Using Raster Data in R

Bernardo Blanco-Martin (TERN)

September 6, 2018

1 Introduction

A `raster` is a spatial (geographic) data structure that divides the space into elements of equal size (in units of the coordinate reference system) called `cells`. Cells have a square or rectangular shape and can store one or more values. Raster are also sometimes referred to as `grids` and cells as `pixels`. Rasters represent spatial data as a continuous. Raster are therefore in sharp contrast with the other main structure used to store and manipulate geographical data, `vectors`. Vectors represent discrete (i.e. object based) spatial data, such as points, lines, polygons.

The fundamental R package for working with spatial data in raster format is `raster`, originally developed by Robert J. Hijmans. The `raster` package provides classes and functions to create, read, manipulate and write raster data. In this tutorial we will describe and experiment with many of these classes and functions. A notable feature of this package is that it can work with very large spatial datasets that cannot be loaded into computer memory. Functions process these large datasets in chunks, without attempting to load all values into memory at once. `raster` has revolutionised the manipulation, geo-processing and analysis of raster data in R. More information on `raster` can be found in the package vignette here (https://cran.r-project.org/web/packages/raster/vignettes/Raster.pdf).

The `raster` package relies on the `rgdal` package to read, write, and geo-process raster data. `rgdal` provides bindings to the Geospatial Data Abstraction Library ('GDAL') and access to projection/transformation operations from the 'PROJ.4' library, both external to `rgdal`. It is also possible to call GDAL functionalities directly from R and the command line in a terminal (e.g. using the function `system()` in R). Many spatial software packages also use GDAL to read/write gridded spatial data (e.g. ArcGIS, QGIS, GRASS, etc.). More information on the GDAL can be found on this link (https://www.gdal.org/) and more information in PROJ.4 can be found in this link (https://proj4.org).

2 Raster Classes

The `raster` package creates and uses objects of several new classes. The main new classes provided by this package are: `RasterLayer`, `RasterStack`, and `RasterBrick`. Objects in these three classes are collectively referred as `Raster*` objects.

- `RasterLayer`: Object containing a single-layer raster.
- `RasterStack`: Object containing a multi-layer (band) raster. It can ''virtually'' connect several raster objects written to different files or in memory and/or a few layers in a single file.
- `RasterBrick`: Object containing a multi-layer (band) raster. It is a truly multi-layered object. It can only be linked to a single multi-layer file (i.e. all data must be stored in a single file on disk) or is in itself a multi-layer object with data loaded in memory. Typical examples of multi-layered raster files are multi-band satellite images and rasters containing time series (e.g. each layer contains values for a different day or month).

RasterStack and RasterBrick objects are quite similar. However, RasterBricks have a shorter processing time than RasterStacks and RasterStacks are more flexible (e.g. with RasterStacks pixel-based calculations on separate raster layers can be performed).

In both multi-layered object classes, individual layers must have the same spatial extent and resolution. That is, individual layers must represent the same locations with the same level of detail.

In all 3 classes the data can be loaded in memory or on disk depending on the size of the grid(s). Raster objects are typically created from files, but even RasterBrick objects can exist entirely in memory.

3 Preparation

```
# Load  Libraries
# ===============
library(ggplot2)
library(RColorBrewer)

library(sp)
library(rgdal)
library(ncdf4)
#library(RKEA)

library(raster)
library(rasterVis)

library(maps)
library(mapdata)
library(maptools)


# Clean up Memory
# ===============
rm(list=ls()) # WARNING: this will remove all objects in the *current* environment.
```

4 Creating Rasters

Raster objects can be created in memory with these functions:

- `raster` : Creates a RasterLayer object.
- `stack` : Creates a RasterStack object from RasterLayer (i.e. individual layers) objects or from a muliti-layer (band) file. They can also be created from a SpatialPixelsDataFrame object or SpatialGridDataFrame object. Individual raster layers must have the same spatial extent and resolution.
- `brick` : Ceates a RasterBrick object from RasterLayer (i.e. individual layers) objects, a RasterStack, or from a muliti-layer (band) file. They can also be created from SpatialPixels*, SpatialGrid*, and Extent objects. Individual raster layers must have the same spatial extent and resolution.

First we create a frame that will contain the Raster* object. This object will have defined: Dimensions, Spatial Extent, Resolution, and a Coordinate Reference System (CRS)

```
# Single Layer Rasters
# ===================

# Create the raster 'frame/skeleton/container'. Will have defined: Dimensions, Spatial Exten
t,
# Resolution, and CRS.
r1a = r1b = raster(nrow=10, ncol=15)
r2a  = r2b = raster(nrow=10, ncol=10, xmn=100, xmx=120, ymn=-30, ymx=0)
class(r2a)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

```
hasValues(r2a)
```

```
## [1] FALSE
```

```
r1a; r1b; r2a; r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
# Add contents (i.e. values) to the rasters
values(r1a) = 1   # OR: r1a[] = 1
values(r1b) = 2   # OR: r1b[] = 2
values(r2a) = 1:ncell(r2a)
values(r2b)  = rnorm(n=ncell(r2b))
hasValues(r2a)
```

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

```
r1a; r1b; r2a; r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 2, 2  (min, max)
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -2.044656, 2.425094  (min, max)
```

```
# Multiple- Layer Rasters
# ======================



# Stack
# -----
s1 = stack(r1a, r1b)
class(s1)
```

```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

```
nlayers(s1)
```

```
## [1] 2
```

```
s1
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
s2 = stack(r2a, r2b)
class(s2)
```

```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

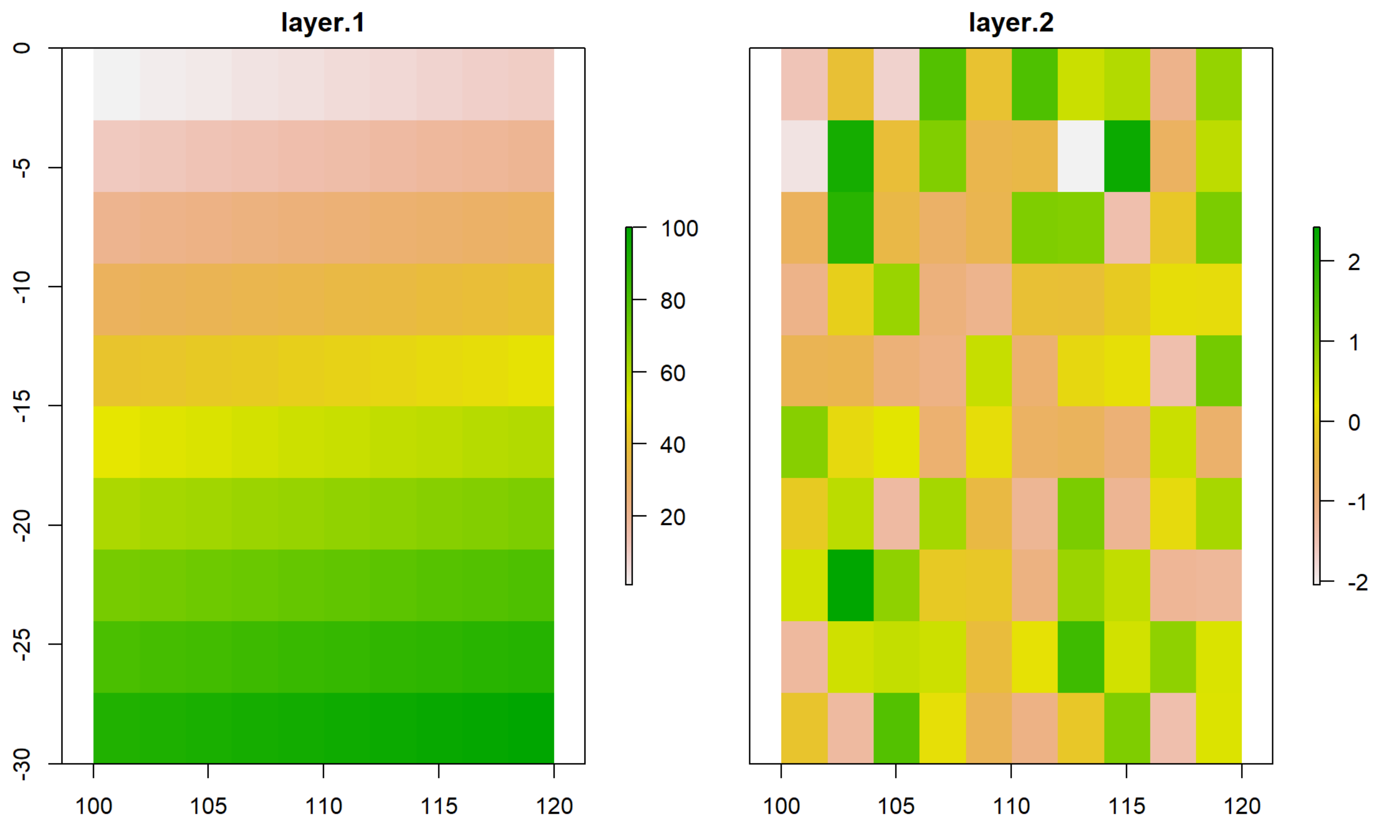
```
nlayers(s2)
```

```
## [1] 2
```

```
s2
```

```
## class       : RasterStack
## dimensions  : 10, 10, 100, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       :    layer.1,    layer.2
## min values  :   1.000000,  -2.044656
## max values  : 100.000000,   2.425094
```

```
plot(s2)
```



```
# Remember: Individual Raster Layers must have the same extent
#sNoNo = stack(r1a,r2b)  # It doesn't work


