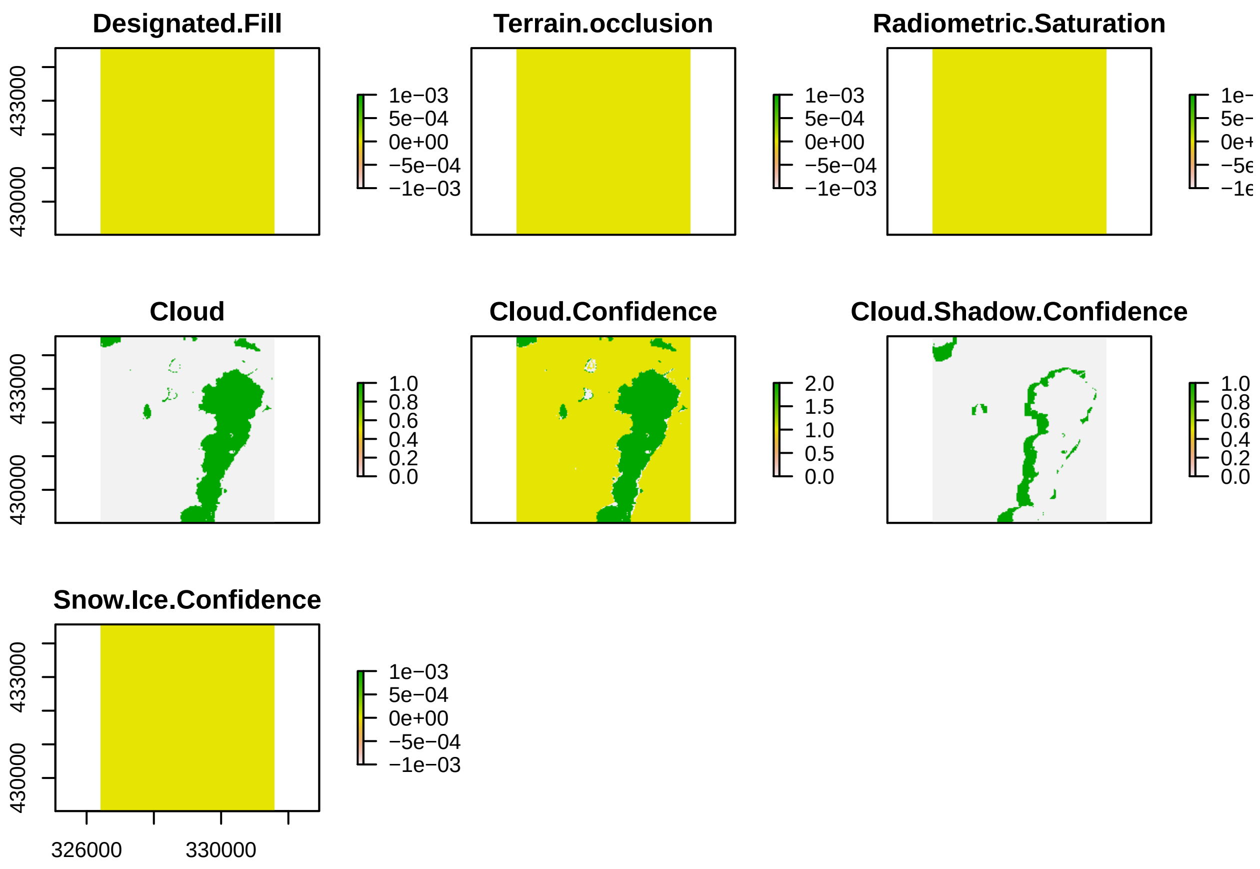
```
unravelQA_l2_rad <- function(sceneQA.r) {

    # transform integer values to 16 bits
    ints <- sapply(values(sceneQA.r), first_k_bits, reverse = F)
    # fill
    fill.r <- sceneQA.r
    values(fill.r) <- as.numeric(as.factor(substr(ints, 1, 1))) -
        1
    # band 1 saturation flag
    band1.r <- sceneQA.r
    values(band1.r) <- as.numeric(as.factor(substr(ints, 2, 2))) -
        1
    # band 2 saturation flag
    band2.r <- sceneQA.r
    values(band2.r) <- as.numeric(as.factor(substr(ints, 3, 3))) -
        1
    # band 3 saturation flag
    band3.r <- sceneQA.r
    values(band3.r) <- as.numeric(as.factor(substr(ints, 4, 4))) -
        1
    # band 4 saturation flag
    band4.r <- sceneQA.r
    values(band4.r) <- as.numeric(as.factor(substr(ints, 5, 5))) -
        1
    # band 5 saturation flag
    band5.r <- sceneQA.r
    values(band5.r) <- as.numeric(as.factor(substr(ints, 6, 6))) -
        1
    # band 6 saturation flag
    band6.r <- sceneQA.r
    values(band6.r) <- as.numeric(as.factor(substr(ints, 7, 7))) -
        1
    # band 7 saturation flag
    band7.r <- sceneQA.r
    values(band7.r) <- as.numeric(as.factor(substr(ints, 8, 8))) -
        1
    # should not be used according to manual
    test.r <- sceneQA.r
    values(test.r) <- as.numeric(as.factor(substr(ints, 9, 9))) -
        1

    # since we don't use keep bands 8-11 we won't keep their
    # saturation flags

    QA.r <- stack(band1.r, band2.r, band3.r, band4.r, band5.r,
        band6.r, band7.r, test.r)
    names(QA.r) <- c("B1", "B2", "B3", "B4", "B5", "B6", "B7",
        "test")
    return(QA.r)
}
```

We apply decoding functions to obtain stacks with respective quality flags.

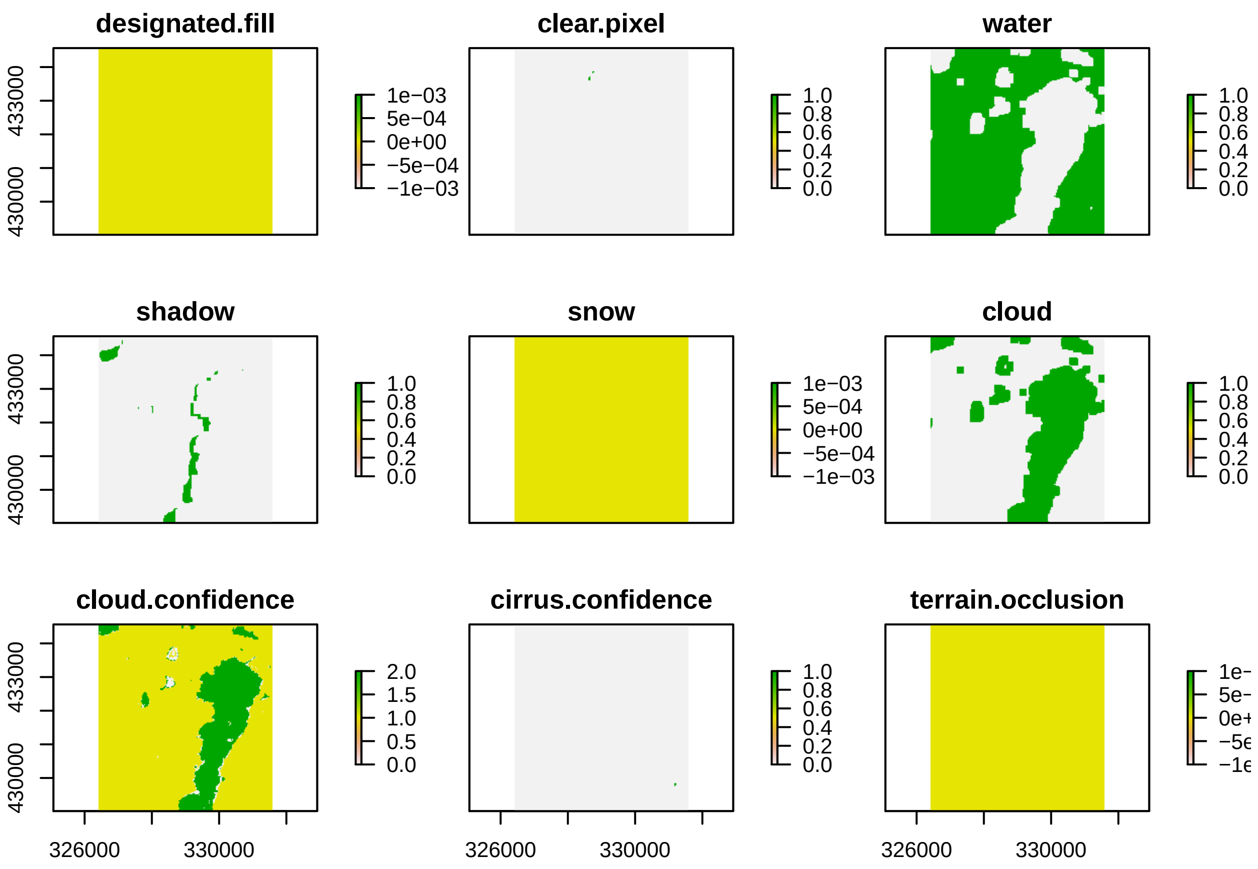
```
QA_TOAR.r <- unravelQA(scene_TOAR_QA.r)
cloudQA_SR.r <- unravelQA_l2_cld(scene_SR_cloudQA.r)
radsatQA_SR.r <- unravelQA_l2_rad(scene_SR_radsatQA.r)
```

We know plot these quality flag rasters.

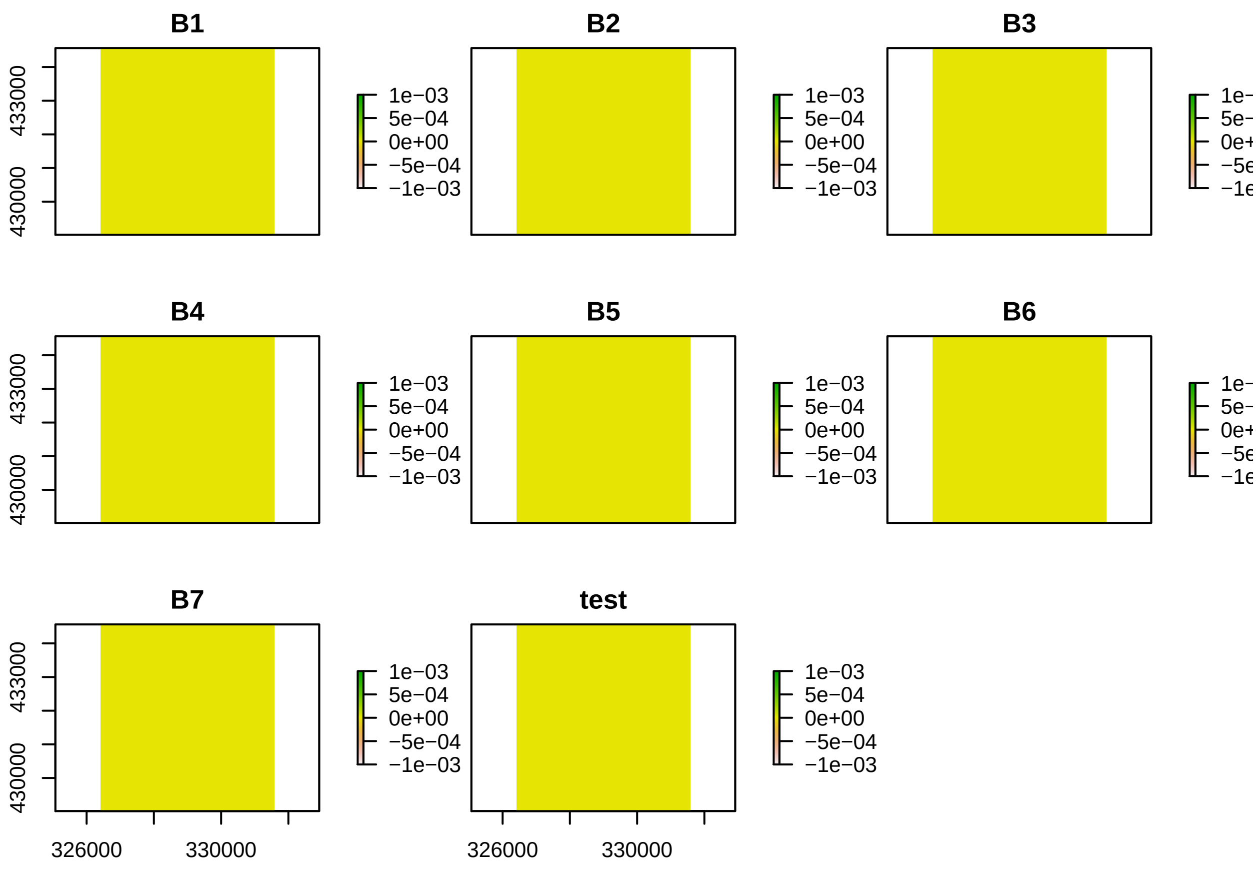
```
plot(QA_TOAR.r)
```



```r
plot(cloudQA_SR.r)
```

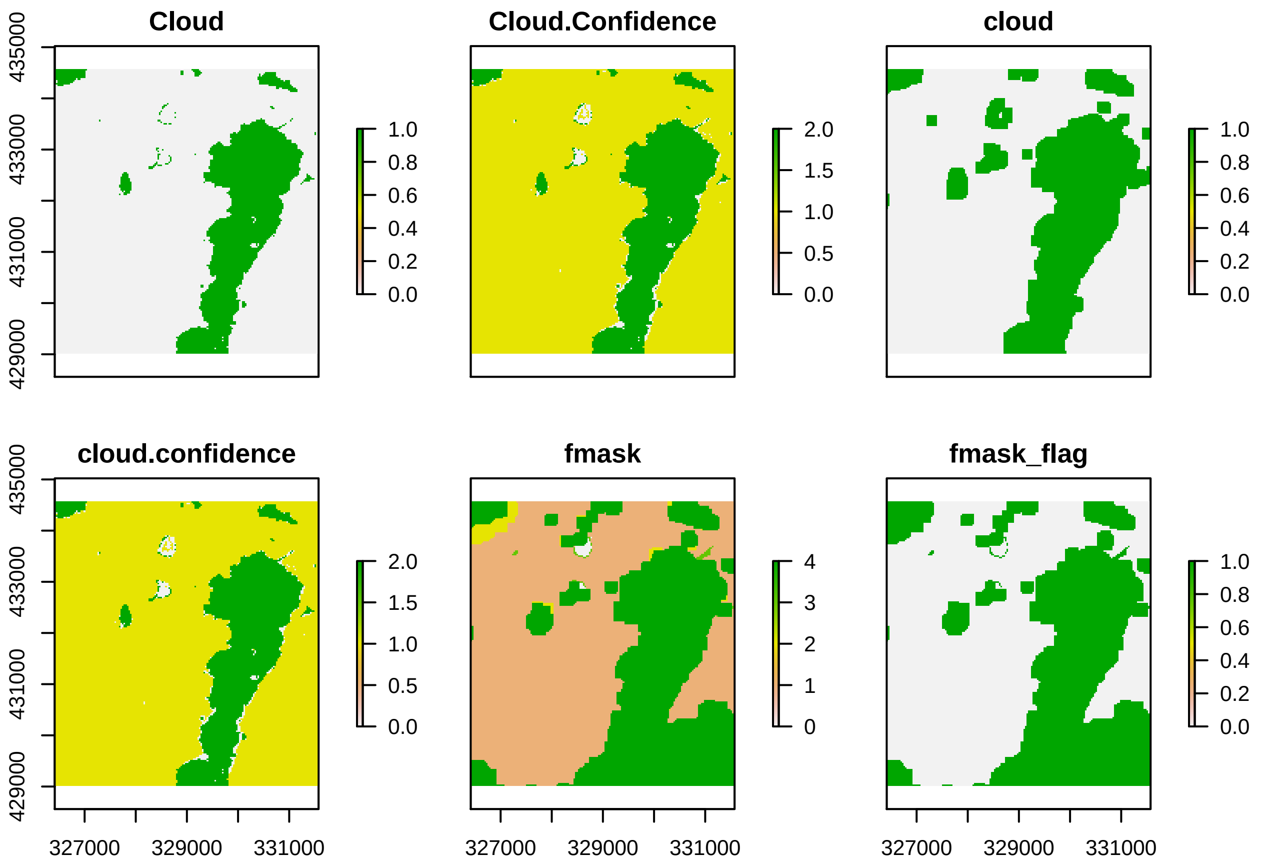


```
plot(radsatQA_SR.r)
```



We can now better compare landsat cloud flags with fmask flag.

```r
plot(stack(list(QA_TOAR.r[[c("Cloud", "Cloud.Confidence")]],
    cloudQA_SR.r[[c("cloud", "cloud.confidence")]], fmask = fmask.r,
    fmask_flag = !(fmask.r %in% c(0, 1)))))
```



We again see there are some differences between landsat cloud flags obtained using cfmask algorithm and fmask cloud flag.