# Brick
# -----

# From RasterLayer objects
b1 = brick(r1a, r1b)
class(b1)
```

```
## [1] "RasterBrick"
## attr(,"package")
## [1] "raster"
```

```
nlayers(b1)
```

```
## [1] 2
```

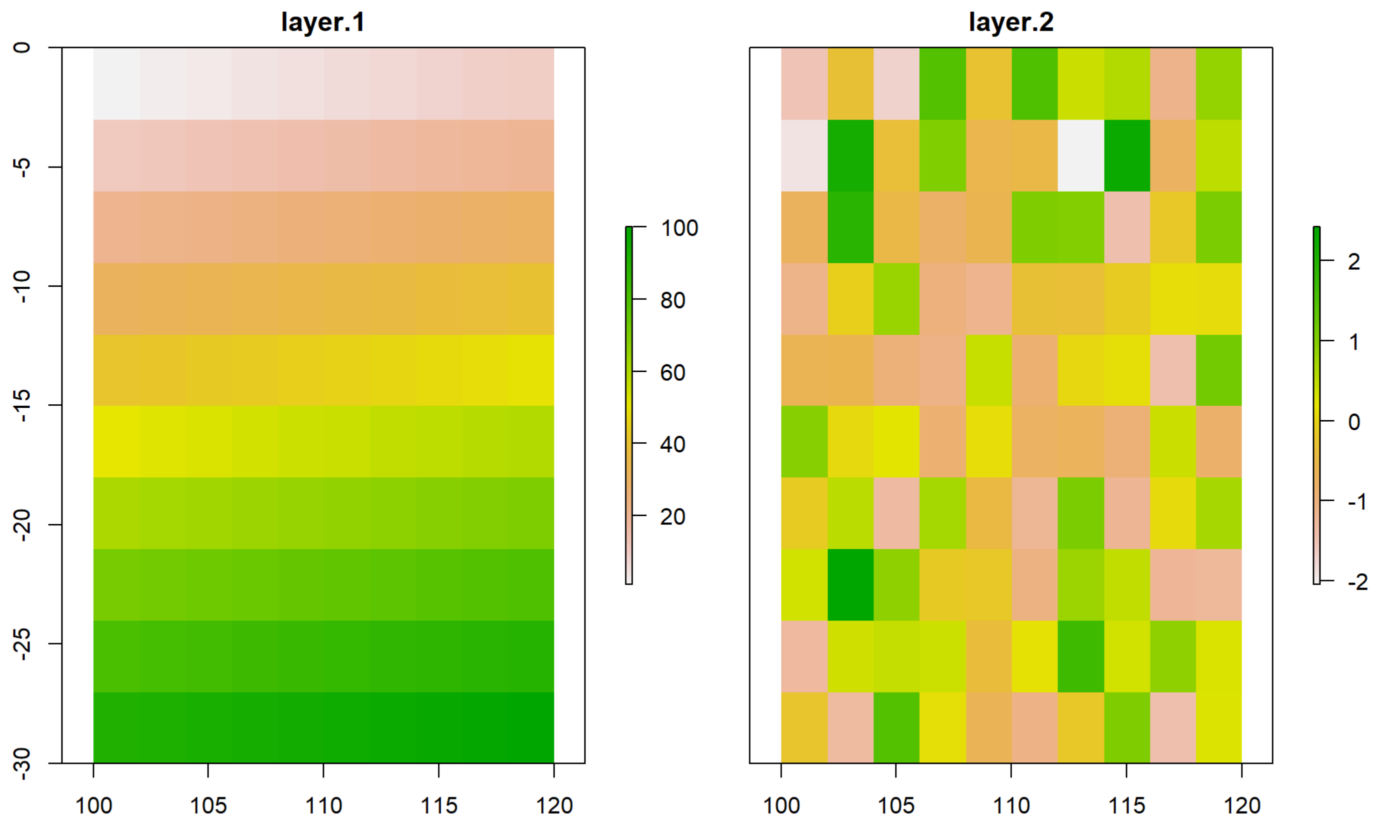
```
b1
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
# From a RasterStack object
b2 = brick(s2)
nlayers(b2)
```

```
## [1] 2
```

```
plot(b2)
```



```
# Again: Individual Raster Layers must have the same extent
#bNoNo = brick(r1a,r2b)  # It doesn't work; but why not giving it a try?



# Create (i.e. Extract) a Single Layer Raster from a Multi-Layer Raster
# =====================================================================

# From  a RasterStack object
r2a.from.s2 = raster(s2, layer=1)
r2a.from.s2
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1
## values      : 1, 100  (min, max)
```

```
r2a.from.s2[]
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
# From  a RasterBrick object
r2b.from.b2 = raster(b2, layer=2)
r2b.from.b2
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.2
## values      : -2.044656, 2.425094  (min, max)
```

```
r2b.from.b2[]
```

```
##   [1] -1.52322432 -0.30776275 -1.71457845  1.47006691 -0.26995228
##   [6]  1.52749419  0.42591433  0.61867298 -1.11413929  0.85554528
##  [11] -1.90030381  2.15823221 -0.33666087  1.03788802 -0.52687303
##  [16] -0.48263070 -2.04465641  2.28740979 -0.71837584  0.52209001
##  [21] -0.69384598  1.94081197 -0.48583950 -0.77500980 -0.56805548
##  [26]  1.04619975  1.02279357 -1.43466272 -0.17673206  1.09114564
##  [31] -1.08638711 -0.05508689  0.82014383 -0.96658245 -1.12770154
##  [36] -0.30931418 -0.31414472 -0.12533629  0.12581871  0.10097958
##  [41] -0.60371868 -0.57356988 -0.92817816 -1.05818817  0.44633439
##  [46] -0.87080332  0.04742747  0.13288168 -1.43279003  1.16134493
##  [51]  0.97640987  0.06071646  0.23153359 -0.86822777  0.11829384
##  [56] -0.73451187 -0.66518590 -0.91217523  0.41217501 -0.80912855
##  [61] -0.11945539  0.53400353 -1.32761692  0.72296647 -0.44381777
##  [66] -1.19127943  1.09394548 -1.18034857  0.08145673  0.69617817
##  [71]  0.37247587  2.42509433  0.90874652 -0.13667496 -0.17699776
##  [76] -1.01854426  0.80658420  0.48661195 -1.21921060 -1.25120262
##  [81] -1.28346663  0.38368417  0.47922564  0.39505050 -0.37601787
##  [86]  0.15420731  1.68880287  0.37410263  0.92019270  0.29987827
##  [91] -0.23819291 -1.30005631  1.47077510  0.13259741 -0.62370966
##  [96] -1.06265483 -0.17087935  1.05251022 -1.43406452  0.27182371
```

```
# Assigning names to raster layers
# =================================
# The function `names` can be used to assign and extract names to/from the layers of a Raster
\* object.

# For a single layer raster object
r1a.c = r1a
names(r1a.c) = c("r1a.c_Layer1")
names(r1a); names(r1a.c)
```

```
## [1] "layer"
```

```
## [1] "r1a.c_Layer1"
```

```
# For a multi=layer raster object
s1.c = s1
names(s1.c) = c("s1.c_Layer1", "s1.c_Layer2")
names(s1); names(s1.c)
```

```
## [1] "layer.1" "layer.2"
```

```
## [1] "s1.c_Layer1" "s1.c_Layer2"
```

4.1 Adding, Dropping, and Subsetting Layers

`addLayer` : Adds one or more layers to a Raster* object. The object returned is a RasterStack (unless nothing to add or drop was provided, in which case the original object is returned).

`dropLayer` : Drops one or more layers from a RasterStack or RasterBrick object.

- A layer cannot be dropped from a RasterLayer object (makes sense).

- If the multi-layer object has > 2 layers it returns an object of the same type,
- If the multi-layer object has 2 layers it returns a RasterLayer object.

`Subset` : In addition of using the function `raster` to extracts a layer or subset of layers from a RasterStack or RasterBrick object the function `subset` can also be used for to achieve tis task. The argument `subset` is an integer or character indicating the layers to extract. If a single layer is selected and the argument `drop` :

- `drop=TRUE` : The returned object would be of class RasterLayer.
- `drop=FALSE``: The returned object would be the same as that in the original raster object (i.e. RasterStack or RasterBrick).

```
# Adding Layers
# -------------
# It always returns a RasterStak object

# Adding to a RasterLayer object
s1.aL2 = addLayer(r1a, r1b) # = to 'stack(r1a,r2b)'
r1a; s1.aL2
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
# Adding to a RasterStack object
s1.aL3 = addLayer(s1, r1a)
s1; s1.aL3
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
## class       : RasterStack
## dimensions  : 10, 15, 150, 3  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2, layer
## min values  :       1,       2,     1
## max values  :       1,       2,     1
```

```
# Adding to a RasterBrick object
s1.aL4 = addLayer(b1, r1a, r1b)
b1; s1.aL4
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 4  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1.1, layer.2.1, layer.1.2, layer.2.2
## min values  :         1,         2,         1,         2
## max values  :         1,         2,         1,         2
```

```
# Dropping Layers
# ---------------
# It always returns a RasterStak object

# Dropping from a RasterLayer object
dim(r1a)
```

```
## [1] 10 15  1
```

```
#s1.dL0 = dropLayer(r1a, i=1)  # It doesn't work (cannot drop a layer from a RasterLayer obje
ct); but why not giving it a try!
# r1a; s1.dL02

# Dropping from a RasterStack object
s.dL2 = dropLayer(s1.aL4, i=c(2,4))
s1.aL4; s.dL2
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 4  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1.1, layer.2.1, layer.1.2, layer.2.2
## min values  :         1,         2,         1,         2
## max values  :         1,         2,         1,         2
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1.1, layer.1.2
## min values  :         1,         1
## max values  :         1,         1
```

```
# Dropping from a RasterBrick object
  # Dropping from a > 2 Layers brick returns a 'RasterStack' object
b1.aL3 = brick(addLayer(s1, r1a))
b1.aL3
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 3  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2, layer
## min values  :       1,       2,     1
## max values  :       1,       2,     1
```

```
s.aL2 = dropLayer(b1.aL3, i=3)
b1.aL3; s.aL2
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 3  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2, layer
## min values  :       1,       2,     1
## max values  :       1,       2,     1
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
  # Dropping from a 2 Layers brick returns a 'RasterLayer' object
s.dL1 = dropLayer(b1, i=1)
b1; s.dL1
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       1,       2
## max values  :       1,       2
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.2
## values      : 2, 2  (min, max)
```

```
# Subsetting Layers
# -----------------

dim(s1.aL4)
```

```
## [1] 10 15  4
```

```
names(s1.aL4)
```

```
## [1] "layer.1.1" "layer.2.1" "layer.1.2" "layer.2.2"
```

```
# Subset multiple ayers
s1.aL4.s1n2 = subset(s1.aL4, subset=1:2)
dim(s1.aL4.s1n2)
```