We have loaded all the landsat top-of-atmosphere and surface reflectance information for the one date for our chosen study area. We now load the data for the three dates for which landsat data is provided. We do this in order to provide the code necessary to load several dates in bulk. We have previously ran this code for all 74 dates and will simply load the results, which are provided, further on. We start by obtaining the names of the directories for each of the dates which we wish to load.

```
# We read the date folders available in the appropriate
# directory using the dir() function.  In the case of TOA
# files, these also make up the first part of the TOA
# reflectance files.
dateFolders_TOAR <- dir(paste(orderFolder_TOAR, product_TOAR,
    sep = "/"))
dateFolders_SR <- dir(paste(orderFolder_SR, product_SR, sep = "/"))
# In the case of SR files we need to reorder the string in
# order to obtain the first part of the SR reflectance files
dateFolders_SR_2 <- paste(substr(dateFolders_SR, 1, 4), paste("L1",
    substr(dateFolders_SR, 50, 51), sep = ""), substr(dateFolders_SR,
    5, 10), substr(dateFolders_SR, 11, 18), substr(dateFolders_SR,
    41, 48), substr(dateFolders_SR, 19, 20), substr(dateFolders_SR,
    21, 22), sep = "_")
```

We now obtain the the dates for which we have top-of-atmosphere and surface reflectance information.

```
# we isolate the date from the folder string
# top-of-atmosphere
dates_TOAR <- as.Date(sapply(strsplit(dateFolders_TOAR, "_"),
    function(x) x)[4, ], format = "%Y%m%d")
indx <- order(dates_TOAR)
dateFolders_TOAR <- dateFolders_TOAR[indx]
dates_TOAR <- dates_TOAR[indx]
# surface
dates_SR <- as.Date(substr(dateFolders_SR, 11, 18), format = "%Y%m%d")
indx <- order(dates_SR)
dateFolders_SR <- dateFolders_SR[indx]
dates_SR <- dates_SR[indx]

# we will use dates for which there is both top-of-atmosphere
# and surface reflectance available
dates <- as.Date(intersect(dates_TOAR, dates_SR), origin = as.Date("1970-01-01"))
```

Before looping through the dates and loading the pixelwise reflectance information we can already load certain scenewise information which is encoded in the folder names. The folder and file naming conventions can be found at https://landsat.usgs.gov/what-are-naming-conventions-landsat-scene-identifiers in the case of level-1 data and on page 17 of https://landsat.usgs.gov/sites/default/files/documents/lasrc__product__guide.pdf in the case of level-2 data.

```
# we obtain the indices relating the dates we will use to the
# folder string names
indx_TOAR <- match(dates, dates_TOAR)
indx_SR <- match(dates, dates_SR)

# we obtain the satellite-sensor (satSens), processing
# correction level or tier (procLevel), and the collection
# number and category (colCat) available for each date
# top-of-atmosphere
satSens_TOAR <- substr(dateFolders_TOAR[indx_TOAR], 1, 4)
procLevel_TOAR <- substr(dateFolders_TOAR[indx_TOAR], 8, 9)
colCat_TOAR <- paste("01", substr(dateFolders_TOAR[indx_TOAR],
    39, 40), sep = "")  #we only used collection 1
# surface
satSens_SR <- substr(dateFolders_SR[indx_SR], 1, 4)
procLevel_SR <- substr(dateFolders_SR[indx_SR], 50, 51)
colCat_SR <- paste("01", substr(dateFolders_SR[indx_SR], 21,
    22), sep = "")
# we make sure that the TOA variables are the same as the SR
# variables
all(satSens_TOAR == satSens_SR)
```

```
## [1] TRUE
```

```
all(procLevel_TOAR == procLevel_SR)
```

```
## [1] TRUE
```

```
all(colCat_TOAR == colCat_SR)
```

```
## [1] TRUE
```

```
# since they are all the same we keep datewise
# satellite-sensor, tier and collection number-category
# variables.
satSens <- satSens_SR
remove(satSens_TOAR, satSens_SR)
procLevel <- procLevel_SR
remove(procLevel_TOAR, procLevel_SR)
colCat <- colCat_SR
remove(colCat_TOAR, colCat_SR)
```

We will now loop through all the dates loading the corresponding landsat top-of-atmosphere and surface reflectance information as we did in the case of one date. We will not store it in raster/stack objects since they only provide three dimensions: two spatial dimensions and one for time and bands. We prefer to store it in four-dimensional arrays so that we may have two spatial dimensions, a time-dimension and a spectral band dimension.

```
# load abind library which contains abind() function allowing
# arrays to be bound along a chosen dimension
library(abind)

# we loop through available dates
for (i in 1:length(dates)) {

    # we form strings with filenames of different bands including
    # complete path
    files_TOAR <- paste(dateFolders_TOAR[indx_TOAR[i]], "_",
        bands_TOAR[-c(8)], ".TIF", sep = "")
    files_SR <- paste(dateFolders_SR_2[indx_SR[i]], "_", bands_SR[-c(8,
        9)], ".tif", sep = "")
    file_TOAR_QA <- paste(dateFolders_TOAR[indx_TOAR[i]], "_",
        bands_TOAR[8], ".TIF", sep = "")
    file_SR_cloudQA <- paste(dateFolders_SR_2[indx_SR[i]], "_",
        bands_SR[8], ".tif", sep = "")
    file_SR_radsatQA <- paste(dateFolders_SR_2[indx_SR[i]], "_",
        bands_SR[9], ".tif", sep = "")
    fileFmask <- paste(dateFolders_TOAR[indx_TOAR[i]], "_MTLFmask",
        sep = "")
    file_TOAR_MTL <- paste(dateFolders_TOAR[indx_TOAR[i]], "_MTL.txt",
        sep = "")
    file_SR_MTL <- paste(dateFolders_SR_2[indx_SR[i]], "_MTL.txt",
        sep = "")

    files_TOAR <- paste(orderFolder_TOAR, product_TOAR, dateFolders_TOAR[indx_TOAR[i]],
        files_TOAR, sep = "/")
    files_SR <- paste(orderFolder_SR, product_SR, dateFolders_SR[indx_SR[i]],
        files_SR, sep = "/")
    file_TOAR_QA <- paste(orderFolder_TOAR, product_TOAR, dateFolders_TOAR[indx_TOAR[i]],
        file_TOAR_QA, sep = "/")
    file_SR_cloudQA <- paste(orderFolder_SR, product_SR, dateFolders_SR[indx_SR[i]],
        file_SR_cloudQA, sep = "/")
    file_SR_radsatQA <- paste(orderFolder_SR, product_SR, dateFolders_SR[indx_SR[i]],
        file_SR_radsatQA, sep = "/")
    fileFmask <- paste(orderFolder_TOAR, product_TOAR, dateFolders_TOAR[indx_TOAR[i]],
        fileFmask, sep = "/")
```

```
file_TOAR_MTL <- paste(orderFolder_TOAR, product_TOAR, dateFolders_TOAR[indx_TOAR[i]],
    file_TOAR_MTL, sep = "/")
file_SR_MTL <- paste(orderFolder_SR, product_SR, dateFolders_SR[indx_SR[i]],
    file_SR_MTL, sep = "/")

# load files into raster/stack objects using stack
# functionfrom raster package bands 1-7
aux_scene_TOAR.r <- stack(files_TOAR)
names(aux_scene_TOAR.r) <- bands_TOAR[-c(8)]
aux_scene_SR.r <- stack(files_SR)
names(aux_scene_SR.r) <- bands_TOAR[-c(8)]
# quality bands
aux_scene_TOAR_QA.r <- raster(file_TOAR_QA)
aux_scene_SR_cloudQA.r <- raster(file_SR_cloudQA)
aux_scene_SR_radsatQA.r <- raster(file_SR_radsatQA)
# fmask band
aux_fmask.r <- raster(read.ENVI(fileFmask, headerfile = paste(fileFmask,
    ".hdr", sep = "")), crs = CRS(projection(aux_scene_TOAR.r)),
    template = aux_scene_TOAR.r)

# load scenewise acquisition date including time images were
# captured
aux_MTL_TOAR <- readMeta(file_TOAR_MTL, raw = FALSE)
aux_TIME_TOAR <- aux_MTL_TOAR$ACQUISITION_DATE
aux_MTL_SR <- readMeta(file_SR_MTL, raw = FALSE)
aux_TIME_SR <- aux_MTL_SR$ACQUISITION_DATE

# crop scenes to our chosen 5.35km^2 area to avoid loading
# too much information to memory
aux_scene_TOAR.r <- crop(aux_scene_TOAR.r, myExtent)
aux_scene_SR.r <- crop(aux_scene_SR.r, myExtent)
aux_scene_TOAR_QA.r <- crop(aux_scene_TOAR_QA.r, myExtent)
aux_scene_SR_cloudQA.r <- crop(aux_scene_SR_cloudQA.r, myExtent)
aux_fmask.r <- crop(aux_fmask.r, myExtent)
aux_scene_SR_radsatQA.r <- crop(aux_scene_SR_radsatQA.r,
    myExtent)
# wait until after cropping to recode fill values to make it
# faster
values(aux_fmask.r)[which(values(aux_fmask.r) == 255)] <- NA

# decode quality bands into easily interpretable quality
# flags
aux_QA_TOAR.r <- unravelQA(aux_scene_TOAR_QA.r)
aux_cloudQA_SR.r <- unravelQA_l2_cld(aux_scene_SR_cloudQA.r)
aux_radsatQA_SR.r <- unravelQA_l2_rad(aux_scene_SR_radsatQA.r)


# transform raster objects into 3D arrays
aux_scene_TOAR.ar <- as.array(aux_scene_TOAR.r)
aux_scene_SR.ar <- as.array(aux_scene_SR.r)
aux_fmask.ar <- as.array(aux_fmask.r)
aux_QA_TOAR.ar <- as.array(aux_QA_TOAR.r)
aux_cloudQA_SR.ar <- as.array(aux_cloudQA_SR.r)
```

```r
    aux_radsatQA_SR.ar <- as.array(aux_radsatQA_SR.r)

    # bind arrays along fourth time dimension if its the first
    # date we simply initialize final arrays
    if (i == 1) {
        scene_TOAR.ar <- aux_scene_TOAR.ar
        scene_SR.ar <- aux_scene_SR.ar
        fmask.ar <- aux_fmask.ar
        QA_TOAR.ar <- aux_QA_TOAR.ar
        cloudQA_SR.ar <- aux_cloudQA_SR.ar
        radsatQA_SR.ar <- aux_radsatQA_SR.ar

        TIME_TOAR <- aux_TIME_TOAR
        TIME_SR <- aux_TIME_SR

        # for second date onwards we bind current 3d array to 4d
        # array holding previous dates along the fourth time
        # dimension
    } else {
        scene_TOAR.ar <- abind(scene_TOAR.ar, aux_scene_TOAR.ar,
            along = 4)
        scene_SR.ar <- abind(scene_SR.ar, aux_scene_SR.ar, along = 4)
        fmask.ar <- abind(fmask.ar, aux_fmask.ar, along = 4)
        QA_TOAR.ar <- abind(QA_TOAR.ar, aux_QA_TOAR.ar, along = 4)
        cloudQA_SR.ar <- abind(cloudQA_SR.ar, aux_cloudQA_SR.ar,
            along = 4)
        radsatQA_SR.ar <- abind(radsatQA_SR.ar, aux_radsatQA_SR.ar,
            along = 4)

        # acquisition time is simply bound in a vector
        TIME_TOAR <- c(TIME_TOAR, aux_TIME_TOAR)
        TIME_SR <- c(TIME_SR, aux_TIME_SR)
    }

    # print dimension of array to keep track of stacking progress
    print(dim(scene_TOAR.ar))
}
```

```
## [1] 185 172   7
## [1] 185 172   7   2
## [1] 185 172   7   3
```

```r
# once we have finished looping through dates we name the
# dimensions and dimension values of arrays as this will
# facilitate their processing and manipulation later on
coords <- as.data.frame(coordinates(aux_scene_TOAR.r))
dimnames(scene_TOAR.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = names(scene_TOAR.r), date = as.character(dates))
dimnames(scene_SR.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = names(scene_SR.r), date = as.character(dates))
dimnames(QA_TOAR.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = names(QA_TOAR.r), date = as.character(dates))
dimnames(cloudQA_SR.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = names(cloudQA_SR.r), date = as.character(dates))
```

```
dimnames(radsatQA_SR.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = names(radsatQA_SR.r), date = as.character(dates))
dimnames(fmask.ar) <- list(y = unique(coords$y), x = unique(coords$x),
    var = "fmask_class", date = as.character(dates))

# we make sure that the TOA time is the same as the SR time
all(TIME_TOAR == TIME_SR)
```

```
## [1] TRUE
```

```
# since they are the same we keep datewise time of
# acquisition
TIME <- TIME_SR
remove(TIME_TOAR, TIME_SR)
```

We now load the top of atmosphere and surface reflectance arrays with all 74 files which we have obtained earlier.

```
# Load all dates of TOA, SR and Fmask data arrays for chosen
# area and dates
load(file = "../data/landsat/croppedScene/arrays74dates_atoll.RData")
dates <- as.Date(dimnames(scene_TOAR.ar)$date)
```

### 2.1.6 Study period

We know we have 74 dates available. The landsat passes over every scene on earth once every 16 days so that should be the frequency, however errors or malfunctions can cause information for some dates to be lost or never captured. Lets see the frequency and distribution of the dates we have obtained.

```
# number of dates
length(dates)  # 74 dates
```

```
## [1] 74
```

```
# range of dates
summary(dates)  # From 2013-05-22 to 2017-06-02
```

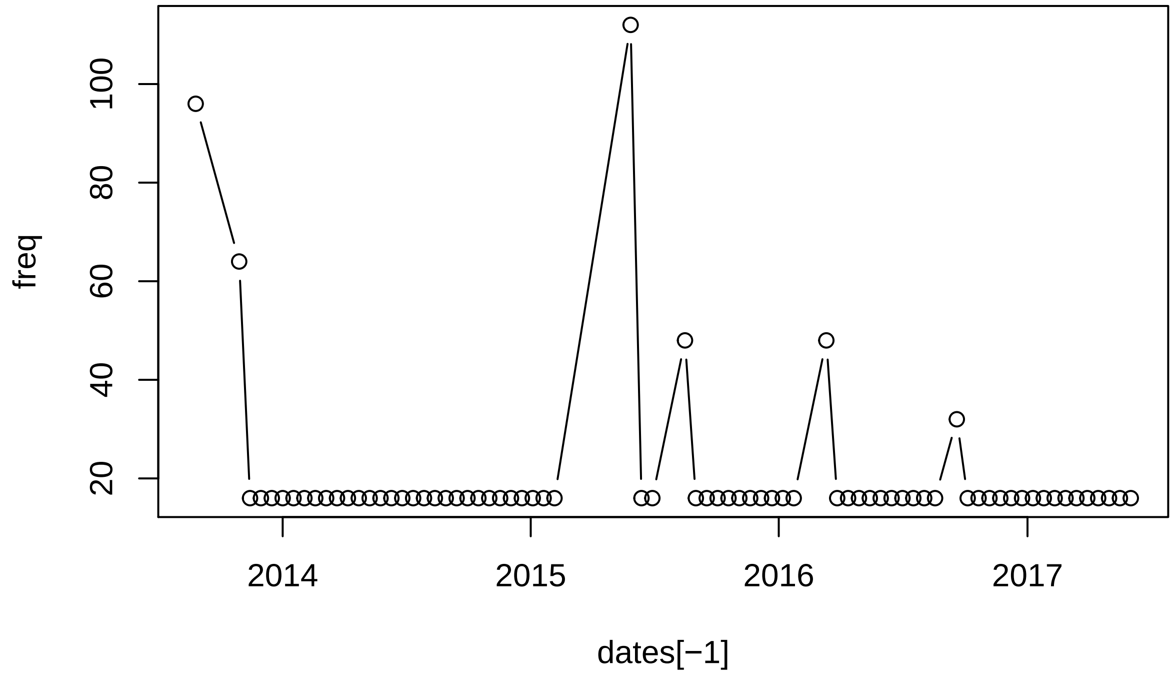
```
##         Min.      1st Qu.       Median         Mean      3rd Qu.
## "2013-05-22" "2014-07-16" "2015-09-09" "2015-07-27" "2016-07-29"
##         Max.
## "2017-06-02"
```

```
# obtain the frequency of observations
freq <- dates[2:length(dates)] - dates[1:(length(dates) - 1)]
# range of frequencies
summary(as.numeric(freq))  # max period between dates is 112
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16.00   16.00   16.00   20.16   16.00  112.00
```

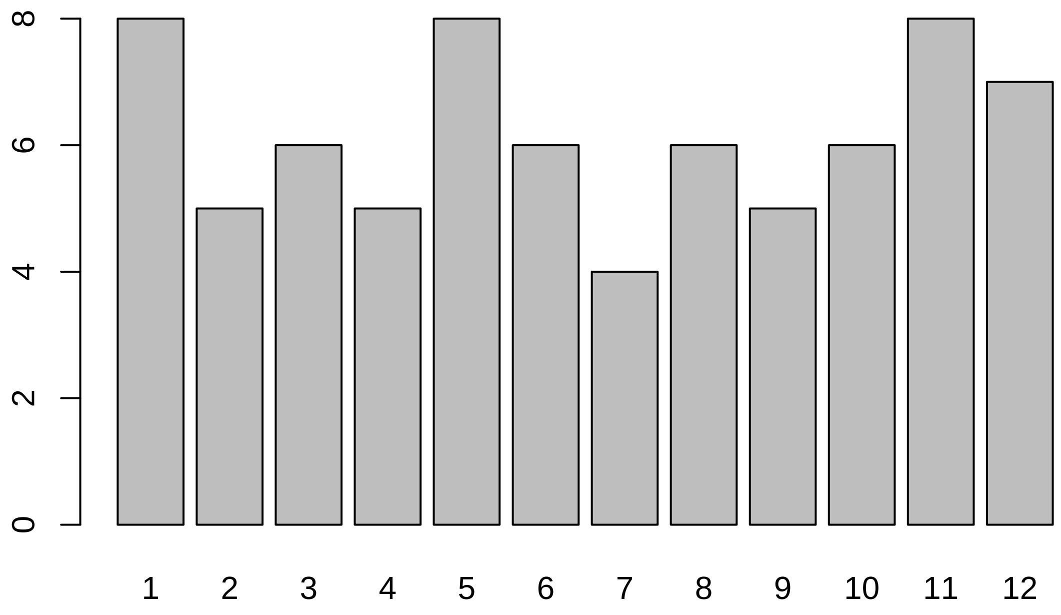
Lets plot the frequency of reflectance observations for the study period between 2013-05-22 to 2017-06-02.

```
plot(dates[-1], freq, type = "b")
```



We can see that the frequency of observations is mostly 16 days however sometime during 2015 there 112 days passed between observations. This is possibly due to some kind of technical error. Lets see the distribution of observations accross the months of the year.

```r
# Use function month from lubridate package to obtain the
# month from a date
library(lubridate)
barplot(table(month(dates)))
```



We see that the observations are fairly evenly distributed accross the months of the year.

## 3 Change point analysis

In this section we will perform the analysis on the surface reflectance of the study period and study area chosen in order to see if there is any evidence of a bleaching event. We will proceed as follows:

1. Build a historical pixelwise database of reflectance using previously loaded data.
2. Model the surface reflectance of each pixel as a function of the time of year and the tidal cycles. Use forward selection to choose the relevant seasonal and tidal factors.
3. Using surface reflectance model apply change point analysis independently to each pixel to determine the most likely date of a change occurring at the site of each pixel.
4. Determine the most likely date of bleaching event affecting the entire study area.
5. Produce a map of likely bleached areas within study area.