```
## [1] 10 15  2
```

```
names(s1.aL4.s1n2)
```

```
## [1] "layer.1" "layer.2"
```

```
# Subset a single layer
  # drop=TRUE
s1.aL4.s3dT = subset(s1.aL4, subset="layer.1.2",drop=TRUE)
dim(s1.aL4.s3dT)
```

```
## [1] 10 15  1
```

```
names(s1.aL4.s3dT)
```

```
## [1] "layer.1"
```

```
class(s1.aL4.s3dT)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

```
  # drop=FALSE
s1.aL4.s3dF = subset(s1.aL4, subset=3,drop=FALSE)
dim(s1.aL4.s3dF)
```

```
## [1] 10 15  1
```

```
names(s1.aL4.s3dF)
```

```
## [1] "layer.1"
```

```
class(s1.aL4.s3dF)
```

```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

5 Loading and Saving Rasters

```
# Choose a Directory where to store the save raster file:
# my.file.path = "C:\\Users\\uqbblanc\\Documents\\TERN\\CWDir"
my.file.path = getwd()
```

5.1 Saving Rasters

We can use two functions:

- `writeRaster` : Writes an entire Raster* object to a file in a supported format.
- `writeValues` : Writes chucks (e.g. by row) of a Raster* object to a file in a supported format.

Supported formats include (in parenthesis is the file extension for the format). The last 3 formats cannot store Multi-band rasters. See help on 'writeRaster' and 'writeFormat' for further details:

- native R 'raster' package format (.grd): It conserves the original file names in the individual band names.
- netCDF (.nc): Requires the library `ncdf4` .
- GeoTiff (.tif): Requires the library `rgdal` .
- ENVI .hdr labelled (.envi),
- ESRI hdr. labelled (.bil)
- Erdas Imagine Images (.img)
- ESRI ASCII (.asc)
- SAGA GIS (.sdat),
- IDRISI (.rst).

If argument `prj` is TRUE, the CRS is written to a '.prj' file. This can be useful when writing to a file type that does not store the crs, such as ASCII files.

For multi-layer rasters (RasterStack and RasterBrick objects), individual layers can be stored in the same file or in separate files. When argument `bylayer` is TRUE, layers are saved to separate files. In this case, to name the separate files the user provide naming directions using the argument 'suffix':

- A vector with filenames, with as many file names as layers (i.e. names(x))
- A single filename that will get a unique suffix (i.e. a number between 1 and nlayers(x))

```
# Single layer raster (RasterLayer object)
# ----------------------------------------

# Save a single layer from a single layer raster (RasterLayer object) in GeoTiff format (requ
ires library 'rgdal')
writeRaster(r2a, filename=paste(my.file.path,"r2a.tif",sep="\\"), format="GTiff", overwrite=T
RUE)



# Multi-layer rasters (RasterStack and RasterBrick objects)
# ---------------------------------------------------------

# Save all layers of a RasterStack object in a single file in netCDF format (requires library
'ncdf4')
writeRaster(s2, filename=paste(my.file.path,"s2.nc",sep="\\"), format="CDF", overwrite=TRUE)


# Save individual layers of a RasterBrick object in separate files in native R 'raster' packa
ge format
writeRaster(b2,  filename=paste(my.file.path,"b2.grd",sep="\\"),  format="raster", overwrite=
TRUE,
                   bylayer=TRUE)
```

5.2 Loading Rasters

Loading rasters is done with the same functions that are used to create them from data (i.e. `raster` , `stack` , and `brick`).

Note that when extracting a single layer from a multi-layer raster:

- Argument `layer` is used for extracting the layer from a RasterStack or RasterBrick object or a File.
- Argument `band` is used for extracting the layer from a File (exclusively).

```
# Single layer raster (RasterLayer object)
# ----------------------------------------


# Load single Layer from a single layer file
r2a.from.r2a.tif = raster(paste(my.file.path,"r2a.tif",sep="\\"))
hasValues(r2a.from.r2a.tif)
```

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

```
inMemory(r2a.from.r2a.tif)
```

```
## [1] FALSE
```

```
r2a.from.r2a.tif
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : C:/Users/uqbblanc/Documents/TERN/04b-DSDP_GitHub/Lanscape_AusCover-RemoteSen
sing/UsingRasterDatainR/r2a.tif
## names       : r2a
## values      : 1, 100  (min, max)
```

```
r2a.from.r2a.tif[]
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
# Load single Layer from a multi-layer file

# Layer 1, using argument 'layer'
r2a.from.s2.nc = raster(paste(my.file.path,"s2.nc",sep="\\"), layer=1)
hasValues(r2a.from.s2.nc)
```

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

```
inMemory(r2a.from.s2.nc)
```

```
## [1] FALSE
```

```
r2a.from.s2.nc
```

```
## class       : RasterLayer 
## band        : 1  (of  2  bands)
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : C:/Users/uqbblanc/Documents/TERN/04b-DSDP_GitHub/Lanscape_AusCover-RemoteSen
sing/UsingRasterDatainR/s2.nc 
## names       : variable 
## z-value     : 1 
## zvar        : variable
```

```
r2a.from.s2.nc[]
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
# Layer 2, using argument 'band'
r2b.from.s2.nc = raster(paste(my.file.path,"s2.nc",sep="\\"), band=2)
hasValues(r2b.from.s2.nc)
```

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

```
inMemory(r2b.from.s2.nc)
```

```
## [1] FALSE
```

```
r2b.from.s2.nc
```

```
## class       : RasterLayer 
## band        : 2  (of  2  bands)
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : C:/Users/uqbblanc/Documents/TERN/04b-DSDP_GitHub/Lanscape_AusCover-RemoteSen
sing/UsingRasterDatainR/s2.nc 
## names       : variable 
## z-value     : 2 
## zvar        : variable
```

```
r2b.from.s2.nc[]
```

```
##   [1] -1.52322435 -0.30776274 -1.71457839  1.47006691 -0.26995227
##   [6]  1.52749419  0.42591432  0.61867297 -1.11413932  0.85554528
##  [11] -1.90030384  2.15823221 -0.33666086  1.03788805 -0.52687305
##  [16] -0.48263070 -2.04465652  2.28740978 -0.71837586  0.52209002
##  [21] -0.69384599  1.94081199 -0.48583952 -0.77500981 -0.56805545
##  [26]  1.04619980  1.02279353 -1.43466270 -0.17673206  1.09114563
##  [31] -1.08638716 -0.05508690  0.82014382 -0.96658242 -1.12770152
##  [36] -0.30931419 -0.31414473 -0.12533629  0.12581871  0.10097957
##  [41] -0.60371870 -0.57356989 -0.92817813 -1.05818820  0.44633439
##  [46] -0.87080330  0.04742747  0.13288169 -1.43279004  1.16134489
##  [51]  0.97640985  0.06071646  0.23153359 -0.86822778  0.11829384
##  [56] -0.73451185 -0.66518593 -0.91217524  0.41217500 -0.80912852
##  [61] -0.11945539  0.53400356 -1.32761693  0.72296649 -0.44381776
##  [66] -1.19127941  1.09394550 -1.18034852  0.08145673  0.69617820
##  [71]  0.37247586  2.42509437  0.90874654 -0.13667497 -0.17699775
##  [76] -1.01854432  0.80658418  0.48661196 -1.21921062 -1.25120258
##  [81] -1.28346658  0.38368416  0.47922564  0.39505050 -0.37601787
##  [86]  0.15420730  1.68880284  0.37410262  0.92019272  0.29987827
##  [91] -0.23819292 -1.30005634  1.47077513  0.13259742 -0.62370968
##  [96] -1.06265485 -0.17087935  1.05251026 -1.43406451  0.27182370
```

```
# Multi-layer raster
# ------------------

# stack (RasterStack object)
s2.from.s2.nc = stack(paste(my.file.path,"s2.nc",sep="\\"))
hasValues(s2.from.s2.nc)
```

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

```
inMemory(s2.from.s2.nc)
```

```
## [1] FALSE
```

```
s2.from.s2.nc
```

```
## class       : RasterStack
## dimensions  : 10, 10, 100, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : X1, X2
```

```
head(s2.from.s2.nc[])
```

```
##      X1         X2
## [1,]  1 -1.5232244
## [2,]  2 -0.3077627
## [3,]  3 -1.7145784
## [4,]  4  1.4700669
## [5,]  5 -0.2699523
## [6,]  6  1.5274942
```

```
# brick (RasterBrick object)
  # From b2.grd: Cannot open it because layers were saved in separated files
# b2.from.b2.grd = brick(paste(my.file.path,"b2.grd",sep="\\"))
  # From s2.nc
b2.from.s2.grd = brick(paste(my.file.path,"s2.nc",sep="\\"))
hasValues(b2.from.s2.grd)
```

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

```
inMemory(b2.from.s2.grd)
```

```
## [1] FALSE
```

```
b2.from.s2.grd
```

```
## class       : RasterBrick
## dimensions  : 10, 10, 100, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : C:/Users/uqbblanc/Documents/TERN/04b-DSDP_GitHub/Lanscape_AusCover-RemoteSen
sing/UsingRasterDatainR/s2.nc
## names       : X1, X2
## unknown     : 1, 2
## varname     : variable
```

```
head(b2.from.s2.grd[])
```

```
##      X1         X2
## [1,]  1 -1.5232244
## [2,]  2 -0.3077627
## [3,]  3 -1.7145784
## [4,]  4  1.4700669
## [5,]  5 -0.2699523
## [6,]  6  1.5274942
```

6 Navigating Rasters

To navegate rasters we use raster navigation functions, also known as ‘helper’ functions.

Dichotomy: A RasterLayer can be accessed both as a Matrix (via cell row and column numbers) and as a Vector (via cell number).

- *Conceptually*: Easier to think of a RasterLayer as a matrix.
- *In pratice*: Commonly treated as a vector, constituted by the RasterLayer cells from top-left corner to bottom-right ordered by row.

Except `dim` and `names` , which is not really Helper Function, all functions return the same results for uni-layer and multi-layer objects (compose of layers of the same characteristics than the single layer).

6.0.1 Dimensions of a raster

```
# Dimension of a raster
dim(r1a); dim(s1); dim(b1)
```

```
## [1] 10 15  1
```

```
## [1] 10 15  2
```

```
## [1] 10 15  2
```

6.0.2 Names of raster

```
# Get the names of the layers of a raster object
# (it can also be used to set the layers names, see "Assigning names to raster layers").
names(r1a); names(s1); names(b1)
```

```
## [1] "layer"
```

```
## [1] "layer.1" "layer.2"
```

```
## [1] "layer.1" "layer.2"
```

6.0.3 Number of rows, columns, and cells

```
# Number of rows, columns, and cells
nrow(r1a); nrow(s1); nrow(b1)
```

```
## [1] 10
```

```
## [1] 10
```

```
## [1] 10
```

```
ncol(r1a); ncol(s1); ncol(b1)
```

```
## [1] 15
## [1] 15
```

```
## [1] 15
```

```
ncell(r1a); ncell(s1); ncell(b1)
```

```
## [1] 150
```

```
## [1] 150
```

```
## [1] 150
```

```
nlayers(r1a); nlayers(s1); nlayers(b1)
```

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

```
## [1] 2
```

```
## [1] 2
```

6.0.4 Mapping Cells

```
# Mapping cells
 # From Vector format to Matrix format
rowFromCell(r1a, 30); rowFromCell(s1, 30); rowFromCell(b1, 30)
```

```
## [1] 2
```

```
## [1] 2
```

```
## [1] 2
```

```
colFromCell(r1a, 30); colFromCell(s1, 30); colFromCell(b1, 30)
```

```
## [1] 15
```

```
## [1] 15
```

```
## [1] 15
```

```
 # From Matrix format to Vector format
cellFromRowCol(r1a, 2, 5); cellFromRowCol(s1, 2, 5); cellFromRowCol(b1, 2, 5)
```

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

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

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

```
cellFromRowCol(r2a, row=1:2, col=1:3); cellFromRowCol(s2, row=1:2, col=1:3); cellFromRowCol(b
2, row=1:2, col=1:3)  # Returns cell numbers for each row/col pair (i.e 1,1; 2,2; 1,3)
```

```
## [1]  1 12  3
```

```
## [1]  1 12  3
```

```
## [1]  1 12  3
```

```
cellFromRowColCombine(r2a, rownr=1:2, colnr=1:3); cellFromRowColCombine(b2, rownr=1:2, colnr=
1:3); cellFromRowColCombine(s2, rownr=1:2, colnr=1:3)  # Returns cell numbers for all row/col
pairs (1,1; 1,2;....;2,3)
```

```
## [1]  1  2  3 11 12 13
```

```
## [1]  1  2  3 11 12 13
```

```
## [1]  1  2  3 11 12 13
```

6.0.5 Coordinates for the center of raster cells

```
# Coordinates for the center of raster cells
 # Coordinates from Columns (1 column or more)
xFromCol(r1a,1); xFromCol(s1,1); xFromCol(b1,1)
```

```
## [1] -168
```

```
## [1] -168
```

```
## [1] -168
```

```
xFromCol(r1a,14:15); xFromCol(s1,14:15); xFromCol(b1,14:15)
```

```
## [1] 144 168
```

```
## [1] 144 168
## [1] 144 168
```

```
 # Coordinates from Rows (1 row or more)
xFromCol(r1a,1); xFromCol(s1,1); xFromCol(b1,1)
```

```
## [1] -168
```

```
## [1] -168
```

```
## [1] -168
```

```
xFromCol(r1a, 9:10); xFromCol(s1, 9:10); xFromCol(b1, 9:10)
```

```
## [1] 24 48
```

```
## [1] 24 48
```

```
## [1] 24 48
```

```
 # Coordinates from Cells
xyFromCell(r1a,1); xyFromCell(s1,1); xyFromCell(b1,1)   # First cell
```

```
##         x  y
## [1,] -168 81
```

```
##         x  y
## [1,] -168 81
```

```
##         x  y
## [1,] -168 81
```

```
xyFromCell(r1a, ncell(r1a)); xyFromCell(s1, ncell(s1)); xyFromCell(b1, ncell(b1))  # Last cel
l
```

```
##        x   y
## [1,] 168 -81
```

```
##        x   y
## [1,] 168 -81
```

```
##        x   y
## [1,] 168 -81
```

```
 # Longitude only
xFromCell(r1a,1); xFromCell(s1,1); xFromCell(b1,1)  # First cell
```

```
## [1] -168
```

```
## [1] -168
```

```
## [1] -168
```

```
 # Latitude only
yFromCell(r1a, ncell(r1a)); yFromCell(s1, ncell(s1)); yFromCell(b1, ncell(b1))  # Last cell
```

```
## [1] -81
```

```
## [1] -81
```

```
## [1] -81
```

6.0.6 Columns, Rows, and Cells from Coordinates (at the centre of cells)

```
# Columns, Rows, and Cells from Coordinates (at centre of cells)
colFromX(r1a, 90) # 12
```

```
## [1] 12
```

```
rowFromY(r1a, -45) # 8
```

```
## [1] 8
```

```
cellFromXY(r1a, cbind(90,-45))  # 117 = 7 * 15 + 12; ncol(r1a) = 15
```

```
## [1] 117
```

7 Examining Raster Contents

In this section we will examine the contents (i.e. values) of each cell in the raster. The methods to summarize this contents are described in the next section.

7.1 Examining raster contents numerically

7.1.1 Single Layer (RasterLayer) Objects

```
# By cell position
# ................

# All values
r2a[]
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
values(r2a)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
getValues(r2a)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
# Particular cells
r2a[1,1]; r2a[10,10]
```

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

```
## [1] 100
```

```
#values(r2a[1]); values(r2a[1,1])  # WRONG: It doesn't work; but why not giving it a try?
#getValues(r2a[1]); getValues(r2a[1,1]) # WRONG: It doesn't work; but why not giving it a tr
y?

# Values of a row
r2a[2,]
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
getValues(r2a, row=2) # RIGHT: It works ONLY for Rows (not individual Cells or Columns)
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
# Values of a column
r2a[,3]
```

```
##  [1]  3 13 23 33 43 53 63 73 83 93
```

```
#getValues(r2a, col=3)  # WRONG: It doesn't work

# Values of a block (rectangle) of cellsgetValuesBlock
getValuesBlock(r2a, row=1, nrows=2, col=1, ncols=3)
```

```
## [1]  1  2  3 11 12 13
```

```
extract(r2a, cellFromRowColCombine(r2a, rownr=1:2, colnr=1:3))
```

```
## [1]  1  2  3 11 12 13
```

```
# By coordinates
# ..............
# Extract returns raster cells values from both cell indices and cell coordinates.
cells.indices = cellFromRowColCombine(r2a, rownr=1:2, colnr=1:3)
cells.coords = xyFromCell(r2a, cells.indices)
cells.indices
```

```
## [1]  1  2  3 11 12 13
```

```
cells.coords
```

```
##        x    y
## [1,] 101 -1.5
## [2,] 103 -1.5
## [3,] 105 -1.5
## [4,] 101 -4.5
## [5,] 103 -4.5
## [6,] 105 -4.5
```

```
extract(r2a, cells.indices)
```

```
## [1]  1  2  3 11 12 13
```

```
extract(r2a, cells.coords)
```

```
## [1]  1  2  3 11 12 13
```

7.1.2 Multi-Layer (RasterStack and RasterBrick) Objects

For multi-layer rasters the procedures are analogous to those obtained for single-layers rasters, but results include the relevant values for all the layers in the object.

RasterStack and RasterBrick objects would return the same results if their contents are equal (e.g. s2 & b2 return the same results)

```
# By cell position
# ................

# All values
head(s2[])
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]       4  1.4700669
## [5,]       5 -0.2699523
## [6,]       6  1.5274942
```

```
head(values(s2))
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]       4  1.4700669
## [5,]       5 -0.2699523
## [6,]       6  1.5274942
```

```
head(getValues(s2))
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]       4  1.4700669
## [5,]       5 -0.2699523
## [6,]       6  1.5274942
```

```
# Particular cells
s2[1,1]; s2[10,10]
```

```
##      layer.1   layer.2
## [1,]       1 -1.523224
```

```
##      layer.1   layer.2
## [1,]     100 0.2718237
```

```
#values(s2[1]); values(s2[1,1])   # WRONG: It doesn't work; but why not giving it a try?
#getValues(s2[1]); getValues(s2[1,1])   # WRONG: It doesn't work; but why not giving it a tr
y?

# Values of a row
s2[2,]
```

```
##       layer.1    layer.2
##  [1,]      11 -1.9003038
##  [2,]      12  2.1582322
##  [3,]      13 -0.3366609
##  [4,]      14  1.0378880
##  [5,]      15 -0.5268730
##  [6,]      16 -0.4826307
##  [7,]      17 -2.0446564
##  [8,]      18  2.2874098
##  [9,]      19 -0.7183758
## [10,]      20  0.5220900
```

```
getValues(s2, row=2)
```

```
##       layer.1    layer.2
##  [1,]      11 -1.9003038
##  [2,]      12  2.1582322
##  [3,]      13 -0.3366609
##  [4,]      14  1.0378880
##  [5,]      15 -0.5268730
##  [6,]      16 -0.4826307
##  [7,]      17 -2.0446564
##  [8,]      18  2.2874098
##  [9,]      19 -0.7183758
## [10,]      20  0.5220900
```

```
# Values of a column
s2[,3]
```

```
##       layer.1    layer.2
##  [1,]       3 -1.7145784
##  [2,]      13 -0.3366609
##  [3,]      23 -0.4858395
##  [4,]      33  0.8201438
##  [5,]      43 -0.9281782
##  [6,]      53  0.2315336
##  [7,]      63 -1.3276169
##  [8,]      73  0.9087465
##  [9,]      83  0.4792256
## [10,]      93  1.4707751
```