We have 74 dates of reflectance information for our study area. We'll first visualize this information in the format that makes most sense: as a RGB video. We use the `imager` package because its `cimg` class allows for four dimensions - two space dimensions, a time dimension and a spectral band dimension - and because it provides useful plotting and processing methods for this class.

```r
# Load imager library and transform top-of-atmosphere
# reflectance array into cimg
library(imager)
# we only use the RGB bands
sceneRGB.cimg <- cimg(aperm(scene_TOAR.ar[, , c(4, 3, 2), ],
    c(1, 2, 4, 3)))
dim(sceneRGB.cimg)
```

```
## [1] 185 172  74   3
```

```r
depth(sceneRGB.cimg)  # the depth is the time dimension
```

```
## [1] 74
```

We make a video using 74 date observations and play it. You need to have `ffmpeg` on your path for this to work. You may skip this part as it only serves to play video.

```r
# filename and route where we want to save
f <- "../Rmd/sceneRGB.mp4"
# make and save the vido with save.video
save.video(sceneRGB.cimg, f, fps = 2)
play(load.video(f), delay = 500, loop = T)
```

Figure 22: Animation of study area accross study period

We now *reshape* the reflectance and quality arrays into a single database with long format which will allow us to isolate the information of specific pixels and pixel-types (eg. shallow reef pixels, cloudy pixels). we use the `melt` function from the `reshape` package to do this.

```r
# Load reshape package
library(reshape)
# Make top-of-atmosphere and surface reflectance databases
scene_TOAR.df <- apply(scene_TOAR.ar, "var", melt)
scene_TOAR.df <- cbind(scene_TOAR.df[[1]][, c("x", "y", "date")],
    do.call(cbind, lapply(scene_TOAR.df, function(df) df[, 4])))
scene_SR.df <- apply(scene_SR.ar, "var", melt)
scene_SR.df <- cbind(scene_SR.df[[1]][, c("x", "y", "date")],
    do.call(cbind, lapply(scene_SR.df, function(df) df[, 4])))
```

```r
# Make BQA quality and pixel_qa quality band databases
QA_TOAR.df <- apply(QA_TOAR.ar, "var", melt)
QA_TOAR.df <- cbind(QA_TOAR.df[[1]][, c("x", "y", "date")], do.call(cbind,
    lapply(QA_TOAR.df, function(df) df[, 4])))
cloudQA_SR.df <- apply(cloudQA_SR.ar, "var", melt)
cloudQA_SR.df <- cbind(cloudQA_SR.df[[1]][, c("x", "y", "date")],
    do.call(cbind, lapply(cloudQA_SR.df, function(df) df[, 4])))

# Make an fmask database
fmask.df <- apply(fmask.ar, "var", melt)
fmask.df <- cbind(fmask.df[[1]][, c("x", "y", "date")], do.call(cbind,
    lapply(fmask.df, function(df) df[, 4])))

# Make a general database by binding columns and remove
# individual databases
scene.df <- cbind(scene_TOAR.df, scene_SR.df[, 4:ncol(scene_SR.df)],
    QA_TOAR.df[, 4:ncol(QA_TOAR.df)], cloudQA_SR.df[, 4:ncol(cloudQA_SR.df)],
    fmask_class = fmask.df$fmask_class)
colnames(scene.df)[4:(4 + 6)] <- paste("toar", colnames(scene.df)[4:(4 +
    6)], sep = ".")
colnames(scene.df)[(4 + 7):(4 + 7 + 6)] <- paste("sr", colnames(scene.df)[(4 +
    7):(4 + 7 + 6)], sep = ".")
remove(QA_TOAR.df)
remove(cloudQA_SR.df)
remove(fmask.df)

# Make a coral-reef database by using the coral.r stack we
# made earlier by rasterizing maldives shape file
coral.df <- as.data.frame(coral.r, xy = T)

# Add this to general database by merging with respect to
# coordinates
scene.df <- merge(scene.df, coral.df, by = c("x", "y"), sort = F)
# make coralType variable easier to read by numbers to labels
# (rasters can't handle strings but data.frames can!)
scene.df$coralType <- c("deep_reef", "shallow_reef", "variable_depth_reef")[scene.df$coralType]
scene.df$coralType[which(is.na(scene.df$coralType))] <- "non-reef"
# Add pixel, pixel-row, pixel-column, year, month and day
# variables
scene.df$pixel <- cellFromXY(scene_TOAR.r, scene.df[, c("x",
    "y")])
scene.df$row <- rowFromCell(scene_TOAR.r, scene.df$pixel)
scene.df$col <- colFromCell(scene_TOAR.r, scene.df$pixel)
scene.df$date <- as.Date(scene.df$date)
scene.df$year <- year(scene.df$date)
scene.df$month <- month(scene.df$date)
scene.df$day <- as.POSIXlt(scene.df$date)$yday + 1

# add scenewise variables (satellite-sensor, tier, collection
# number-category and acquisition time) by matching data base
# date to scene dates
indx <- match(scene.df$date, dates)
scene.df$satSens <- satSens[indx]
scene.df$procLevel <- procLevel[indx]
scene.df$colCat <- colCat[indx]
scene.df$time <- TIME[indx]
library(pander)  # pander
# pander(head(scene.df), caption='General database format')
```

Lets see how many observations are affected by cloud shadow (fmask=2), snow (fmask=3), clouds (fmask=4) or radiometric saturation.

```r
pander(table(scene.df$fmask_class %in% c(2, 3, 4))/nrow(scene.df) *
    100, caption = "% of cloud/snow/shadow pixels")
```

Table 7: % of cloud/snow/shadow pixels

| FALSE | TRUE |
|---|---|
| 35.45 | 64.55 |

```r
pander(table(scene.df$Radiometric.Saturation > 0)/nrow(scene.df) *
    100, caption = "% of radiometric saturated pixels")
```

Table 8: % of radiometric saturated pixels

| FALSE |
|---|
| 100 |

We know create a variable indicating whether we can use a particular observation or not.

```r
scene.df$filter <- 0
scene.df$filter[which(scene.df$fmask_class %in% c(2, 3, 4) |
    scene.df$Radiometric.Saturation > 0)] <- 1
pander(table(scene.df$filter)/nrow(scene.df) * 100, caption = "% usable pixel observations")
```

Table 9: % usable pixel observations

| 0 | 1 |
|---|---|
| 35.45 | 64.55 |

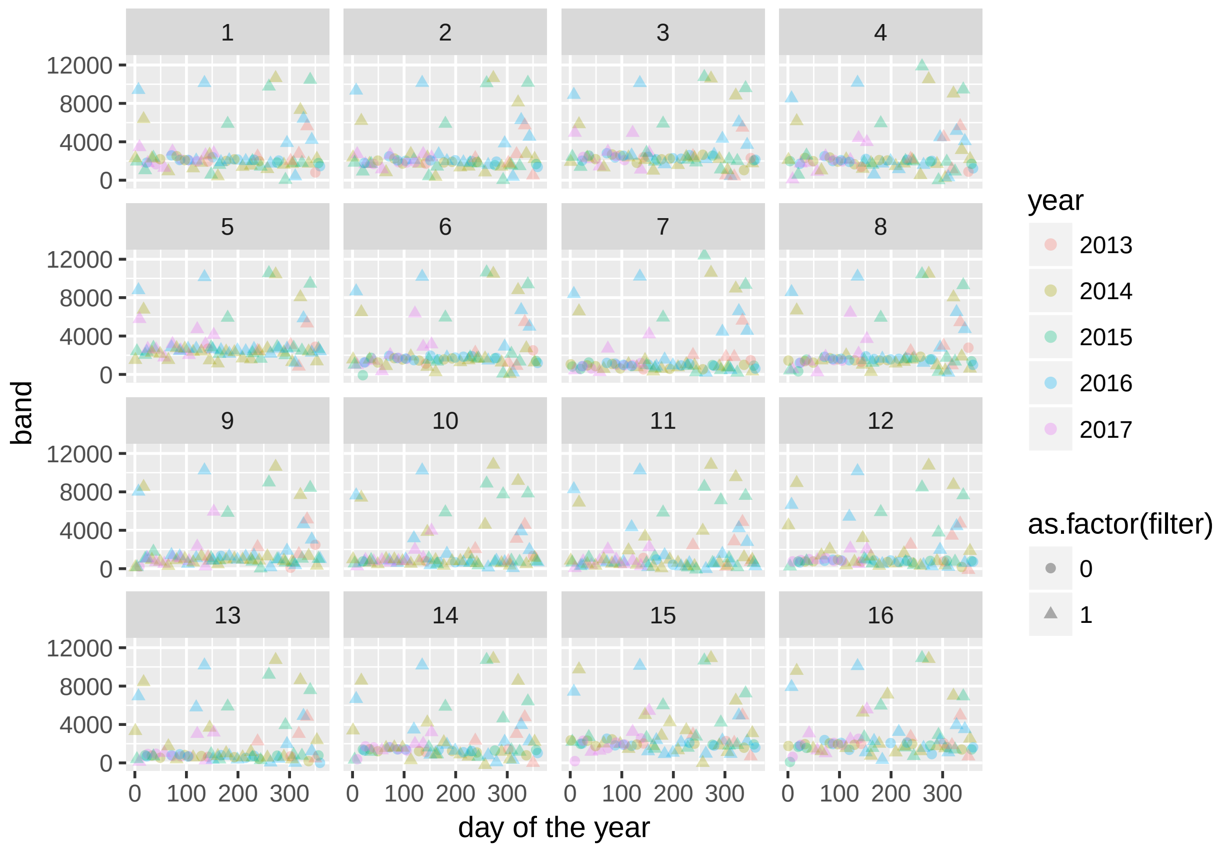
We want to display the reflectance of a few pixels accross time in order to get a feel for its behavior. We first sample a few shallow-reef and non-reef (water) pixels.

```r
# identify shallow-reef pixels
pixels.shallow <- which(values(coral.r[["coralType"]]) == 2 &
    values(coral.r[["distanceWater"]]) > 120)
# identify non-reef pixels
pixels.water <- which(is.na(values(coral.r[["coralType"]])))

num.smpl <- 16
set.seed(8)
# Sample 16 shallow-reef and 16 non-reef pixels
smpl.shallow <- sample(pixels.shallow, num.smpl)
smpl.water <- sample(pixels.water, num.smpl)
```

```r
smpl <- c(smpl.shallow, smpl.water)
```

We plot the surface reflectance of the ultra-blue band for the 16 sampled shallow-reef pixels. We do not filter out cloudy pixels but they are identified in plot.

```r
library(ggplot2)
# B1 - Ultra blue, shallow-reef, UNfiltered
p <- ggplot(scene.df[which(scene.df$pixel %in% smpl & scene.df$coralType ==
    "shallow_reef"), ])
p <- p + geom_point(aes(x = day, y = sr.B1, shape = as.factor(filter),
    colour = as.factor(year)), alpha = 0.3)
p <- p + facet_wrap(~pixel, ncol = 4, labeller = as_labeller(labels))
p <- p + xlab("day of the year") + ylab("band") + scale_colour_discrete(name = "year")
p
```



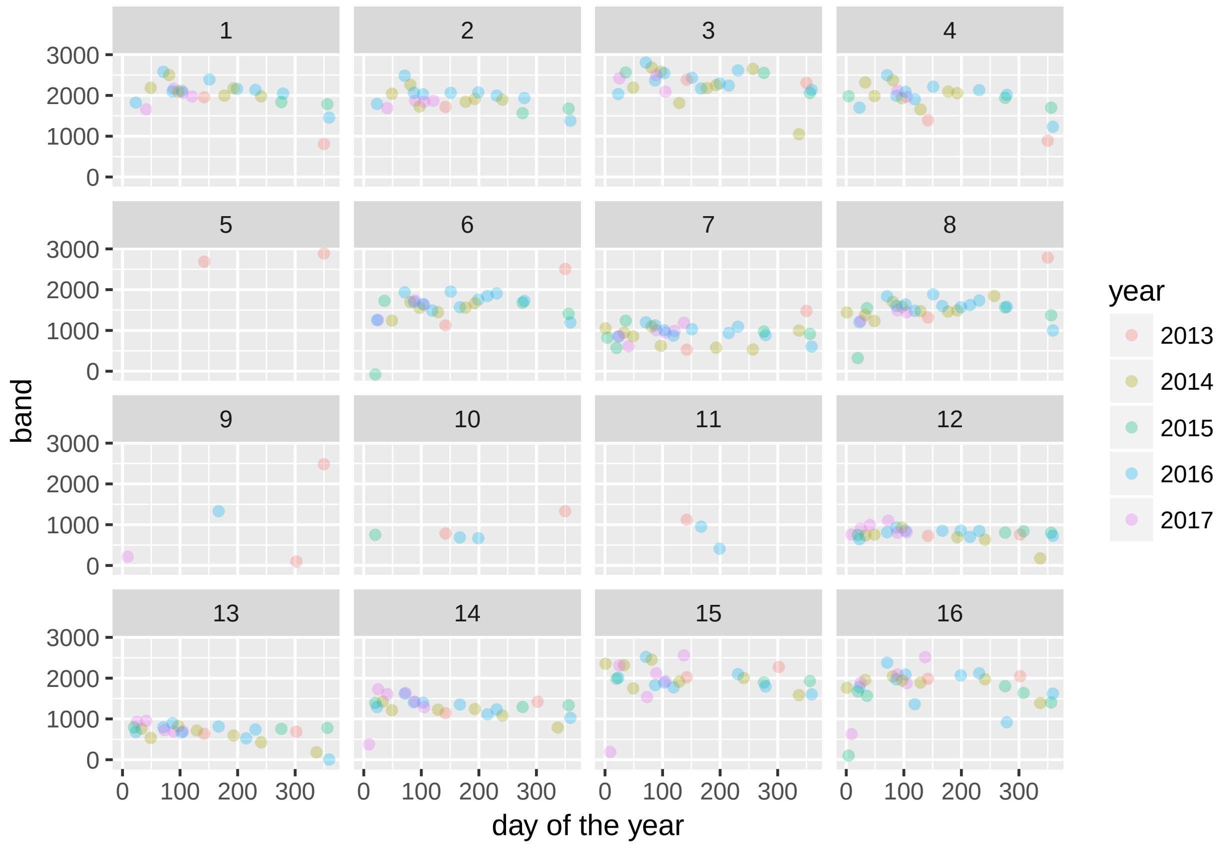
As we can see cloudy pixels produce very high reflectance values which distort the scale of the plot and do not allow us to observe the surface reflectance behavior of the pixels. Lets filter out these cloudy pixels.

```r
# take out all cloud and radiometric saturation data points
scene.df <- scene.df[which(scene.df$filter == 0), ]
```

We repeat plot having filtered out cloudy pixels.

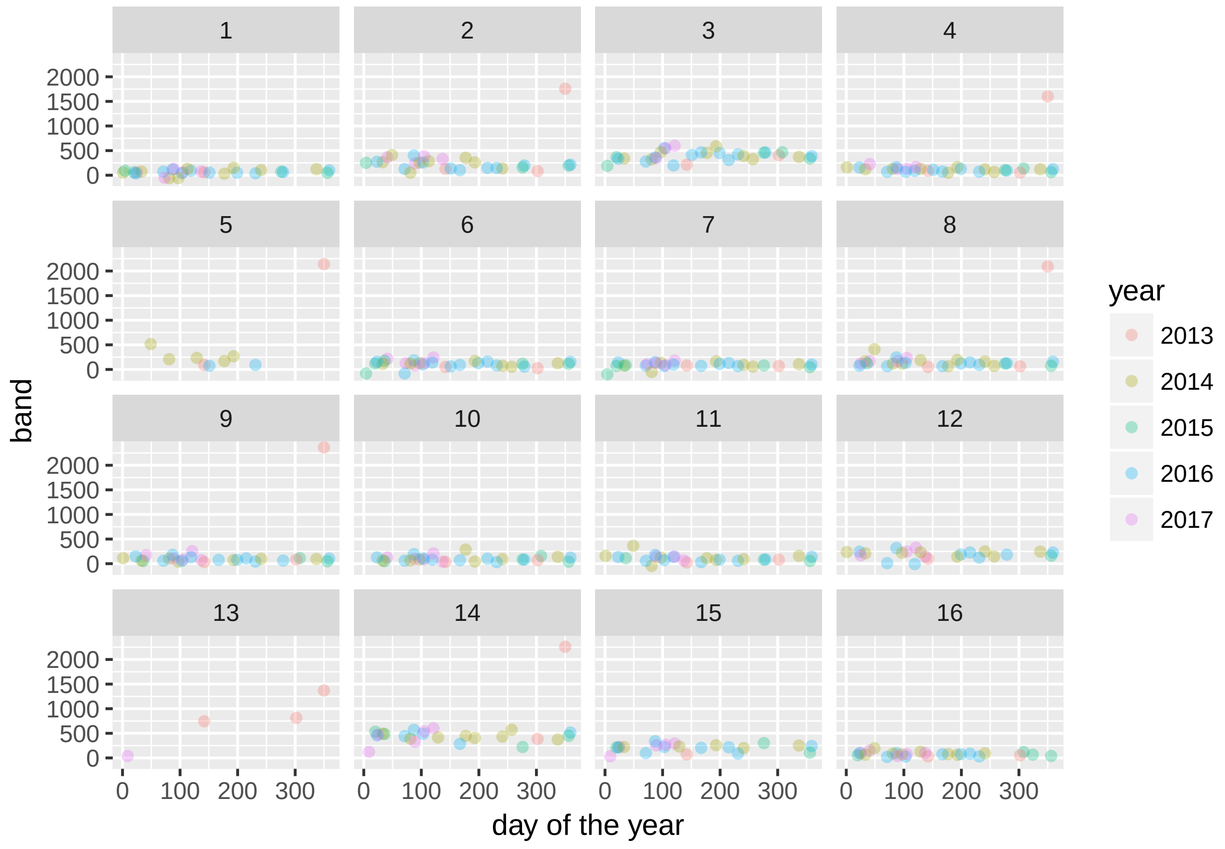
```r
# B1 - Ultra blue, shallow-reef, filtered
p <- ggplot(scene.df[which(scene.df$pixel %in% smpl & scene.df$coralType ==
    "shallow_reef"), ])
p <- p + geom_point(aes(x = day, y = sr.B1, colour = as.factor(year)),
    alpha = 0.3)
```

```
p <- p + facet_wrap(~pixel, ncol = 4, labeller = as_labeller(labels))
p <- p + xlab("day of the year") + ylab("band") + scale_colour_discrete(name = "year")
p
```



We can see, especially for certain pixels such as pixel 14, reflectance displays yearly seasonality. We now plot the surface reflectance of the green band for the 16 sampled non-reef pixels.

```
# B3 - Green, non-reef, filtered
p <- ggplot(scene.df[which(scene.df$pixel %in% smpl & scene.df$coralType ==
    "non-reef"), ])
p <- p + geom_point(aes(x = day, y = sr.B3, colour = as.factor(year)),
    alpha = 0.3)
p <- p + facet_wrap(~pixel, ncol = 4, labeller = as_labeller(labels))
p <- p + xlab("day of the year") + ylab("band") + scale_colour_discrete(name = "year")
p
```



## 3.1 Pixelwise fourier model

We have already seen that the surface reflectance displays yearly seasonality. We know that since the areas captured have relatively shallow water, tide will be changing and will affect the reflectance recorded. Tide depends mainly on fourier components of the type $a * \sin(\frac{2\pi h}{P})$ where $P$ is the period of the tide cycle and $h$ the hours passed within that cycle. We will fit following model to the ultra-blue band surface reflectance of each pixel:

$$\begin{aligned} r = g(d,h) + \epsilon = \alpha_0 + \alpha_1 \cos\left(\frac{2\pi d}{T}\right) + \beta_1 \sin\left(\frac{2\pi d}{T}\right) \\ + \alpha_2 \cos\left(\frac{2\pi d}{0.5T}\right) + \beta_2 \sin\left(\frac{2\pi d}{0.5T}\right) \\ \sum_{i=1}^{N} + \alpha_{i+2} \cos\left(\frac{2\pi h_i}{P_i}\right) + \beta_{i+2} \sin\left(\frac{2\pi h_i}{P_i}\right) + \epsilon \end{aligned} \tag{1}$$

Where:

- $r$ is ultra-blue surface reflectance,
- $d \in \{1, 2, ..., 366\}$ is the day of the year,
- $h_i \in [0, P_i]$ is the hours that have passed in tide cycle $i \in \{1, ..., N\}$
- $T$ is the number of days in the year, taken to be 366,
- $P_i$ is the length in hours of tide cycle $i$

- Coefficients $\alpha_1$ and $\beta_1$ represent variation that occurs yearly cycles ,
- Coefficients $\alpha_2$ and $\beta_2$ capture bimodal variations,
- Coefficients $\alpha_j$ and $\beta_j$ for $j > 2$ represent variation due to tidal forces,
- Coefficient $\alpha_0$ represents mean overall surface reflectance, and
- $\epsilon$ is the error term.