```
#getValues(s2, col=3)     # WRONG: It doesn't work; but why not giving it a

# Values of a block (rectangle) of cellsgetValuesBlock
getValuesBlock(s2, row=1, nrows=2, col=1, ncols=3)
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]      11 -1.9003038
## [5,]      12  2.1582322
## [6,]      13 -0.3366609
```

```
extract(s2, cellFromRowColCombine(s2, rownr=1:2, colnr=1:3))
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]      11 -1.9003038
## [5,]      12  2.1582322
## [6,]      13 -0.3366609
```

```
# By coordinates
# ....................
# Extract returns raster cells values from both cell indices and cell coordinates.
cells.indices = cellFromRowColCombine(s2, rownr=1:2, colnr=1:3)
cells.coords = xyFromCell(s2, cells.indices)
cells.indices
```

```
## [1]  1  2  3 11 12 13
```

```
cells.coords
```

```
##        x    y
## [1,] 101 -1.5
## [2,] 103 -1.5
## [3,] 105 -1.5
## [4,] 101 -4.5
## [5,] 103 -4.5
## [6,] 105 -4.5
```

```
extract(s2, cells.indices)
```

```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]      11 -1.9003038
## [5,]      12  2.1582322
## [6,]      13 -0.3366609
```

```
extract(s2, cells.coords)
```

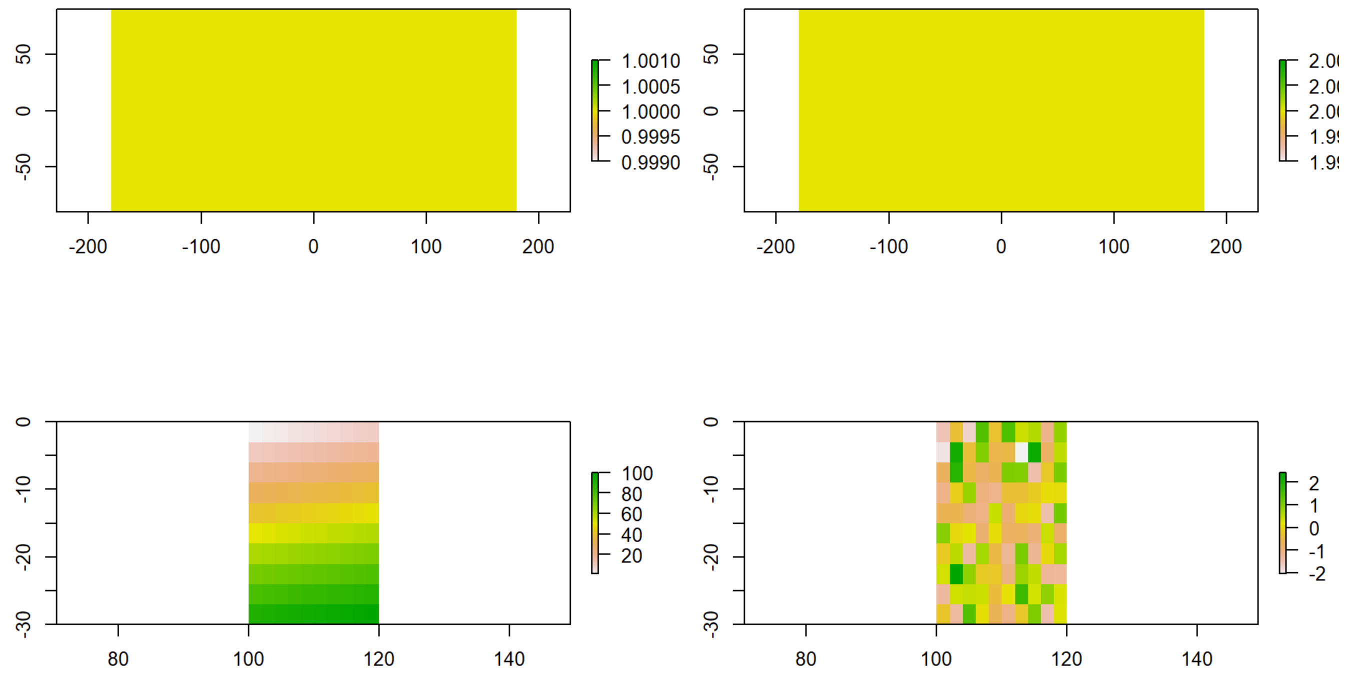
```
##      layer.1    layer.2
## [1,]       1 -1.5232243
## [2,]       2 -0.3077627
## [3,]       3 -1.7145784
## [4,]      11 -1.9003038
## [5,]      12  2.1582322
## [6,]      13 -0.3366609
```

7.2 Visualising raster contents

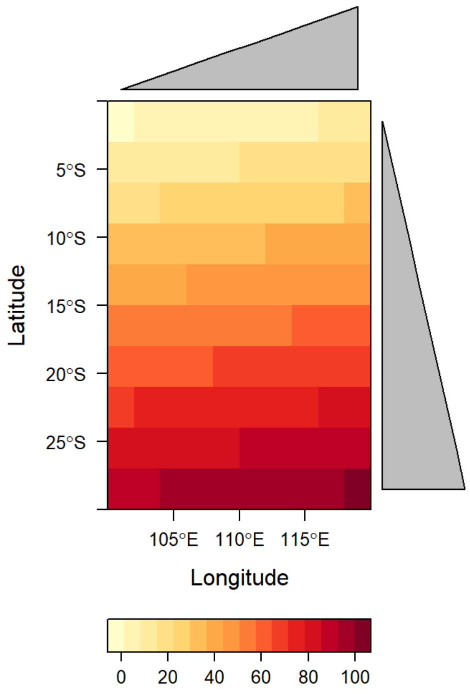
Rasters can be visualised using the function `plot` in the package `raster`. Even better visualisations can be easily achieved with the function `levelplot` in the package `rasterVis`.

7.2.1 Single Layer (RasterLayer) Objects

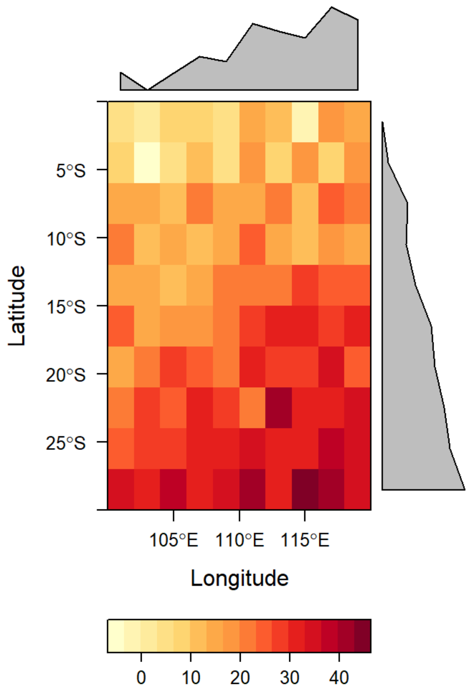
```
# Basic plot
par(mfrow=c(2,2))
plot(r1a)
plot(r1b)
plot(r2a)
plot(r2b)
```