We chose the ultra-blue band to try to detect model bleaching because in Yamano and Tamura (2004) it was found that the lower wavelength bands were best for detecting bleaching according to their simulation model.

The main tide components have the following periods (see https://www.r-bloggers.com/predicting-tides-in-r/):

| Name | Period(hours) | relative strength % |
|---|---|---|
| Main lunar, semi diurnal | 12.42 | 100.0 |
| Lunar-solar, diurnal | 23.93 | 58.4 |
| Main solar, semi-diurnal | 12.00 | 46.6 |
| Main lunar, diurnal | 25.82 | 41.5 |
| Main solar, diurnal | 24.07 | 19.4 |
| Lunar elliptic, semi-diurnal | 12.66 | 19.2 |
| lunar-solar, semi-diurnal | 11.97 | 12.7 |

We will use these seven components ($i \in \{1, ..., N = 7\}$) for fourier model 1.

We start by building an *absolute hour* $h$ variable which will count the hours passed since 2013-01-01 00:00:00. We can then calculate the different $h_i$ variables by dividing by the respective period $P_i$ and taking the remainder:

$$h_i = \frac{h}{P_i} - \left\lfloor \frac{h}{P_i} \right\rfloor \tag{2}$$

```r
# We set the periods we are interested in
tidePeriods <- c(12.42, 23.93, 12, 25.82, 24.07, 12.66, 11.97)

# We first calculate the absolute hour, counting from
# 2013-01-01 00:00:00 base-line
class(scene.df$time)
```

```
## [1] "POSIXlt" "POSIXt"
```

```r
dts <- as.POSIXct(scene.df$time)
head(dts)
```

```
## [1] "2013-05-22 00:21:28 CEST" "2017-03-29 23:19:01 CEST"
## [3] "2016-03-26 23:19:09 CET"  "2016-08-18 00:19:32 CEST"
## [5] "2014-07-12 00:19:18 CEST" "2017-03-13 23:19:10 CET"
```

```r
attributes(dts)
```

```
## $class
## [1] "POSIXct" "POSIXt"
##
## $tzone
## [1] ""
```

```
# change the time zone of the date to a single time
attributes(dts)$tzone <- "GMT"
head(dts)
```

```
## [1] "2013-05-21 22:21:28 GMT" "2017-03-29 21:19:01 GMT"
## [3] "2016-03-26 22:19:09 GMT" "2016-08-17 22:19:32 GMT"
## [5] "2014-07-11 22:19:18 GMT" "2017-03-13 22:19:10 GMT"
```

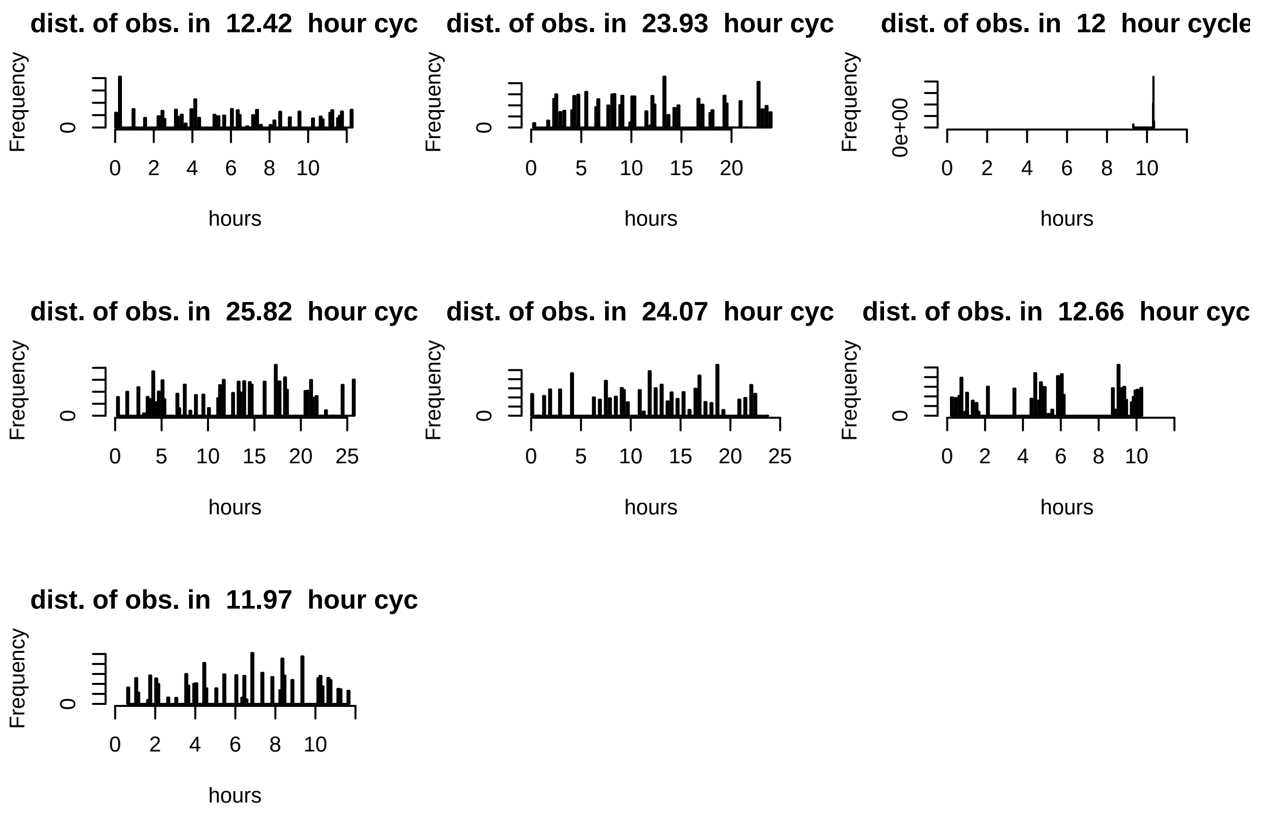
```
# isolate the time of day
tms <- format(dts, format = "%H:%M:%S")
tms <- as.numeric(substr(tms, 1, 2)) + as.numeric(substr(tms,
    4, 5))/60 + as.numeric(substr(tms, 4, 5))/3600
summary(tms)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   21.32   22.32   22.32   22.29   22.32   22.36
```

```
scene.df$time_hour <- tms

# transform to a hours since 2013-01-01 00:00:00
origin <- as.POSIXct("2013-01-01 01:00:00")
attributes(origin)$tzone <- "GMT"
dys <- dts - origin
hrs <- as.numeric(dys) * 24
scene.df$daysAbs <- dys
scene.df$hoursAbs <- hrs
```

We will now check that there is enough variance in the observations with respect to the tide cycles. We would like all tide cycles to have observations accross it.

```
# See if we have enough variance - do we have observations
# spread out accross the relevant tide cycles
distPeriod <- lapply(tidePeriods, function(f) (scene.df$hoursAbs%%f))
par(mfrow = c(3, 3))
ap <- lapply(1:length(tidePeriods), function(i) hist(distPeriod[[i]],
    100, xlab = "hours", xlim = c(0, tidePeriods[i]), main = paste("dist. of obs. in ",
        tidePeriods[i], " hour cycle")))
```



As we may have expected there is hardly any variance in observations for the 12 hour cycle. This is because landsat has a sun-synchronous orbit meaning that it captures the scene at the same time of day. We have no variance with respect to this tide cycle so we may exclude it as factor affecting surface reflectance.

```
# Exclude 12 hour tide cycle period
tidePeriods <- tidePeriods[-3]
```

We now define a function that will build all the fourier components which depend on the day of the year $d$, the absolute hour $h$ and the periods $P_i$ of the relevant tide cycles.

```
makeModelMat <- function(days, hours, tidePeriods) {

    # year cycles
    T <- 365
    # create the anual and bianual cycle fourier components
    modMat <- data.frame(cosAnual = cos((days * 2 * pi)/(T)),
        sinAnual = sin((days * 2 * pi)/(T)), cosBianual = cos((days *
            2 * pi)/(0.5 * T)), sinBianual = sin((days * 2 *
            pi)/(0.5 * T)))

    # create tidal cycle fourier components using modulus
    # operator (%%)
    aux <- sapply(tidePeriods, function(f) sin(2 * pi * (hours%%f)/f))
    colnames(aux) <- paste("sin_tide", tidePeriods, sep = "_")
    modMat <- cbind(modMat, aux)
    aux <- sapply(tidePeriods, function(f) cos(2 * pi * (hours%%f)/f))
    colnames(aux) <- paste("cos_tide", tidePeriods, sep = "_")
    modMat <- cbind(modMat, aux)

    return(modMat)
}
```

We now fit the linear regression model 1 for one pixel. We refer to *seasonal* predictors as those fourier terms depending on the day of the year and to *tidal* predictors as those depending on the hour of a tidal cycle.

```
# obtain data for pixel smpl[2]
datPix <- scene.df[which(scene.df$pixel == smpl[2]), ]
dim(datPix)
```

```
## [1] 26 52
```

```
# calculate predictors and add to response variable
# (ultra-blue surface reflectance-sr.B1)
modMat <- cbind(datPix[, c("sr.B1", "date")], makeModelMat(datPix$day,
    datPix$hoursAbs, tidePeriods))
# create formula: sr.B1 ~ seasonal predictors + tidal
# predictors
nms <- colnames(modMat)
refl <- nms[1]
seasonalPreds <- nms[3:6]
tidalPreds <- nms[7:ncol(modMat)]
preds <- nms[3:ncol(modMat)]
form <- as.formula(paste(refl, paste(preds, collapse = "+"),
    sep = "~"))
# fit model
fit <- lm(form, modMat)
pander(summary(fit)$coefficients, caption = paste("Summary of full model for pixel ",
    smpl[2]))
```

Table 11: Summary of full model for pixel 7368

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| **(Intercept)** | 2209 | 77.08 | 28.66 | 3.734e-10 |
| **cosAnual** | 551.3 | 1313 | 0.4198 | 0.6845 |
| **sinAnual** | -198 | 1218 | -0.1626 | 0.8744 |
| **cosBianual** | -408.3 | 287.2 | -1.421 | 0.1889 |
| **sinBianual** | 599.3 | 633.3 | 0.9463 | 0.3687 |
| **sin__tide__12.42** | 600.1 | 1419 | 0.4228 | 0.6824 |
| **sin__tide__23.93** | 795.5 | 4294 | 0.1853 | 0.8571 |
| **sin__tide__25.82** | 193 | 122.9 | 1.57 | 0.1508 |
| **sin__tide__24.07** | 7684 | 10602 | 0.7248 | 0.487 |
| **sin__tide__12.66** | 59.4 | 151.3 | 0.3925 | 0.7038 |
| **sin__tide__11.97** | -1155 | 1522 | -0.7589 | 0.4673 |
| **cos__tide__12.42** | -1461 | 1638 | -0.8919 | 0.3957 |
| **cos__tide__23.93** | 9223 | 17474 | 0.5278 | 0.6104 |
| **cos__tide__25.82** | -186.9 | 127.3 | -1.468 | 0.1762 |
| **cos__tide__24.07** | -6440 | 16511 | -0.39 | 0.7056 |
| **cos__tide__12.66** | -47.62 | 119 | -0.4001 | 0.6984 |
| **cos__tide__11.97** | 1119 | 943.4 | 1.186 | 0.2658 |

As we can see non of the predictors are significant. This is because we are overfitting considering we have 18 predictors and only 26 data points. We now apply stepwise selection using the Akaike Information Criterion (AIC) which balances model fit as measured by the log-likelihood by penalizing it with the number of predictors.

```
library(MASS)
# apply stepwise selection based on AIC
step <- stepAIC(fit, direction = "both", trace = F)
pander(summary(step)$coefficients, caption = paste("Summary of reduced model for pixel ",
    smpl[2]))
```

Table 12: Summary of reduced model for pixel 7368

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| **(Intercept)** | 2214 | 60.64 | 36.51 | 1.74e-14 |
| **cosAnual** | 460.3 | 344.7 | 1.335 | 0.2047 |
| **cosBianual** | -386.6 | 179.7 | -2.151 | 0.05086 |
| **sinBianual** | 804 | 201 | 4 | 0.001513 |
| **sin_tide_23.93** | 1565 | 931.7 | 1.68 | 0.1168 |
| **sin_tide_25.82** | 190.9 | 103.5 | 1.843 | 0.08821 |
| **sin_tide_24.07** | 6799 | 2590 | 2.625 | 0.02099 |
| **sin_tide_11.97** | -1644 | 552.2 | -2.977 | 0.0107 |
| **cos_tide_12.42** | -1908 | 733.5 | -2.602 | 0.02192 |
| **cos_tide_23.93** | 7241 | 3408 | 2.125 | 0.05338 |
| **cos_tide_25.82** | -159.2 | 93.88 | -1.696 | 0.1137 |
| **cos_tide_24.07** | -4284 | 2728 | -1.57 | 0.1404 |
| **cos_tide_11.97** | 740.9 | 227.9 | 3.251 | 0.006315 |

The stepwise selection reduced the number of predictors from 16 to 11 and now all predictors are statistically significant. We would like to carry out this model fit and selection for all pixels. However there are 31,815 pixels in our chosen study area so in the next section we will look to simplify the selection process.

## 3.2 Forward selection

For each pixel we will use the following steps to select a model:

- Fit the null model: $r = \alpha_0 + \epsilon$,
- Use forward selection to add one seasonal predictor, and,
- Use forward selection to add two predictors (seasonal or tidal).

The idea is to make sure at least one seasonal predictor enters the model of each pixel and two other predictors according to the AIC criterion.

```
# Fit a null model
fitNull <- lm("sr.B1~1", modMat)
fit <- fitNull
# Define scope of model: those variables elligible for
# selection.  In this case seasonal predictors.
scope <- as.formula(paste("~ ", paste(seasonalPreds, collapse = "+"),
    sep = ""))
# Obtain AIC scores of single term additions to null model
pander(test <- add1(fit, scope), caption = "AIC for single term additions to null model")
```

Table 13: AIC for single term additions to null model

| | Df | Sum of Sq | RSS | AIC |
|---|---|---|---|---|
| | NA | NA | 3032019 | 305.3 |

| | Df | Sum of Sq | RSS | AIC |
|---|---|---|---|---|
| **cosAnual** | 1 | 91628 | 2940392 | 306.5 |
| **sinAnual** | 1 | 87264 | 2944755 | 306.6 |
| **cosBianual** | 1 | 627006 | 2405013 | 301.3 |
| **sinBianual** | 1 | 536526 | 2495493 | 302.3 |

We see that the cosine bianual term obtains the biggest decrease in AIC. We add it to the model:

```r
# obtain string of best seasonal predictor
newVar <- rownames(test)[which.min(test$AIC[2:length(test$AIC)]) +
    1]
# add best seasonal predictor to model
fit <- update(fit, as.formula(paste("~ . + ", newVar, sep = "")))
pander(summary(fit), caption = "Null model + 1 seasonal predictor")
```

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| **(Intercept)** | 2312 | 62.15 | 37.2 | 9.782e-23 |
| **cosBianual** | -201.1 | 80.41 | -2.501 | 0.01959 |

Table 15: Null model + 1 seasonal predictor

| Observations | Residual Std. Error | $R^2$ | Adjusted $R^2$ |
|---|---|---|---|
| 26 | 316.6 | 0.2068 | 0.1737 |

We'll now add two terms from the full scope of predictors (excluding term already added ).

```r
# Define scope of model: those variables elligible for
# selection.  In this case all predictors.
scope <- as.formula(paste("~ ", paste(preds, collapse = "+"),
    sep = ""))
# Obtain AIC scores of single term additions to null model +
# cosBianual
test <- add1(fit, scope)
# obtain string of best predictor
(newVar <- rownames(test)[which.min(test$AIC[2:length(test$AIC)]) +
    1])
```

```
## [1] "sinBianual"
```

```r
# add best seasonal predictor to model
fit <- update(fit, as.formula(paste("~ . + ", newVar, sep = "")))
# Repeat once more
test <- add1(fit, scope)
(newVar <- rownames(test)[which.min(test$AIC[2:length(test$AIC)]) +
    1])
```

```
## [1] "sin_tide_12.66"
```

```r
fit <- update(fit, as.formula(paste("~ . + ", newVar, sep = "")))
pander(summary(fit), caption = "Final model with 3 predictors")
```

|  | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| **(Intercept)** | 2293 | 52.87 | 43.37 | 8.353e-23 |
| **cosBianual** | -199.8 | 68.13 | -2.933 | 0.007707 |
| **sinBianual** | 217.2 | 83.15 | 2.612 | 0.01593 |
| **sin_tide_12.66** | 163.6 | 74.67 | 2.19 | 0.03938 |

Table 17: Final model with 3 predictors

| Observations | Residual Std. Error | $R^2$ | Adjusted $R^2$ |
|---|---|---|---|
| 26 | 267.6 | 0.4802 | 0.4093 |