```
# Nicer plot (using package raterVis)
levelplot(r2a, par.settings=YlOrRdTheme)
```



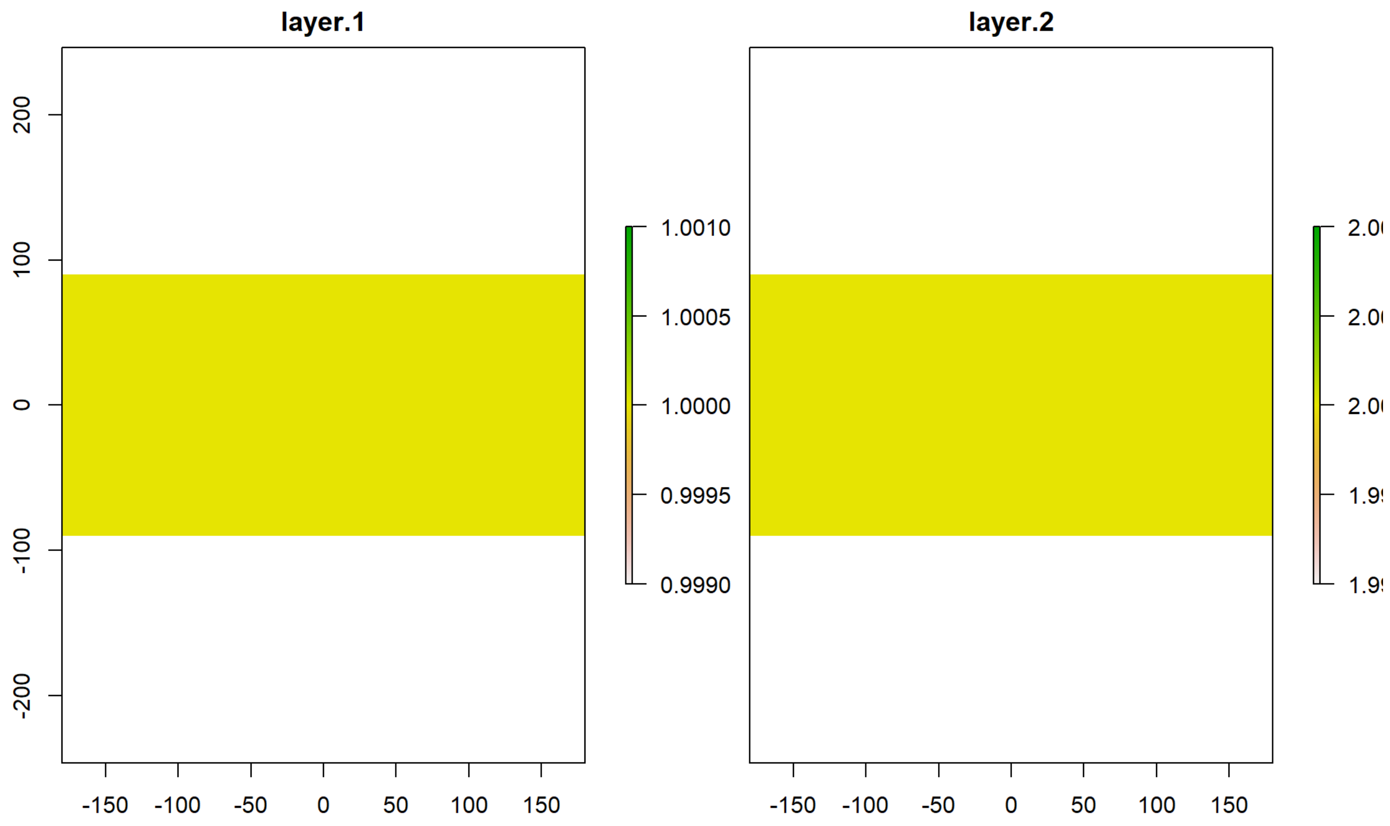
```
# Plot something more interesting: Latitudinal and Longitudinal gradients, with former 3 time
s stronger
# Could (should) also implement it using: lapply or mapply
# Used 'for loop' for clarity
for(i in 1:nrow(r2b)) {
    for(j in 1:ncol(r2b)) {
            r2b[i,j]  = 3*i + j + rnorm(n=1, mean=0, sd=5)
    }
}
levelplot(r2b, par.settings=YlOrRdTheme)
```



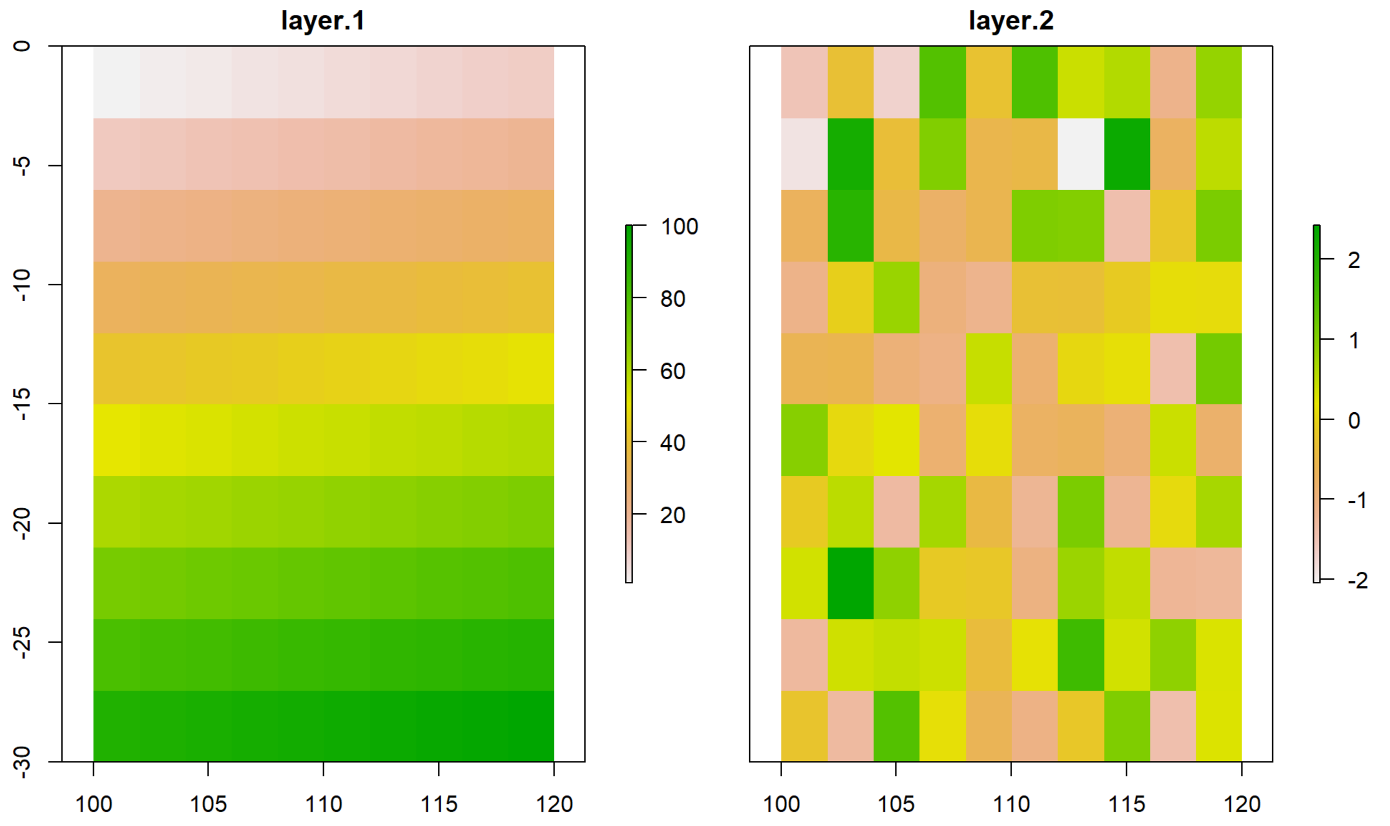
7.2.2 Multi-Layer (RasterStack and RasterBrick) Objects

Works similarly than for single-layer rasters, but for multi-layered rasters all layers are plotted.

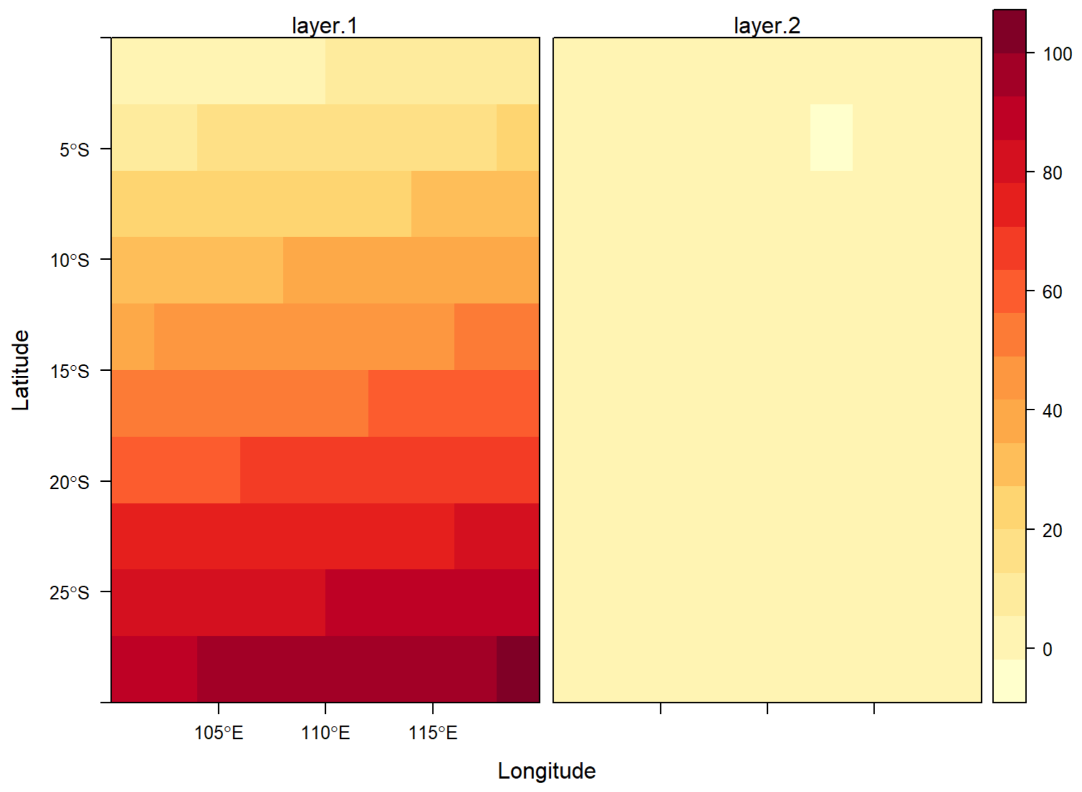
```
# Basic plot
# par(mfrow=c(2,2))  # 'par': Doesn't work for plotting multi-layer objects.
plot(s1)
```



```
plot(s2)
```



```
# Nicer plot (using package raterVis)
levelplot(s2, par.settings=YlOrRdTheme)
```



8 Summarising raster contents

We will explore how to summarise the raster contents both numerically and visually.

8.1 Numerical Summaries

We will examine how compute numerical summaries for:

- The **whole raster layer**. `cellStats` , `freq` , and `crosstab` are part of the `raster` package. `summary` is part of the `base` package, but it includes a method for Raster* objects. The three functions from the raster packacge can be used for rasters with multiple-layers. However, `summary` can only be used with RasterLayer objects. We can produce:
    - *Summary statistics for a single layer*: The function `cellStats` provides a single summary statistics (e.g. max, mean, min, sd, sum). Some of these summary statistics will fail with very large Raster* objects. For Raster* objects the function `summary` returns the: minimum, 1st quartile, median, 3rd quartile, maximum, and the number of cells containing missing data.
    - *Tables for a single layer*: Frequency and Count table, using the function `freq`
    - *Tables to compare the contents of 2 RasterLayer objects* (i.e. cross-tabulate), using the function `crosstab`
- **Portions of a raster layer** using zonal statistics, via the function `zonal` . This function can also be used with multiple-layers.

- The **neighbourhood of the cells** using a moving window, via the function ‘focal’ `focal` uses a the moving window to calculate the values for the neighbourhood of the focal cell. The moving window shape and values are defined using a matrix of weights with a function. In image processing moving windows (generally a moving average) are used to smooth a data set; that is, to attempt to remove the noise and/or fine-scale structures capturing the main patters in the data. `focal’ currently only works for (single layer) RasterLayer objects. Its output is a new raster. Peculiarities of this function:

1. *Weight Matrix*: Must have an odd number of rows and columns. If an odd number is required acol/row of weight 0 can be added.
2. *Weight Function*:
- Must take multiple numbers and return one (e.g. 'mean', 'modal', 'max', 'min',…)
- Must accept a 'na.rm' argument (e.g. 'length' will fail)
- Default weight function is 'sum': It is computationally more efficient to adjust the weights-matrix, than to use some other summary functions through the 'fun' argument (i.e. can compute mean using fractions, see below).