We now write a function that implements this procedure so that we can apply this to all pixels.

```
addN <- function(fitNull, seasonalPreds, preds, N) {

    scope <- as.formula(paste("~", paste(seasonalPreds, collapse = "+"),
        sep = ""))
    fit <- fitNull
    test <- add1(fit, scope)
    (newVar <- rownames(test)[which.min(test$AIC[2:length(test$AIC)]) +
        1])
    fit <- update(fit, as.formula(paste("~ . + ", newVar, sep = "")))
    scope <- as.formula(paste("~", paste(preds, collapse = "+"),
        sep = ""))

    for (i in 1:(N - 1)) {
        test <- add1(fit, scope)
        rnms <- rownames(test)[2:length(rownames(test))]
        newVar <- rnms[which.min(test$AIC[2:length(test$AIC)])]
        fit <- update(fit, as.formula(paste("~ . + ", newVar,
            sep = "")))
    }

    return(fit)
}

pander(summary(addN(fitNull, seasonalPreds, preds, 3)), caption = "Final model with 3 predictors")
```

|  | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| **(Intercept)** | 2293 | 52.87 | 43.37 | 8.353e-23 |
| **cosBianual** | -199.8 | 68.13 | -2.933 | 0.007707 |
| **sinBianual** | 217.2 | 83.15 | 2.612 | 0.01593 |
| **sin_tide_12.66** | 163.6 | 74.67 | 2.19 | 0.03938 |

Table 19: Final model with 3 predictors

| Observations | Residual Std. Error | $R^2$ | Adjusted $R^2$ |
|---|---|---|---|
| 26 | 267.6 | 0.4802 | 0.4093 |

We want to apply forward selection to all pixels so long as they have enough data.

```
# vector with all pixels
pixels <- scene.df$pixel
length(unique(pixels))  #there are 31,815 pixels
```

```
## [1] 31815
```

```
table(table(pixels))  #pixels have anywhere from 1 to 35 observations
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  422  422  273  148   61   40   38   51   39   36   45   51   89   77   76
##   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30
##   84  120  165  215  174  230  429  865 1416 2276 3299 4278 4678 4195 3354
##   31   32   33   34   35
## 2288 1196  474  157   54
```

```
cumsum(rev(table(table(pixels))))/length(unique(pixels)) * 100
```

```
##          35          34          33          32          31          30
##   0.1697313   0.6632092   2.1530725   5.9123055  13.1038818  23.6460789
##          29          28          27          26          25          24
##  36.8316832  51.5354393  64.9819268  75.3512494  82.5051077  86.9558384
##          23          22          21          20          19          18
##  89.6746818  91.0231023  91.7460317  92.2929436  92.9687254  93.4873487
##          17          16          15          14          13          12
##  93.8645293  94.1285557  94.3674367  94.6094609  94.8892032  95.0495050
##          11          10           9           8           7           6
##  95.1909477  95.3041018  95.4266855  95.5869873  95.7064278  95.8321546
##           5           4           3           2           1
##  96.0238881  96.4890775  97.3471633  98.6735816 100.0000000
```

```
# Pixels with 20+ obs make up 92% of the pixels we obtain a
# vector of pixels and their type
# (shallow-reef/non-reef/variable depth reef)
pixels <- unique(pixels)
types <- scene.df$coralType[match(pixels, scene.df$pixel)]
```

As we can see if we define *enough data* for a pixel as it having at least 20 observations we will be able to fit models to 92% of the pixels. We will use this criterion and now apply the forward selection procedure. Ultimately, we are not interested in the model fits themselves but the variables selected for each pixel. This is because in section 3.3 we will re-fit the model to subsets (sub-periods) of the data for each pixel in order to determine if and when an intervention changed the pattern of surface reflectance (potential bleaching event).

```
# we initialize a list where we keep selected variables for
# each pixel
vars <- list()
# loop through vector of pixels
for (pix in pixels) {
    # keep track of progress
    count <- which(pix == pixels)
    if (count%%100 == 0)
        print(count/length(pixels) * 100)

    # obtain data for pixel 'pix'
    datPix <- scene.df[which(scene.df$pixel == pix), ]
    aux <- ""

    # if we have enough data we appy forward selection
    if (nrow(datPix) >= 20) {
        # plot(datPix$date, datPix$sr.B1)
        modMat <- cbind(datPix[, c("sr.B1", "date")], makeModelMat(datPix$day,
            datPix$hoursAbs, tidePeriods))
        fitNull <- lm("sr.B1~1", modMat)
        fit <- addN(fitNull, seasonalPreds, preds, N = 3)
        aux <- names(coef(fit))[2:length(coef(fit))]
    }

    # record selected variables
    vars[[length(vars) + 1]] <- aux

}
```

Lets take a look at the percentage of pixels which *selected* each predictor. We look at it by coral type.

```
# % that predictors were added for each type of coral
predPcts <- sapply(unique(types), function(typ) {
    indx <- which(types == typ)
    pixs <- pixels[indx]
    vs <- vars[indx]
    table(unlist(vs))/length(pixs) * 100
})
rownames(predPcts)[1] <- "insuf. data"
predPcts <- cbind(predPcts, mean = apply(predPcts, 1, mean))
predPcts <- predPcts[order(predPcts[, "mean"], decreasing = T),
    ]
pander(predPcts, caption = "% of pixels selecting each predictor")
```

Table 20: % of pixels selecting each predictor

| | non-reef | shallow_reef | variable_depth_reef | mean |
|---|---|---|---|---|
| **cosAnual** | 70.98 | 34.37 | 74.89 | 60.08 |
| **sinAnual** | 51.03 | 25.13 | 42.75 | 39.64 |
| **cosBianual** | 23.15 | 37.58 | 31.85 | 30.86 |
| **sin_tide_11.97** | 28.8 | 15.13 | 22.96 | 22.29 |
| **sinBianual** | 22.86 | 22.04 | 14.78 | 19.89 |
| **cos_tide_12.66** | 17.39 | 10.14 | 19.23 | 15.59 |
| **cos_tide_23.93** | 15.47 | 10.57 | 15.64 | 13.89 |
| **sin_tide_25.82** | 16.13 | 9.768 | 13.49 | 13.13 |
| **sin_tide_12.66** | 3.127 | 25.74 | 7.461 | 12.11 |
| **sin_tide_12.42** | 12.33 | 9.145 | 8.465 | 9.982 |
| **sin_tide_23.93** | 8.801 | 5.854 | 13.2 | 9.285 |
| **insuf. data** | 3.388 | 19.51 | 3.73 | 8.877 |
| **cos_tide_24.07** | 7.041 | 7.617 | 8.608 | 7.755 |
| **cos_tide_25.82** | 4.75 | 9.028 | 4.878 | 6.219 |
| **sin_tide_24.07** | 3.786 | 7.288 | 3.3 | 4.791 |
| **cos_tide_12.42** | 1.893 | 7.805 | 4.304 | 4.667 |
| **cos_tide_11.97** | 2.282 | 4.244 | 3.013 | 3.18 |

We now delete from relevant vectors and lists entries corresponding to pixels without sufficient data.

```r
# obtain indices of pixels with sufficient data
indx <- which(!sapply(vars, function(v) all(v == "")))
# delete entries of pixels with insufficient data
pixels <- pixels[indx]
types <- types[indx]
vars <- vars[indx]
names(vars) <- pixels
```

We have established a procedure for model fitting and selection for all pixels and have determined which predictors we wish to use in the fourier model 1 for each pixel. In the next section we will use this to establish the likliest date of a change in model coefficients indicating a possible intervention. If this occurs at similar dates for most pixels then we are dealing with an intervention affecting the entire 5.35km$^2$ area, possibly a bleaching event.

## 3.3 Seemingly Unrelated Regression

### 3.3.1 Theory

Model 1 can be rewritten in vector form in the following way:

$$y_i = \beta^T x_i + \epsilon_i \tag{3}$$

Where:

- $y$ is ultra-blue surface reflectance time series for a given pixel,
- $\beta = (\alpha_0 \alpha_1 \beta_1, ..., \alpha_8, \beta_8)^T$,
- $x := x(d, h_1, P_1, ..., h_8, P_8) = (\cos(\frac{2\pi d}{T})\sin(\frac{2\pi d}{T})\cos(\frac{2\pi d}{T})\sin(\frac{2\pi d}{T}))^T$,
- $\epsilon$ is the random error term which we assume satisfies usual assumptions of linear model:
    1. unbiased: zero mean,
    2. homogenous: contant variance,
    3. independent identically distributed, and
    4. normally distributed.
- $i$ indicates the observation number (associated to a date)

If we have $n$ observations we can write model 3 in matrix form:

$$Y = X\beta + \varepsilon \tag{4}$$

Where:

- $Y \in \mathbb{R}^{n\times 1}$,
- $\beta \in \mathbb{R}^{p\times 1}$,
- $X \in \mathbb{R}^{n\times p}$, and
- $\varepsilon \in \mathbb{R}^{n\times 1}$

If the assumptions for the errors hold and we can carry out a hypothersis test of the form:

$$\begin{aligned} H_0 &: C\beta = d \\ H_a &: C\beta \neq d \end{aligned} \tag{5}$$

where $C \in \mathbb{R}^{q\times p}$, $rank(C) = q$, by calculating the $F$ test statistic:

$$\begin{aligned} \hat{\beta} &= (X^TX)^{-1}X^TY \\ \hat{Y} &= X\hat{\beta} \\ r &= Y - \hat{Y} \\ \hat{\sigma}^2 &= \frac{(r^Tr)}{n-p} \\ F &= \frac{(C\hat{\beta}-d)^T(C(X^TX)^{-1}C^T)^{-1}(C\hat{\beta}-d)}{q\hat{\sigma}^2} \end{aligned} \tag{6}$$

which we know is distributed $\mathcal{F}_{q,n-p}$ if the null hypothesis is true. We reject the null hypothesis if the test statistic, at significance level $\alpha$, is larger than $P^{-1}_{\mathcal{F}_{q,n-p}}(1-\alpha)$ where $P_{\mathcal{F}_{q,n-p}}$ is the cumulative distribution function of an $\mathcal{F}$-distributed random variable with $q, n-p$ degrees of freedom.

The Seemingly Unrelated Regression (SUR) model uses a generalized linear hypothesis with respect to an *augmented* regression model to test the following hypothesis:

$$\begin{aligned} H_0 &: \beta_1 = \beta_2 \\ H_a &: \beta_1 \neq \beta_2 \end{aligned} \tag{7}$$

Where $\beta_1$ is the regression coefficient that applies to subset 1 of the datapoints and $\beta_2$ is the regression coefficient that applies to subset 2 of the data points.

To do this we carry out the following steps:

1. Choose two subsets, $\mathcal{S}_1$ and $\mathcal{S}_2$, of the data points which we wish to compare,
2. Augment model 3 to following model:

$$y_i = \beta_1^T w_i + \beta_2^T z_i + \epsilon_i$$

$$w_i = \begin{cases} x_i, i \in \mathcal{S}_1 \\ 0, i \in \mathcal{S}_2 \end{cases} \tag{8}$$

$$z_i = \begin{cases} 0, i \in \mathcal{S}_1 \\ x_i, i \in \mathcal{S}_2 \end{cases} \tag{9}$$

We can then express hypothesis 7 in the form of hypothesis 5 by carrying out following steps:

a. Use $d = 0$ and the following matrix $C \in \mathbb{R}^{p\times 2p}$:

$$C = \begin{pmatrix} 1 & 0 & \dots & 0 & -1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & -1 \end{pmatrix} \tag{10}$$

such that $C\beta = d$ corresponds to $\beta_1 = \beta_2$.

b. Form new matrix $X$:

$$X = \begin{pmatrix} X_1 & 0 \\ 0 & X_2 \end{pmatrix} \tag{11}$$

c. Calculate $F$ statistic and reject null hypothesis if $F > P^{-1}_{\mathcal{F}_{q,n-p}}(1-\alpha)$

### 3.3.2 Application to change point analysis

To apply the SUR model to change point analysis we simply define the two subsets of observations with respect to a *change* date: those that occurred before or on the date and those that occurred after. We now define a function to carry out the hypothesis test and try it out for one pixel and one change date.

```r
# obtain model-matrix and response variable for ALL pixels
modMat <- cbind(scene.df[, c("sr.B1", "date", "pixel")], makeModelMat(scene.df$day,
    scene.df$hoursAbs, tidePeriods))

# we define a function that performs SUR hypothesis test.
# function takes pixel and change date so as to define two
# subsets of informations and model matrix and the predictors
# (vars) which were selected and will be used to fit the
# model of each pixel.
hypPixDate <- function(pix, chng.date, modMat, vars) {
    # obtain the model matrix and response variable for the
    # chosen pixel
    modMatPix <- modMat[which(modMat$pixel == pix), ]
    # define the model matrix for both subsets of observations
    X1 <- as.matrix(modMatPix[which(modMatPix$date <= chng.date),
        vars[[as.character(pix)]]])
    X2 <- as.matrix(modMatPix[which(modMatPix$date > chng.date),
        vars[[as.character(pix)]]])

    # intitialize pvalue
    pval <- NA
    # we only carry out hypothesis test if there are at least 11
    # observations in both subsets
    if (nrow(X1) > 10 & nrow(X2) > 10) {
        # we construct augmened model-matrix
        X <- rbind(cbind(X1, matrix(0, nrow(X1), ncol(X2))),
            cbind(matrix(0, nrow(X2), ncol(X1)), X2))
        # get response variable
        Y <- modMatPix[, "sr.B1"]
        # total number of observations
        n <- nrow(X)
        # total number of predcitors
        p <- ncol(X1)

        # calcualte OLS estimator for beta
        beta_h <- solve(t(X) %*% X) %*% t(X) %*% Y
        # calculate OLS estimator for response variable
        Y_h <- X %*% beta_h
        # calculate residuals
        res <- Y - Y_h
        #
        sigma_h <- sqrt(sum(res^2)/(n - 2 * p))
        # Construct C matrix which encodes our null hypothesis H0:
        # beta_0_1 = beta_0_2,....,beta_p_1=beta_p_2
        C <- t(rbind(diag(1, p), diag(-1, p)))
        # Calculate F statistic
        Fstat <- as.numeric((t(C %*% beta_h) %*% solve(C %*%
            solve(t(X) %*% X) %*% t(C)) %*% (C %*% beta_h))/(p *
            sigma_h^2))
        # Calculate p-value with respect F statistic and degrees of
        # freedom
        (pval <- 1 - pf(Fstat, df1 = p, df2 = n - 2 * p))

    }
    return(pval)
}

# Set a change date
chng.date <- as.Date("2016-03-01")

# calculate the p-value for the no-change at '2016-03-01'
# null hypothesis for pixel 51
indx.pix <- 51
pix <- pixels[indx.pix]
hypPixDate(pix, chng.date, modMat, vars)
```

```
## [1] 0.5797709
```

In this case we would not reject null hypothesis at an $\alpha$ level of 10%. In the next section we apply hypothesis to all pixels and all elligible change dates. We want to see if the reflectance pattern of most pixels changed at a particular date providing evidence of a scene-wide intervention such as a bleaching event.

### 3.3.3 Results

We apply SUR hypothesis test to all pixels and with all *elligible* change dates. We first define elligible change dates: observation dates such that there are at least nine observation dates prior to that date and 9 observations posterior to that date. We do this in order to allow enough observations in both subsets $\mathcal{S}_1$ and $\mathcal{S}_2$.

```
# minimum number of observation dates prior to and including
# a change date, also minimum number of observation dates
# posterior to and including a change date.
minDts <- 10
# we define the elligible change dates
leftLimit <- sort(unique(scene.df$date))
leftLimit <- leftLimit[minDts:(length(leftLimit) - minDts)]
chng.dates <- leftLimit
length(chng.dates)
```

```
## [1] 27
```

We now calculate the p-values for the hypotheses of all pixels with all elligible change dates used to define both subsets.

```
pm <- proc.time()
pvals <- sapply(chng.dates, function(dt) sapply(pixels, function(px) {
    if (px == pixels[1])
        print(dt)
    hypPixDate(px, dt, modMat, vars)
}))
proc.time() - pm
```

```
## [1] 29363    27
```

We see that we have one p-value for every combination of pixel and elligible change date. In the next section we'll analyze the scene-wide p-value statistics by date in order to detect any dates at which the whole scene was affected.

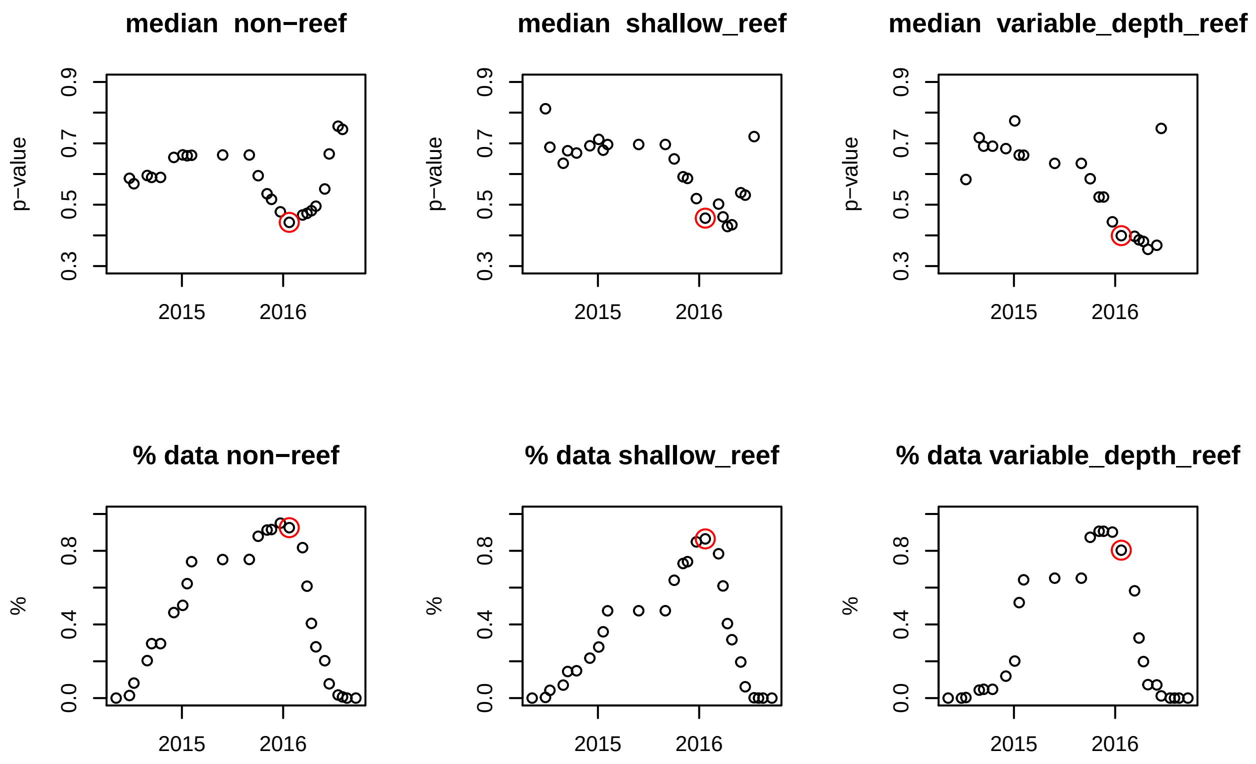
#### 3.3.3.1 Plausible bleaching dates

We'll plot the median scene-wide p-value by date and also the % of pixels for which there was sufficient data to calculate their p-value. As before we'll do this for the different coral-reef groups: shallow-reef, variable depth-reef and non-reef.

```
# We'll mark the observations corresponding to following date
# on graph as reference
indx.dt <- 17
chng.dates[indx.dt]
```

```
## [1] "2016-01-23"
```

```
par(mfcol = c(2, 3))
dumy <- lapply(unique(types), function(typ) {
    indx.ty <- which(types == typ)

    # median p-value per date
    plot(chng.dates, apply(pvals[indx.ty, ], 2, median, na.rm = T),
        main = paste("median ", typ), ylab = "p-value", xlab = "",
        ylim = c(0.3, 0.9))
    lines(chng.dates[indx.dt], apply(pvals[indx.ty, ], 2, median,
        na.rm = T)[indx.dt], type = "p", col = "red", cex = 2)
    # of pixels for which there is enough data for hypothesis
    # test at each date
    plot(chng.dates, apply(pvals[indx.ty, ], 2, function(p) sum(!is.na(p))/length(p)),
        main = paste("% data", typ), ylab = "%", xlab = "", ylim = c(0,
            1))
    lines(chng.dates[indx.dt], apply(pvals[indx.ty, ], 2, function(p) sum(!is.na(p))/length(p))[indx.dt
        type = "p", col = "red", cex = 2)
})
```



We see that towards the beginning of 2016 the median p-value significantly drops perhaps indicating that something affected the reflectance pattern of most pixels in the scene. This could possibly be due to a bleaching event. In the next section we'll use the p-values for the 2016-03-01 observation date to plot a map of p-values. If we interpret low p-values as potential bleaching sites we arrive at our first approximation of a bleaching map for our study area.