```
# Create some raster objects

 # Layer containing integers randomly sampled
r3 = raster(nrow=10, ncol=15)
values(r3) = sample(1:5, size=ncell(r3), replace=TRUE, prob=c(0.05,0.05,0.2,0.3,0.4))
values(r3)
```

```
##   [1] 5 3 5 3 3 5 3 4 5 5 5 5 4 4 4 4 4 2 4 4 3 4 4 5 4 1 4 5 4 4 5 4 3 1 5
##  [36] 4 5 4 3 4 3 5 3 3 4 5 3 4 3 5 5 4 1 5 4 5 2 5 4 4 4 3 5 1 4 4 4 5 3 4
##  [71] 3 3 3 4 3 3 3 1 3 4 3 3 4 5 4 4 5 4 1 3 5 1 5 5 3 5 3 5 5 3 3 5 4 5 5
## [106] 5 2 5 5 3 5 5 5 3 5 3 4 4 5 3 1 1 5 5 3 4 4 4 3 1 3 3 4 5 4 3 3 3 5 5
## [141] 3 4 1 4 1 5 3 4 5 5
```

```
 # Layer containing real numbers randomly sampled with an increasing gradient
 # (gradient increses with cell possition in vector, not matrix)
r4 = raster(nrow=10, ncol=15)
r4[] = runif(ncell(r4)) * 1:ncell(r4)
r4[]
```

```
##   [1]   0.159413   1.071552   1.561663   2.269360   2.409569   2.629865
##   [7]   6.100374   7.632286   3.784776   8.212264   2.549443   9.185763
##  [13]   3.229163  13.100792  14.600212   8.546041   5.600258  14.491976
##  [19]  10.129539  11.217062  14.247378   5.837290   9.644538  18.983006
##  [25]   2.444690  17.906982  14.618086  18.537331   6.982622  15.628578
##  [31]  20.314213  24.368777   4.348606   8.973618   8.766104  12.703006
##  [37]   3.745505   8.169764  29.488471  31.847831  16.048666  32.264140
##  [43]  33.869145   5.210650  19.970549   6.010358  22.502558  18.027698
##  [49]  13.423777  11.488582  40.950344  23.377495  36.490071  33.699179
##  [55]   5.304786  27.065318  47.951720  55.169631  43.140661  32.482320
##  [61]  37.305186  24.511580  24.508240  38.579269  36.842969  42.147970
##  [67]  58.863392  27.694685   3.615630  49.812642  48.323142  25.532526
##  [73]  13.243439  36.810088   8.590821  39.033050   9.682831  62.736131
##  [79]  33.736766  69.082717  29.509802   3.031946   7.162494   3.465050
##  [85]  22.998674  68.531395  26.262978  10.375475  28.725136  41.614928
##  [91]  15.114028  91.262658  72.513764  58.503332  68.773167  34.630875
##  [97]  96.935014  89.347696  43.533672  76.902481  98.473494  66.646860
## [103]  45.391521  65.110114  25.776254  18.206586 105.659733  39.975791
## [109]  90.250430  34.397903  97.811142  82.114862  47.381777  46.571896
## [115]  10.953531  96.572556  67.305385  91.319061  36.552894  55.426992
## [121]  52.562468  95.056791   5.342052 121.767273 106.449398  31.185296
## [127] 101.096465 104.736674  20.198691  59.847885  79.462719 105.971794
## [133]  66.685377  61.704396  30.724971  27.494896   7.372462 129.485085
## [139]  41.405577 113.643762 101.815768  76.179629  95.494443  37.794413
## [145] 141.583838  10.716541  61.795628  54.856640 143.994765  75.800610
```

```
# Summarizing the whole Raster Layer
# ---------------------------------

# mean(r3), min(r3), max(3): Do not work. -> They return a RasterLayer object

# Summary Stats. for the whole layer, one summary statistics at the time: cellStats (from the
raster package)
cellStats(r3, mean)
```

```
## [1] 3.766667
```

```
cellStats(r3, min)
```

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

```
cellStats(r3, max)
```

```
## [1] 5
```

```
# Summary Stats. for the whole layer, several summary statistics at the time: summary (from t
he base package)
summary(r3)
```

```
##         layer
## Min.        1
## 1st Qu.     3
## Median      4
## 3rd Qu.     5
## Max.        5
## NA's        0
```

```
# Frequency and Count Tables
freq(r3)
```

```
##      value count
## [1,]     1    12
## [2,]     2     3
## [3,]     3    41
## [4,]     4    46
## [5,]     5    48
```

```
freq(r3, value=4)
```

```
## [1] 46
```

```
# Cross-tabulate (i.e. compare) the contents of 2 RasterLayer objects
dim(r3)
```

```
## [1] 10 15  1
```

```
r5 = r3
r5[5,] = rep(3,ncol(r5))
#values(r3); values(r5)
freq(r3); freq(r5); freq(r3)-freq(r5)
```

```
##      value count
## [1,]     1    12
## [2,]     2     3
## [3,]     3    41
## [4,]     4    46
## [5,]     5    48
```

```
##      value count
## [1,]     1    11
## [2,]     2     3
## [3,]     3    50
## [4,]     4    40
## [5,]     5    46
```

```
##      value count
## [1,]     0     1
## [2,]     0     0
## [3,]     0    -9
## [4,]     0     6
## [5,]     0     2
```

```
crosstab(r3,r5)  #crosstab(r3,r5, useNA=FALSE) - No difference!?
```

```
##        layer.2
## layer.1  1  2  3  4  5
##       1 11  0  1  0  0
##       2  0  3  0  0  0
##       3  0  0 41  0  0
##       4  0  0  6 40  0
##       5  0  0  2  0 46
```

```
# Summarizing a Portion of a Raster Layer
# --------------------------------------

# Definine and Summarise particular areas: Zonal statistics
r4.zones = r4
r4.zones[] = rep(1:5, each=30)
r4.zones[]
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [106] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [141] 5 5 5 5 5 5 5 5 5 5
```

```
zonal(r4, r4.zones, mean) # Using a function: For small files
```

```
##      zone     value
## [1,]    1  8.443729
## [2,]    2 22.572451
## [3,]    3 31.077699
## [4,]    4 62.313849
## [5,]    5 72.074210
```

```
zonal(r4, r4.zones, 'mean') # Using a 'character value': For big files is better this way
```

```
##      zone      mean
## [1,]    1  8.443729
## [2,]    2 22.572451
## [3,]    3 31.077699
## [4,]    4 62.313849
## [5,]    5 72.074210
```

```
# Summarizing values in the Cell Neighbourhood using a "moving window" (focal function)
# -------------------------------------------------------------------------------------
r5 = raster(ncols=100,nrows=100, xmn=0)
r5[] = runif(ncell(r5))

# 3x3 mean filter
r5.fw3x3 = focal(r5, w=matrix(1/9,nrow=3,ncol=3))
r5.fw3x3
```

```
## class       : RasterLayer
## dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
## resolution  : 1.8, 1.8  (x, y)
## extent      : 0, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0.1483377, 0.8508603  (min, max)
```

```
head(values(r5.fw3x3))
```

```
## [1] NA NA NA NA NA NA
```

```
# 5x5 mean filter
r5.fw5x5 = focal(r5, w=matrix(1/9,nrow=3,ncol=3))
r5.fw5x5
```

```
## class       : RasterLayer
## dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
## resolution  : 1.8, 1.8  (x, y)
## extent      : 0, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0.1483377, 0.8508603  (min, max)
```

```
head(values(r5.fw5x5))
```

```
## [1] NA NA NA NA NA NA
```

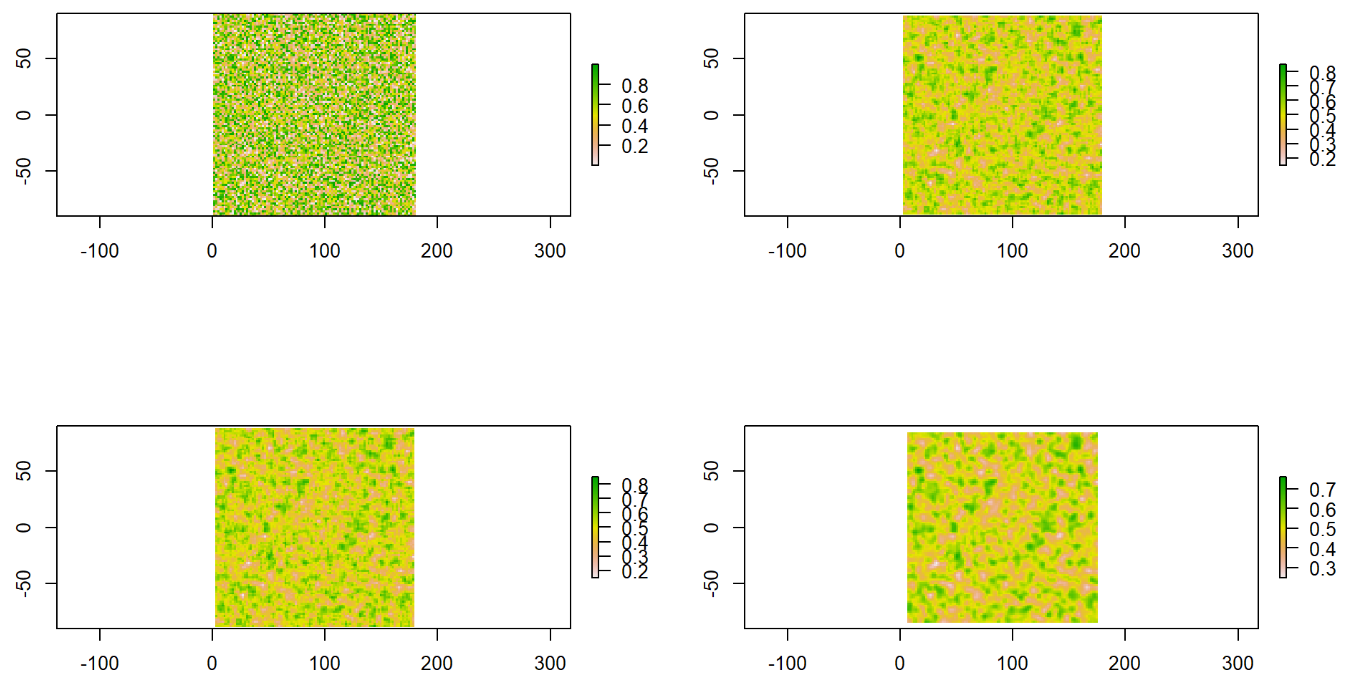
```
# Gaussian filter
wGaus = focalWeight(r5, 2, "Gauss")
r5.fwGaus = focal(r5, w=wGaus)
r5.fwGaus
```