#### 3.3.3.2 Map of bleaching

We want to use the pixelwise p-values for 2016-03-01 change hypothesis as a proxy for potential bleaching sites. However, right now our p-values are stored in a matrix where each row corresponds to a pixel and each column to a change date. Additionally recall that we don't have values for all pixels as around 8% did not have enough data for us to fit a regression model. We first create a p-value raster stack from our p-value matrix where each layer of the stack will correspond to a change date.

```
# transfrom pvals into a data.frame so that we can add a
# pixel id variable
pvals2 <- as.data.frame(pvals)
colnames(pvals2) <- chng.dates
pvals2$pixel <- pixels
pvals2$type <- types
# create a data.frame with the study area raster grid
# coordinates this is where we'll add the p-values since it
# contains all pixels wether they have p-value info or not
coral.Info.df <- as.data.frame(coordinates(coral.r))
# obtain pixel id variable associated to each raster cell
coral.Info.df$pixel <- cellFromXY(coral.r, coral.Info.df)
# identify those pixels that don't have at least 20 data
# points
coral.Info.df$info <- 0
```

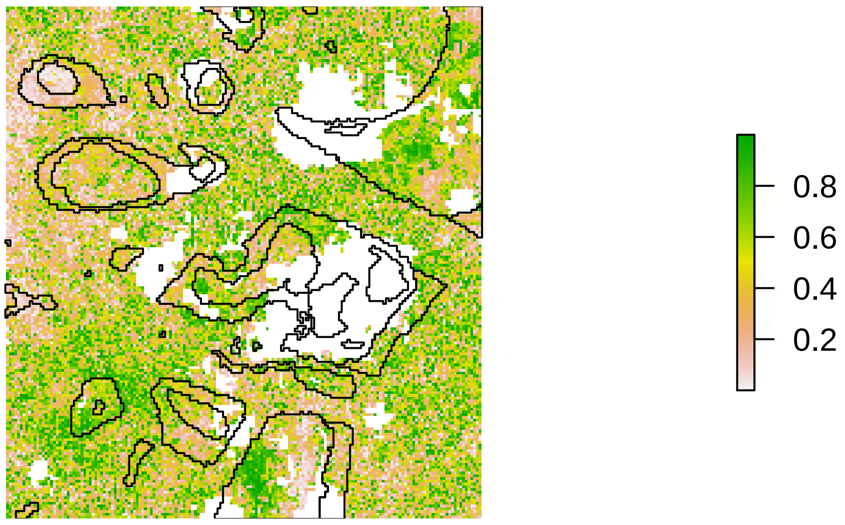
```r
coral.Info.df$info[which(coral.Info.df$pixel %in% pixels)] <- 1
# merge p-values with pixel database using pixel id to match
coral.Info.df <- merge(coral.Info.df, pvals2, by = "pixel", all.x = T)
# transform pixel database into a raster stack
coral.Info.r <- rasterFromXYZ(coral.Info.df[, -1], crs = CRS(projection(coral.r)))
```

```
## Warning in matrix(as.numeric(xyz), ncol = ncol(xyz), nrow = nrow(xyz)): NAs
## introduced by coercion
```

```r
# create a potential bleaching map with coral type layer,
# available info layer and our chosen 'likely' change date.
bleach.r <- stack(coral.r[["coralType"]], coral.Info.r[[c("info",
    "X2016.01.23")]])
```

We are now ready to plot our potential bleaching raster.

```r
plot(bleach.r[["X2016.01.23"]], axes = F, box = F)
plot(maldives.shp, add = T)
```



As we can see our map has some holes due to pixels with insufficient data: either they didn't have at least 20 observations or they didn't have 9 observations before 2016-01-23 and after. Additionally, although we can clearly appreciate a spatial pattern there is some noise to it since the color doesn't change smoothly from pixel to pixel. In the next section we will fill the holes in the map using an interpolation technique, and we will smooth the map using convolutions in order to produce a more easily interpretable map of potential bleaching regions.

## 4 Post-processing

### 4.1 Interpolation

This subsection (subsection 4.1) is largely based on section 8.4 from Bivand, Pebesma, and Gómez-Rubio (2013). In this section we fill the holes in the map by interpolating in such a way that is consistent with the pattern observable in the non-missing pixels.

### 4.1.1 Model

We assume the p-values in the 5.35km$^2$ area can be modeled according to:

$$P(s) = m + e(s) \tag{12}$$

where

- $e(s) \sim N(0, \sigma^2)$

P-values are in $(0, 1)$ interval so modeling them with normal variables is not recommended, however for simplicity we won't perform transformations on the p-values before interpolation. In standard statistical problems, correlation can be estimated from a scatterplot, when several data pairs $x, y$ are available. The spatial correlation between two observations of variable $p(s)$ at locations $s_1$ and $s_2$ cannot be estimated, as only a single pair is available. To estimate spatial correlation from observational data, we therefore need to make stationarity assumptions before we can make any progress. One commonly used form of stationarity is intrinsic stationarity, which assumes that the spatial correlation between random variables $P(s_1)$ and $P(s_2)$ does not depend on locations $s_1$ and $s_2$, but only on separation vector $h = s_1 - s_2$. Under this assumption we only need to estimate the parameter $\sigma^2$ and the correlation function $\rho(h) := Corr(P(s), P(s+h))$ to fully specify the joint distribution of $P(s)$ variables. We can then form multiple pairs $p(s_i), p(s_j)$ that have (nearly) identical separation vectors $h = s_i???s_j$ and estimate correlation from them. If we further assume isotropy, which is direction independence of correlation, we can replace the vector $h$ with its length, $l = ||h||$. Lets get an idea of the behavior of the correlation function $\rho(l)$ by plotting scatterplots by pairing observations according to the distance between them.

```
# we first form a SpatialPointsDataFrame with the p-values
# for potential bleaching date 2016-01-23 and their
# coordinate. We will need this to plot correlations and
# later to fit interpolation model

model.df <- coral.Info.df[, c("pixel", "x", "y", "type", "2016-01-23")]
model.df <- na.omit(model.df)
colnames(model.df) <- c("pixel", "x", "y", "type", "pval")
# lets see if there is any noticable difference in p-values
# by coral-reef type that may justify replacing the model
# P(s) = m + e(s) which has a general mean plus a noise term
# with P(s) = m + t(s) + e(s) where t(s) is a term that
# depends on the coral-reef type at location s
aggregate(model.df$pval, by = list(model.df$type), FUN = mean)
```

```
##               Group.1         x
## 1            non-reef 0.4612899
## 2         shallow_reef 0.4698005
## 3 variable_depth_reef 0.4427532
```

```
# there doesn't seem to be much difference so we stick with
# P(s) = m + e(s) model

# transform into a spatial points data frame so we can plot
# spatial correlations with hscat and estimate sample
# variogram with variogram
coordinates(model.df) <- c("x", "y")
projection(model.df) <- projection(bleach.r)
```

Before plotting correlations we need to establish distance intervals with respect to which we will form data pairs. We first get an idea of the distance distribution of the data pairs. There are

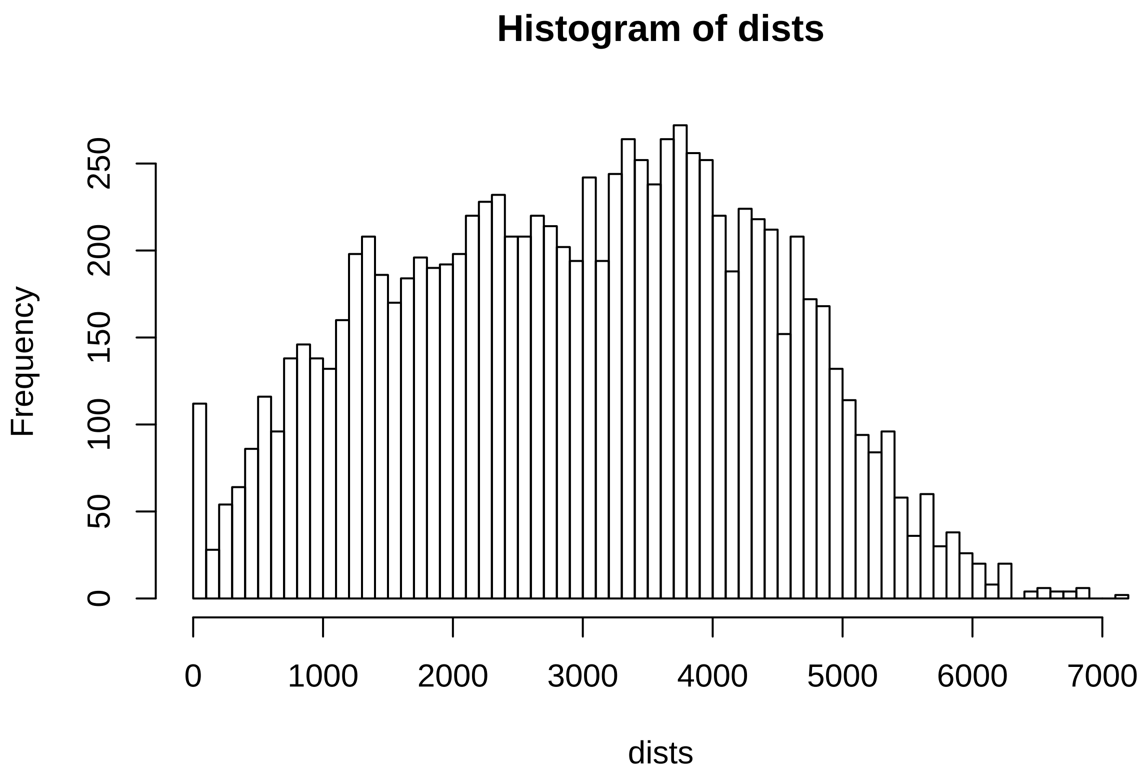
$$\binom{26,678}{2} = 355,871,181 \tag{13}$$

data pairs so we will take a random sample of 100 points so that we only have to calculate distances for 4950 data pairs.

```
dists <- as.numeric(gDistance(model.df[sample(1:nrow(model.df),100),], byid=T))
summary(dists)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0    1847    3032    2967    4049    7106
```

```
hist(dists, 100)
```



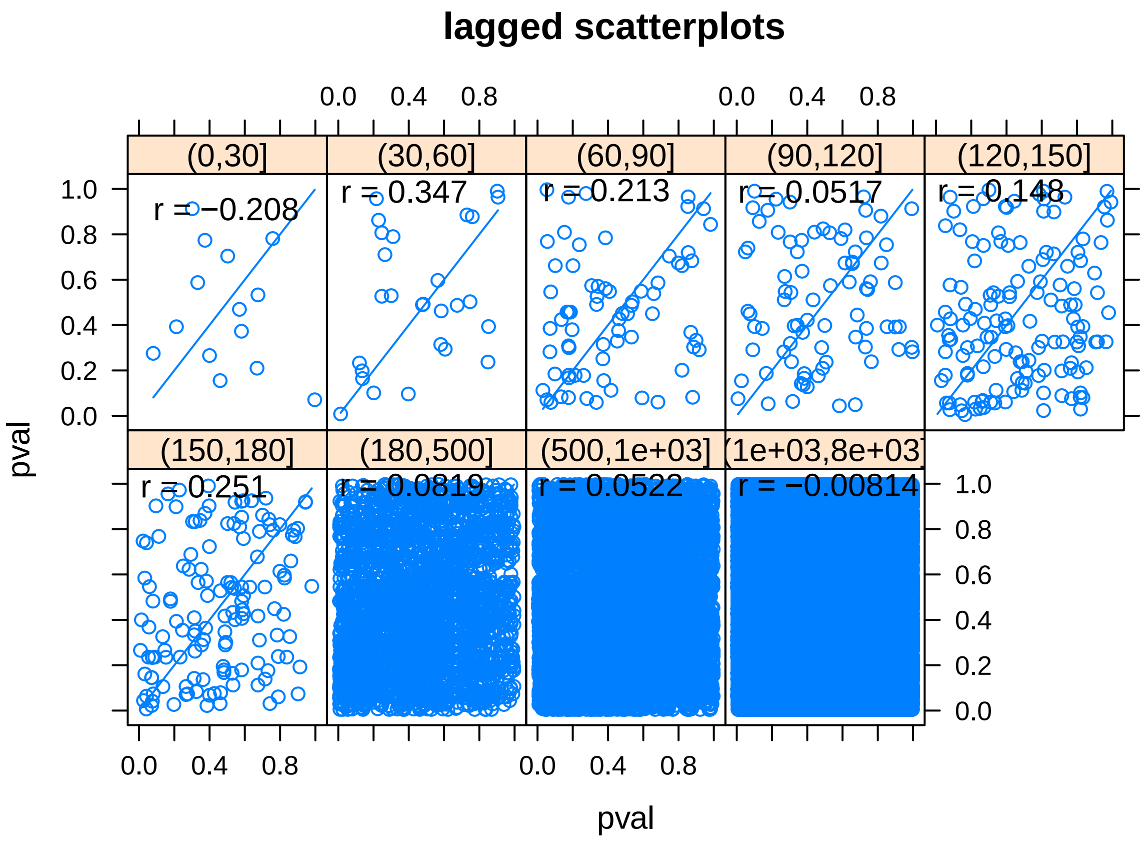
We'll set the distance intervals up to a maximum distance of 8000 meters and focus on smaller distances. Since the resolution of the raster grid is 30 metres the first distance intervals will be multiples of 30 meters.

```
# set distance intervals
brks <- c(0, 30, 60, 90, 120, 150, 180, 500, 1000, 8000)
# we do not produce scatterplots for the all data pairs as we
# have seen there are over 350 million of them. We sample 500
# data points which yield over 100k data pirs
N <- 500
choose(N, 2)  #number of data pairs with sample size
```

```
## [1] 124750
```

```
set.seed(4)
smpl <- sample(1:nrow(model.df), N)
```

```
# we load gstat library for plotting scatterplots for each
# distance interval, for calculating sample variogram,
# fitting parametric variogram and interpolation model in
# general
library(gstat)
hscat(pval ~ 1, model.df[smpl, ], breaks = brks)
```



In general, correlation between points 180 metres apart or less is high and for those further apart much less, although the pattern is somewhat noisy or inconsistent. In the next section we will get a better idea of the correlation pattern with respect to the distance between points by estimating the *variogram* function.

#### 4.1.1.1 Variogram

In geostatistics the spatial correlation is modelled by the variogram instead of a correlogram or covariogram, largely for historical reasons. Under intrinsic stionarity the the variogram is defined as:

$$\gamma(h) = \frac{1}{2}\mathbb{E}[(P(s) ??? P(s+h)]^2 \tag{14}$$

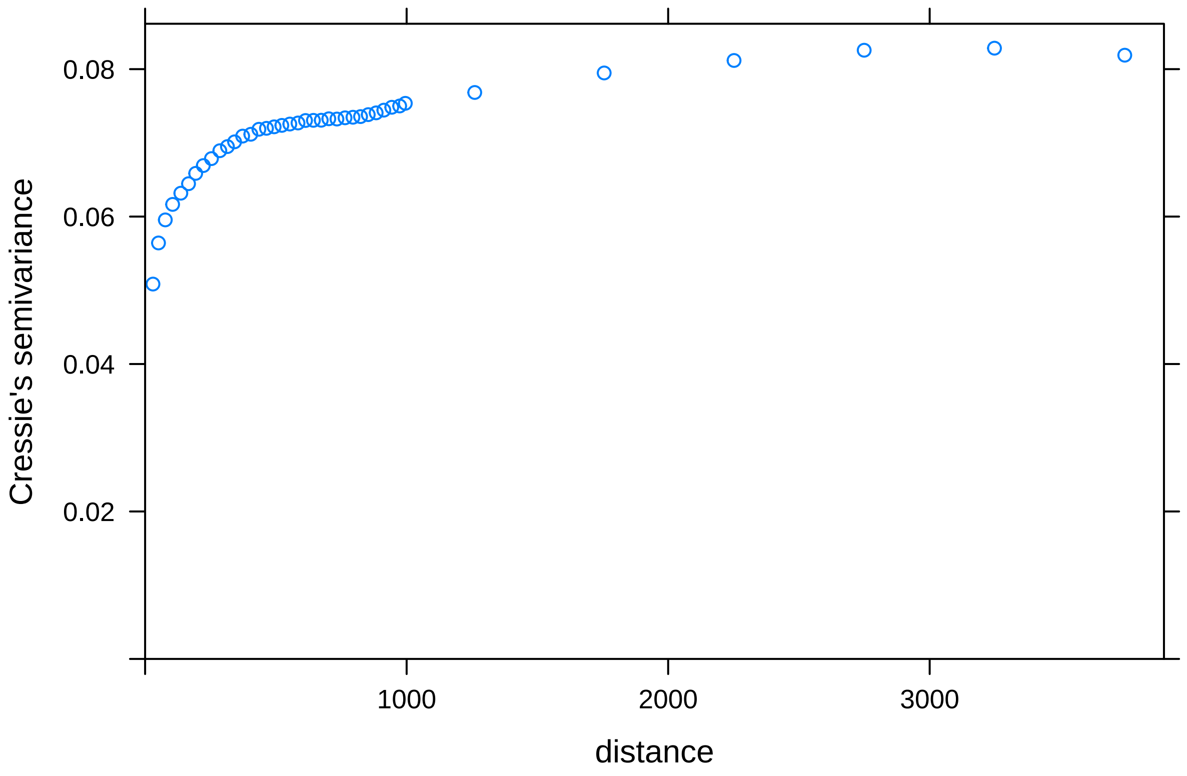
$

Here, the word variogram will be used synonymously with semivariogram. The variogram plots semivariance as a function of distance. In contrast to the correlation function $\rho(h)$ the smaller $\gamma(h)$ is the larger the correlation so that $\gamma(h)$ is usually an increasing function of $l = ||h||$: normally the further two points $s_1$ and $s_2$ are in space the weaker is the correlation of the variables $p(s_1)$ and $p(s_2)$. Under the intrinsic stationarity and isotropy assumption, the variogram can be estimated from $N_h$ sample data pairs $p(s_i), p(s_i + h)$ for a number of distances (or distance intervals) $\tilde{h}_j$ by:

$$\hat{\gamma}(\tilde{h}_j) = \frac{1}{2N_h} \sum_{i=1}^{N_h} (p(s_i) - p(s_i + h))^2 \forall h \in \tilde{h}_j \tag{15}$$

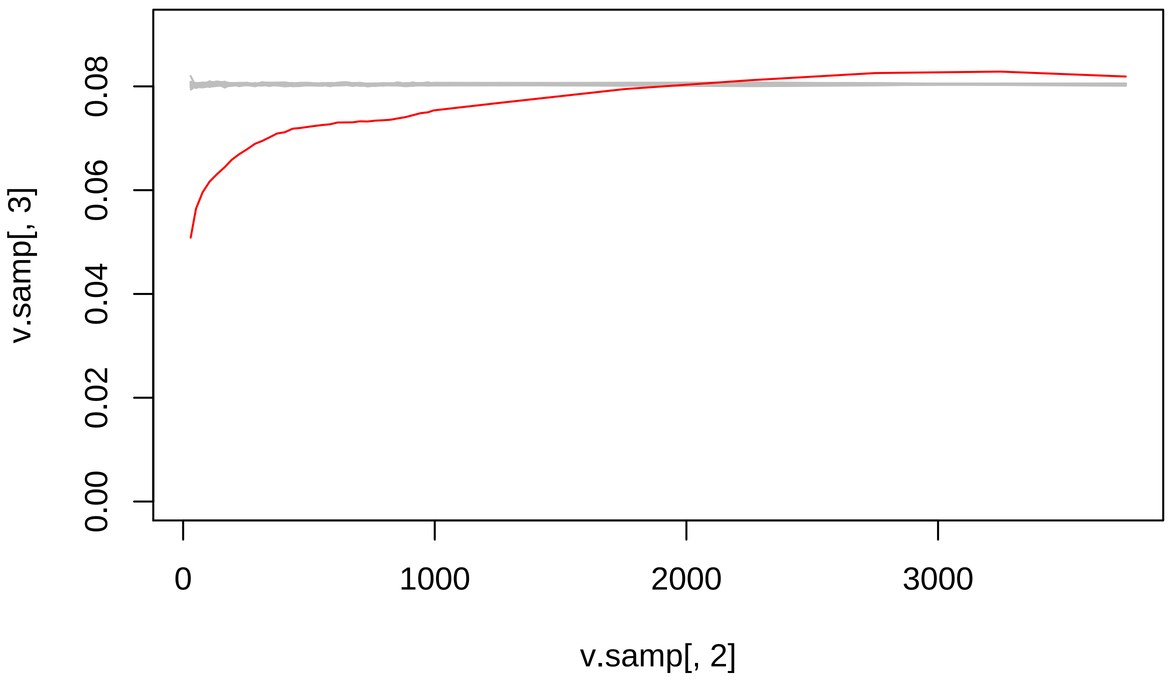
and this estimate is called the sample variogram. We calculate and plot the sample the sample variogram:

```r
brks <- c(seq(0, 990, by = 30), seq(1000, 4000, 500))
v.samp <- variogram(pval ~ 1, model.df, cloud = FALSE, cressie = TRUE,
    boundaries = brks)
plot(v.samp)
```



Although the spatial correlation pattern seems plausible -as the distance between points increases the sample variogram function increases indicating a decrease in correlation- we make sure that it isn't a simply a random pattern by calculating and plotting twenty sample variograms for the same set of p-values but where the associated coordinates have been re-asigned randomly so that any spatial correlation is lost.

```r
# dummy plot to obtain plot window with appropriate p-value
# limits (0,1)
plot(v.samp[, 2], v.samp[, 3], ylim = c(0, 1.1 * max(v.samp[,
    3])), col = "white")
aux.df <- model.df
for (i in 1:20) {
    # re arrange pvalues assigning them to random pixels
    aux.df$pval <- model.df$pval[sample(nrow(model.df))]
    # calculate sample variogram of non spatially correlated
    # p-values
    v.aux <- variogram(pval ~ 1, aux.df, cloud = FALSE, cressie = TRUE,
        boundaries = brks)
    # plot non spatial correlation sample variogram
    lines(v.aux[, 2], v.aux[, 3], col = "grey", type = "l")
}
# plot true variogram at the end
lines(v.samp[, 2], v.samp[, 3], col = "red")
```



We can see that the spatial correlation is not a product of a random p-value observations. We will now fit a parametric variogram function to our sample variogram to guard against overfitting to data and in order to have a simple model (specified by $m$, $\sigma^2$ and $\gamma(l)$) that is easier to use for interpolation. Additionally, to ensure that predictions are associated with non-negative prediction variances, the matrix with semivariance values between all observation points and any possible prediction point needs to be non-negative definite. For this, simply plugging in sample variogram values from the sample variogram is not sufficient, but plugging in from a parametric variogram model is.

Most parametric variograms have a nugget, partial sill and range parameter while some have additional parameters specifying the shape of the variogram function $\gamma(h)$. The following figure shows a variogram with its various components.

We will now fit a parametric diagram by choosing from the family of variograms that most closely matches our sample variogram, estimating the initial values for the partial sill, range and nugget from our plot of the sample variogram and then updating this estimation using the function `fit.variogram` from the `gstat` library. This function uses *damped least squares* to estimate the parameters.

```
# visualize available parametric variogram models
show.vgms()
```

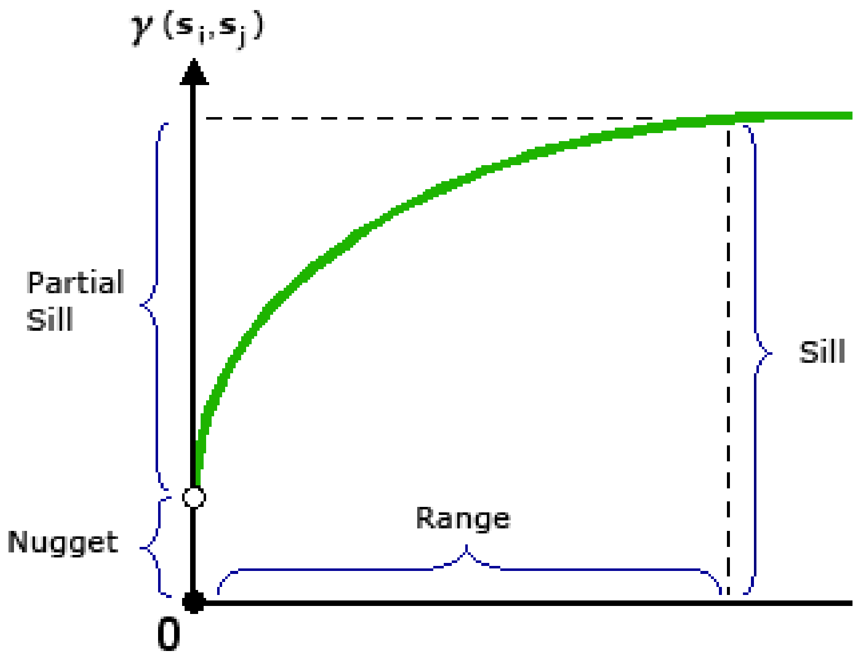
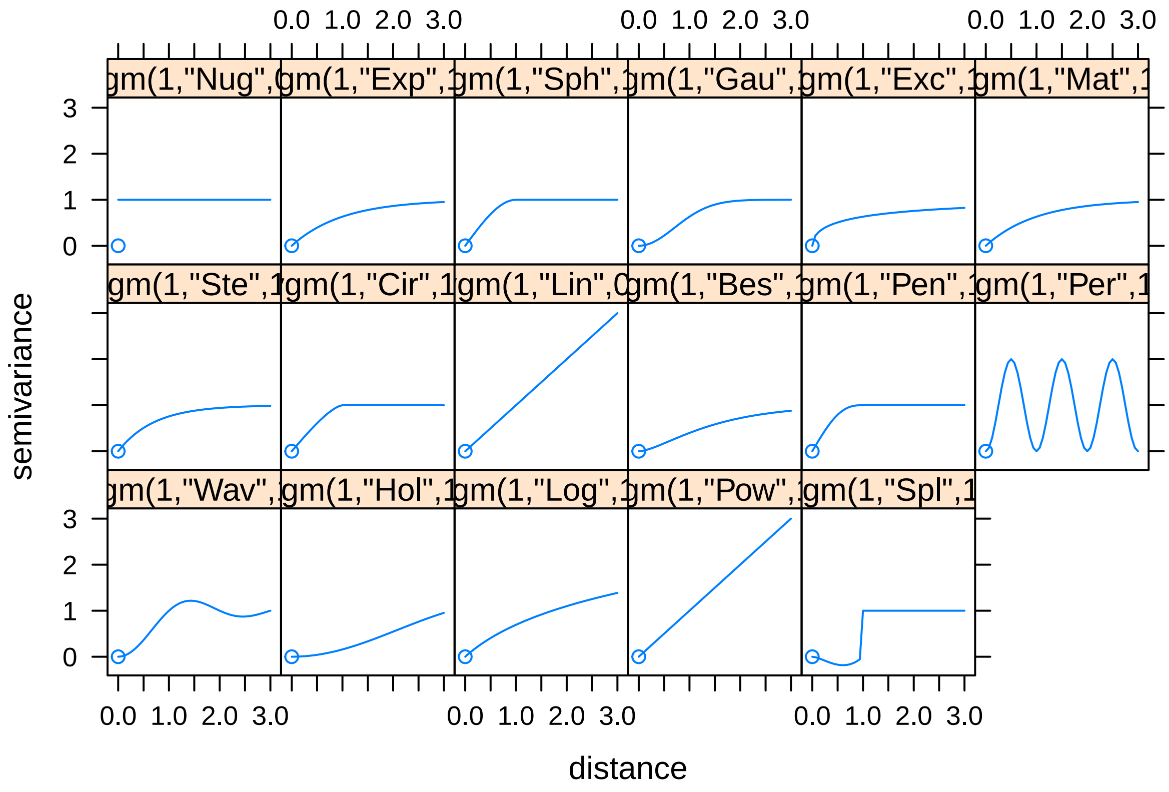


Figure 23: Variogram parameters

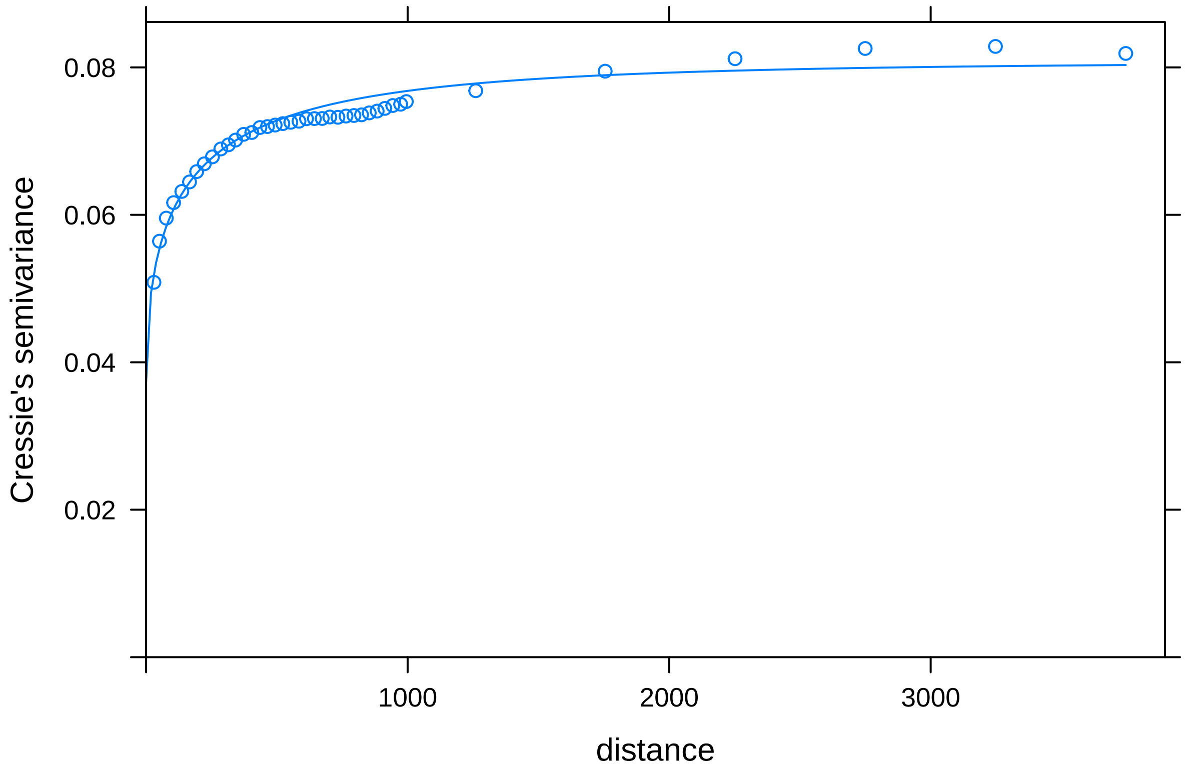


We judge the *exponential class* (`Exc`) to most closely resemble our sample variogram.

```
# we initialize parameters using sample variogram plot
v.mod <- vgm(psill = 0.03, "Exc", range = 200, nugget = 0.05)
# run parameter estimation
v.fit <- fit.variogram(v.samp, v.mod, fit.kappa = TRUE)
```

We now plot the sample variogram and our fitted parametric variogram.

```
plot(v.samp, v.fit)
```



The fit seems to be reasonably close. Normally we would try two or three parametric families of variograms and then select using cross-validation. For simplicity we stick with the exponential class of variograms. In the next section we will use our fitted variogram model to interpolate the missing p-values in our raster grid.

#### 4.1.1.2 Prediction

We use our parametric variogram model to interpolate, or *krige*, the missing values in our p-value raster. Kriging consists of estimating a missing value as a weighted average of observed values. The weight of an observationa distance of $l$ away from the location of the missing value is a function of $\gamma(l)$. Kriging gives the best linear unbiased prediction of the missing value. We first obtain a dataframe with the coordinates of the missing values and transform it into a SpatialPointsDataFrame. The `krige` function from the `gstat` package takes as input the locations at which to interpolate in SpatialPointsDataFrame format.

```
# Obtain coordinates of missing values
fullModel.df <- coral.Info.df[, c("pixel", "x", "y", "type",
    "2016-01-23")]
colnames(fullModel.df) <- c("pixel", "x", "y", "type", "pval")
fullModelNA.df <- fullModel.df[which(is.na(fullModel.df$pval)),
    ]
# transform dataframe into a SpatialPointsDataFrame
```

```
coordinates(fullModelNA.df) <- c("x", "y")
projection(fullModelNA.df) <- projection(bleach.r)
```

We now perform interpolation or *kriging* with our parametric variogram `v.fit` at the location of the missing values. This is quite computationally intensive!

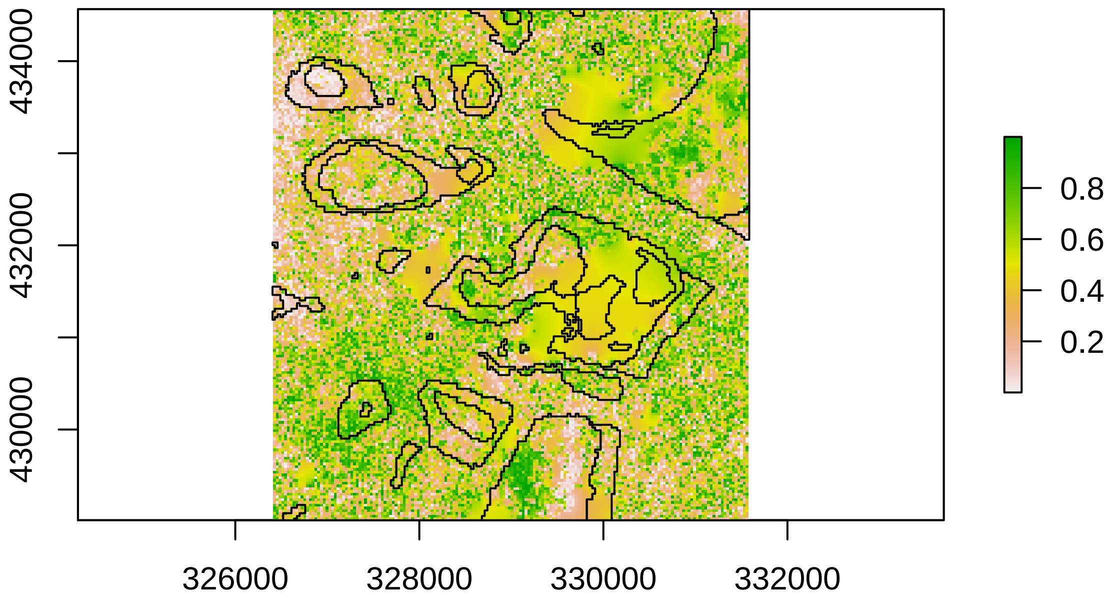
```
pm <- proc.time()
krig <- krige(pval ~ 1, locations = model.df, newdata = fullModelNA.df,
    model = v.fit)
proc.time() - pm
# 4.06 HRS
```

We now fill in the missing raster grid values.

```
fullModel.df$pval[which(is.na(fullModel.df$pval))] <- as.data.frame(krig)[,
    3]
bleachFull.r <- rasterFromXYZ(fullModel.df[, c("x", "y", "pval")],
    crs = CRS(projection(coral.r)))
```

Lets plot our complete potential bleaching raster.

```
plot(bleachFull.r)
plot(maldives.shp, add = T)
```



Interpolated regions are quite smooth since the predicted value is an expectation and lacks any noise. However, the rest of the map is still quite noisy. We could use our variogram model to smooth the map by interpolating at non-missing locations but, as we have seen, kriging is quite computationally expensive. In the next section we'll use convolutions to smooth our map.

## 4.2 Smoothing

We will use 2d-convolutions which correspond to a 2d-moving average. The `imager` package has a `convolve` functon for efficient convolution of images. We will use two approaches to smoothing. In the first we simply spatially smooth the p-value raster directly. In the second we'll first threshold the p-values to obtain a 0/1 raster of predicted bleached locations and then use a majority neighborhood rule to decide if a given pixel is bleached or not.

We first write a conversion function to convert cimg images back to raster format for plotting since the raster plotting functions are better for images with one band.

```
# conversion of rasters to cimg is quite straight forward
bleachFull.cimg <- as.cimg(as.matrix(bleachFull.r))
dim(bleachFull.cimg)
```

```
## [1] 185 172   1   1
```

```
# conversion of cimg to rasters is a little trickier. We
# write a function that takes in cimg and a template
# containing the relevant raster grid and returns the cimg
# image in raster format
cimgToRaster <- function(cimg, tmpl.r) {
    # convert cimg to data frame format
    df <- as.data.frame(cimg)[, c(2, 1, 3)]
    colnames(df) <- c("x", "y", "pval")
    # the y coordinate counts from the top down for cimgs and
    # from bottom up for rasters so we reverse
    df$y <- rev(df$y)
    # obtain coordinates of raster to use the min x and y
    # coordinates and resolution of grid to shift and scale
    # coordinates appropriately
    crds <- as.data.frame(coordinates(tmpl.r))
    df$x <- (df$x - 1) * res(tmpl.r)[1] + min(crds$x)
    df$y <- (df$y - 1) * res(tmpl.r)[1] + min(crds$y)
    # convert to raster format once coordinates are fixed
    rast <- rasterFromXYZ(df, crs = CRS(projection(tmpl.r)))
    return(rast)
}
# we test our conversion function making sure we get back
# original raster
bleachFull.r2 <- cimgToRaster(bleachFull.cimg, bleachFull.r)
all(values(bleachFull.r) == values(bleachFull.r2))
```

```
## [1] TRUE
```

```
all(coordinates(bleachFull.r) == coordinates(bleachFull.r2))
```

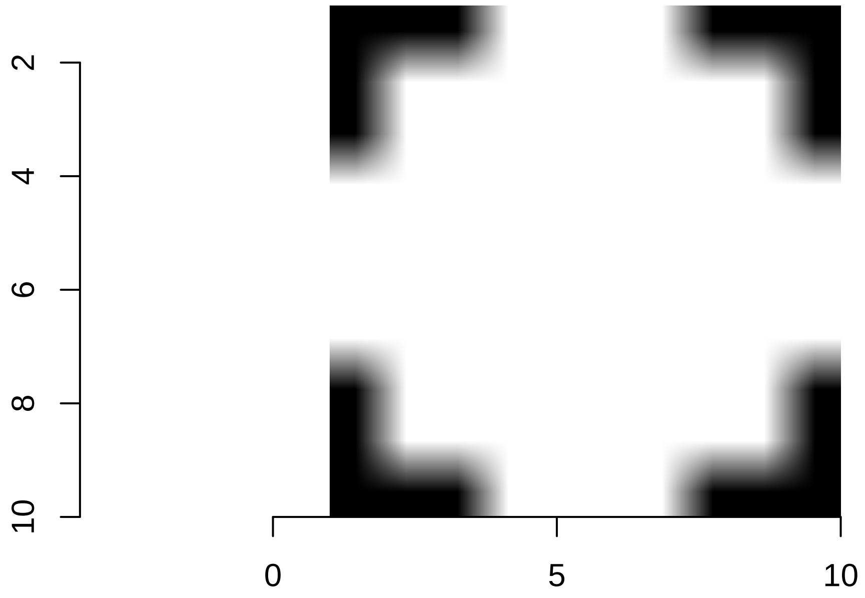
```
## [1] TRUE
```

We now define a *filter* or *window* with a circular shape with a 5 pixel radius. The filter is an actual cimg image of dimension 10 by 10 where 80 pixels make up the circle and 20 the difference between the 10 by 10 square and the circle. The pixel value is 1 if its in the circle and 0 other wise. We want to obtain a moving average so we divide the pixel value by the sum of all pixel values in the window.

```
filter <- px.circle(5, 10, 10)
sum(filter)
```

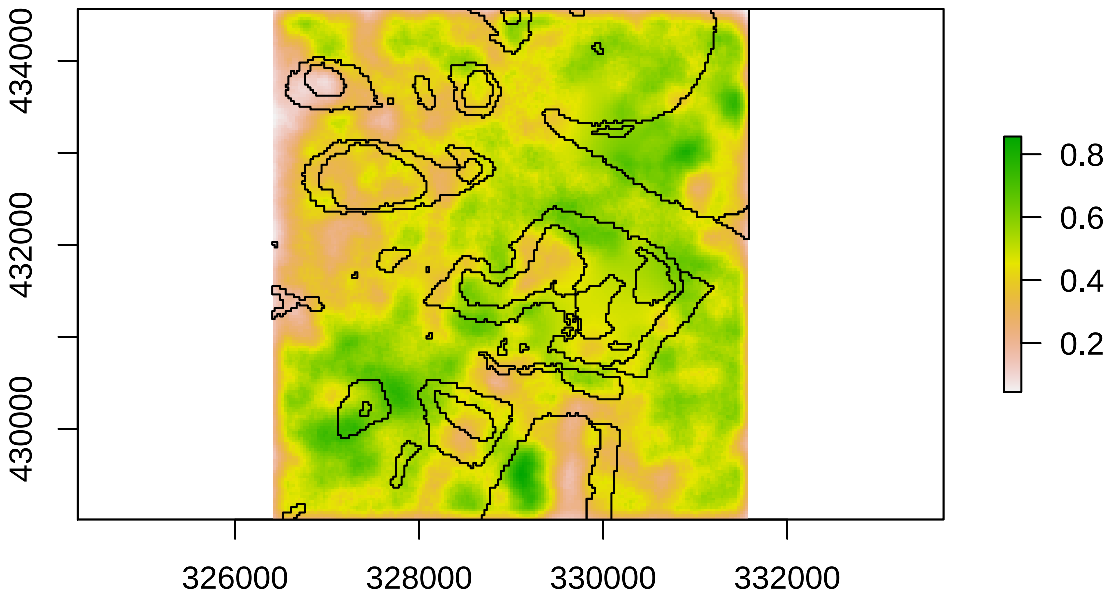
```
## [1] 80
```

```
filter <- filter/sum(filter)
plot(filter)
```



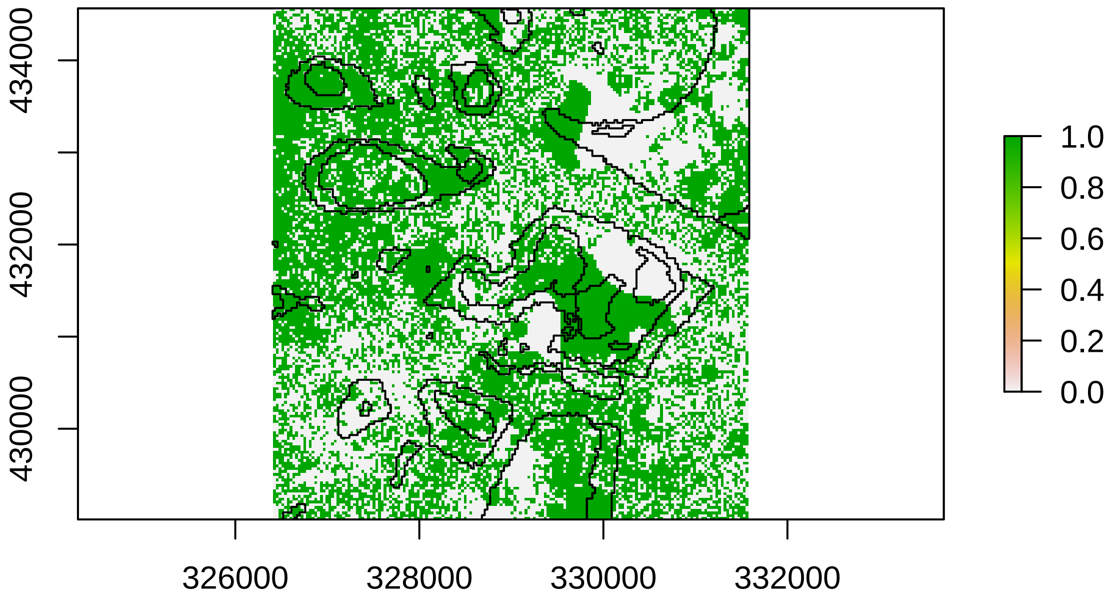
We now obtain a smoothed potential bleaching map by convolving original map with our filter. This is our first approach.

```
bleachFullCnv.cimg <- convolve(bleachFull.cimg, filter)
bleachFullSmooth.r <- cimgToRaster(bleachFullCnv.cimg, bleachFull.r)
plot(bleachFullSmooth.r)
plot(maldives.shp, add = T)
```



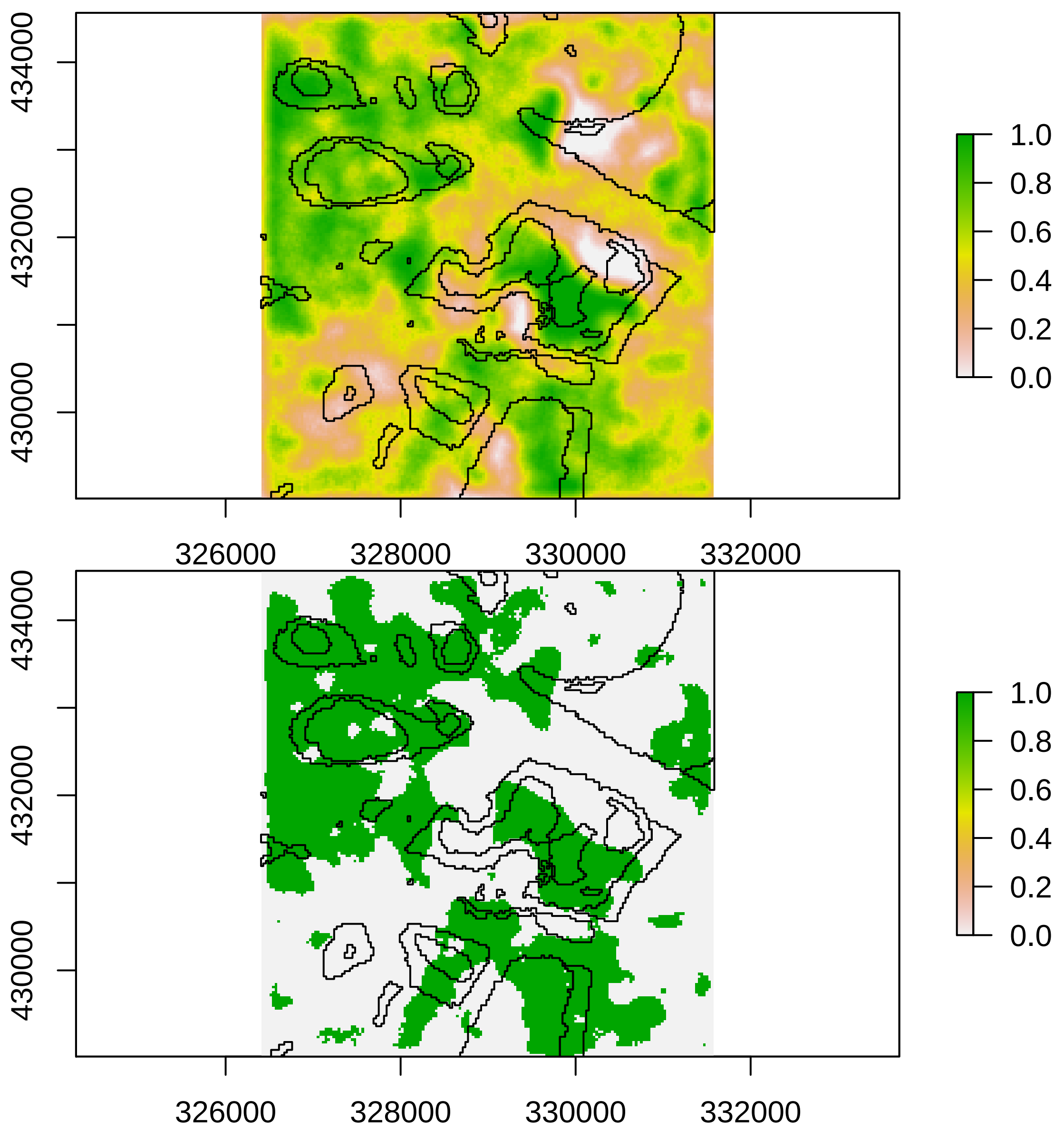
We now use the second approach. We want to threshold p-values to obtain a map of predicted potential bleaching sites. A natural threshold would be 0.05 or 0.1 since these are values at which p-values are normally thresholded. The problem is that since the hypothesis test have a spatial dependence structure if we threshold them individually at a certain level $\alpha$ the actual joint significance is likely to be different. There exist methods to correct for the spatial dependence structure in order to threshold p-values and control the significance for example in Leek and Storey (2008). For simplicity we will simply use the thresholding level $\alpha$ as a parameter of the methodology and arbitrarily set it at 0.5.

```r
# Set thresholding level
thrs.bleach <- 0.5
# Produce a raster of predicted potential bleaching sites
bleachThrs.r <- bleachFull.r < thrs.bleach
plot(bleachThrs.r)
plot(maldives.shp, add = T)
```



We now want to implement a majority rule whereby if 60% or more of the neighborhood of a pixel is bleached we consider the pixel to be bleached. We can implement this by convolving our 0/1 potential bleaching site indicator raster and then thresholding it at 0.6.

```r
bleachThrs.cimg <- as.cimg(as.matrix(bleachThrs.r))
bleachThrsCnv.cimg <- convolve(bleachThrs.cimg, filter)
bleachThrsCnv.r <- cimgToRaster(bleachThrsCnv.cimg, bleachFull.r)
plot(bleachThrsCnv.r)
plot(maldives.shp, add = T)
thrs.neighbor <- 0.6
bleachMajority.r <- bleachThrsCnv.r > thrs.neighbor
plot(bleachMajority.r)
plot(maldives.shp, add = T)
```



We obtain our final potential bleaching site map.

## 5 Extensions

The purpose of this practical is to explore the use of landsat imagery data in a landcover classification application. In order to keep the analysis relatively simple many of the model or parameter choices were not justified sufficiently and model assumptions were not rigorously checked. Additionally, we did not have *ground* data available for maldives bleaching so the validity of the methodology in identifying bleaching sites has not been properly tested.

Possible improvements to the methodology in order to validate its efficacy in identifying bleaching sites include:

- **Corroborate validty**. Ways of doing this include:
    a. Obtain ground bleaching datg and compare results.

b. Perform study in other areas with bleaching events and see if behavior of median p-values is as in this study.

c. Use methodologies with other input information such as those based on ocean surface temperature estimation and compare results.

d. Analyse the fitted fourier model parameter values especially $\alpha_0$ which corresponds to the general reflectance level. Did $\alpha_0$ increase (bleached coral theoretically not only changes but increases surface reflectance) after the bleaching event in all or most pixels? Can we perform the hypothesis test:

$$\begin{aligned} H_0 &: \alpha_{01} < \alpha_{02} \\ H_a &: \alpha_{01} \leq \alpha_{02} \end{aligned} \tag{16}$$

- **Justify model/parameter choices**

  1. **Fourier model**. Justify more thoroughly choice of ultra-blue. Explore use of other bands or indices of bands such as NDVI. Perform residual analysis and explore use of transformations for response variable in case of non-normality. Check for autocorrelation of residuals and for outliers.
  2. **Forward selection**. Explore other ways of selecting variables. Perhaps the reflectance of all pixels should be modeled as a function of the same seasonal and tidal predictors. It might have been best to either add both fourier terms (sin and cos) or neither, but not only one.
  3. **Seemingly Unrelated Regression**. Account for spatial dependence of hypothesis tests and p-values which renders the joint analysis invalid using @Leek approach.
  4. **Interpolation**. Compare several parametric variogram models and use cross validation to select best one. Explore interpolation of a transformation of p-values instead of p-values themselves to justify underlying gaussian process modeling used in kriging.
  5. **Smoothing**. Justify thresholding choices (p-value threshold and majority rule levels).

## References

Bivand, Roger S., Edzer Pebesma, and Virgilio Gómez-Rubio. 2013. *Applied Spatial Analysis with R, 2nd Ed.* ny: Springer.

Leek, Jeffrey T., and John D. Storey. 2008. "A General Framework for Multiple Testing Dependence." *Proceedings of the National Academy of Sciences of the United States of America* 105 (48): 18718–23.

Yamano, Hiroya, and Masayuki Tamura. 2004. "Detection Limits of Coral Reef Bleaching by Satellite Remote Sensing: Simulation and Data Analysis." *Remote Sensing of Environment* 90: 86–103.

Zhu, Zhe, and Curtis E. Woodcock. 2012. "Object-Based Cloud and Cloud Shadow Detection in Landsat Imagery." *Remote Sensing of Environment* 118: 83–94.

## 6 Installing rgeos and rgdal packages on MacOSX

Instructions for installing rgeos and rgdal packages on MacOSX:

- install xquartz, see http://www.xquartz.org/
- install GEOS and GDAL frameworks, see http://www.kyngchaos.com/files/software/frameworks/GDAL_Complete-1.11.dmg

- optionally XCode and Apple Command Line Developer Tools, see http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/
- optionally Fortran compiler, see http://stat.ethz.ch/CRAN/doc/manuals/r-release/R-admin.html#OS-X