```
## class       : RasterLayer
## dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
## resolution  : 1.8, 1.8  (x, y)
## extent      : 0, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0.2465152, 0.7618615  (min, max)
```

```
head(values(r5.fwGaus))
```

```
## [1] NA NA NA NA NA NA
```

```
# Compare rasters visually
# Dimensions & Resolution mantained (!= to 'aggregate', see later)
# With mean (i.e. weighted sum really) filter values are in the same range than in the origin
al
# raster; but in with the Gaussian filter values can be smaller and larger than in the origin
al layer.
par(mfrow=c(2,2))
plot(r5)
plot(r5.fw3x3)
plot(r5.fw5x5)
plot(r5.fwGaus)
```



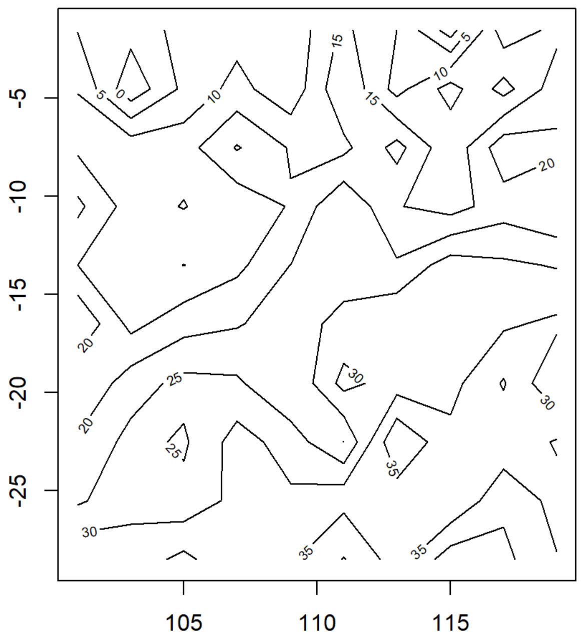
8.2 Visual Summaries

The `raster` package provides many functions to visualy display and summarise the data in `Raster*` objects. The `rasterVis` package complements the `raster` package, providing additional methods for enhanced visualization and interaction.

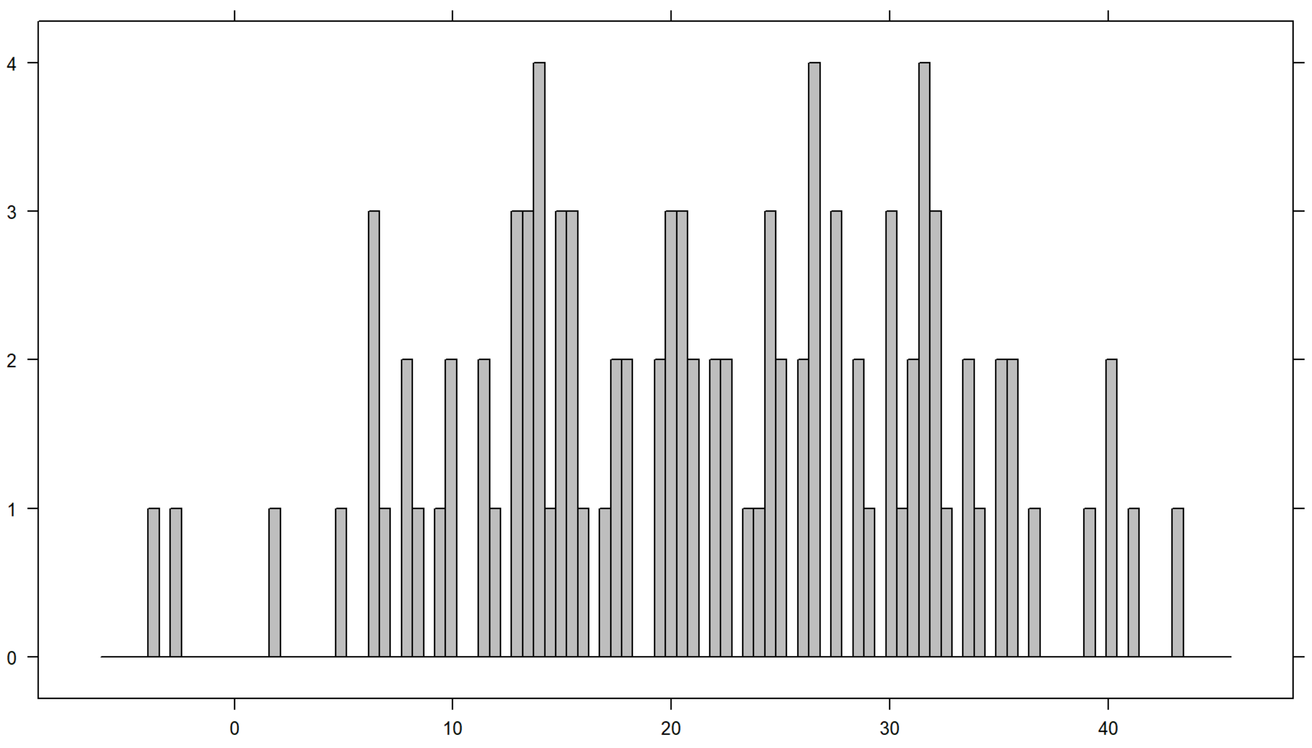
Some of the functions in this packages can only be used in single-layer rasters, others can only be used in muti-layered rasters, and the remaining functions can be used for both single-layer and multi-layer rasters. In particular, the visualisation functions that we explore below can be applied the the following raster data types:

- Single-layer raster objects:
    - In `raster` packge: `contour`
- Single-layer & multi-layer raster objects:
    - In `raster` packge: `histogram` , `densityplot` , `persp`
    - In `rasterVis` packge: `contourplot`
- Multi-layer raster objects:
    - In `raster` packge: `pairs` , `plotRGB`
    - In `rasterVis` packge: `bwplot`

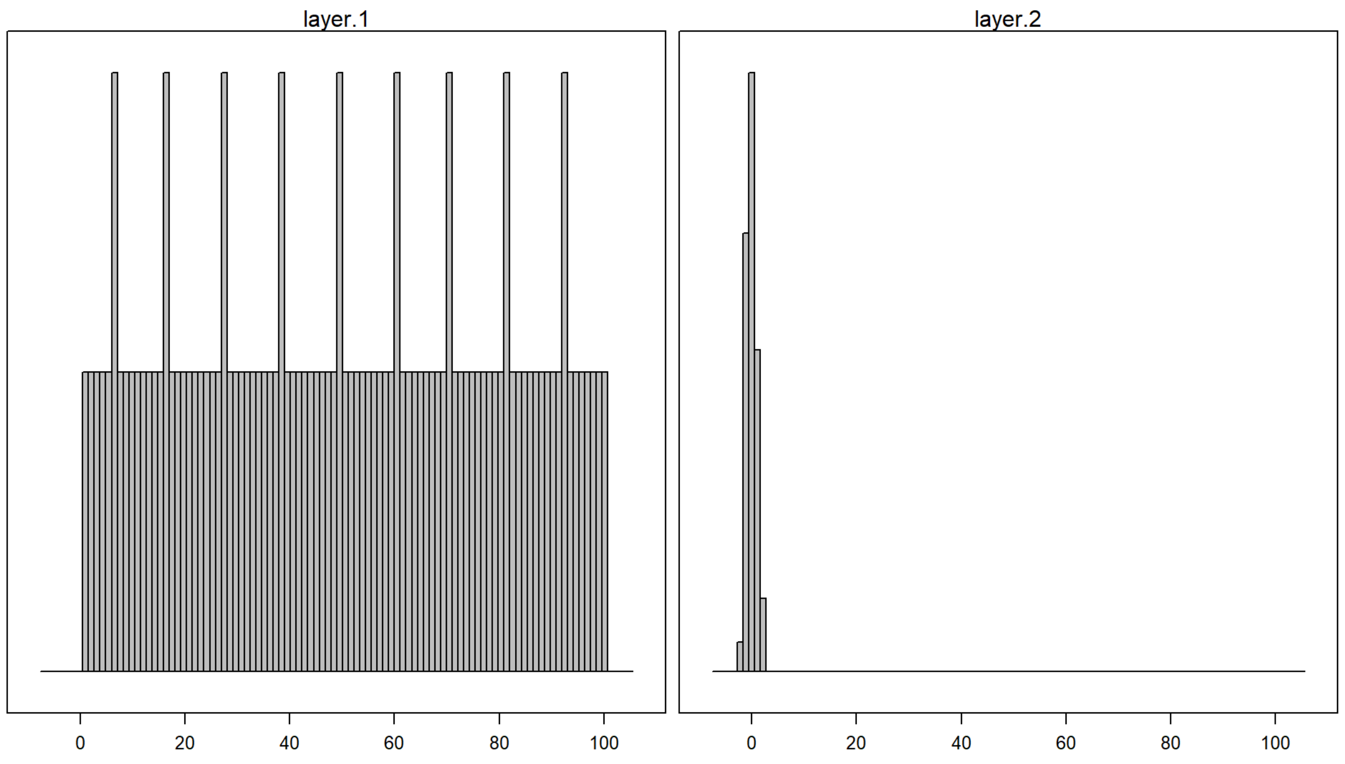
```
# In package 'raster'
# =================
# In addition of 'plot()' to draw the values of a raster (see above)


# For a Single Layer objects exclusively
# --------------------------------------
# Contour: Contour plot of a Raster Layer
par(mfrow=c(1,2))
contour(r2b)
#contour(s2)   # WRONG: It doesn't work! ('contourplot()' in 'rasterVis' pkg works for all Ra
ster* objects)


# For both a Single Layer and Multiple Layers objects
# ---------------------------------------------------

# Histogram: Draw histograms (with lattice) of Raster objects.
par(mfrow=c(1,2))
```



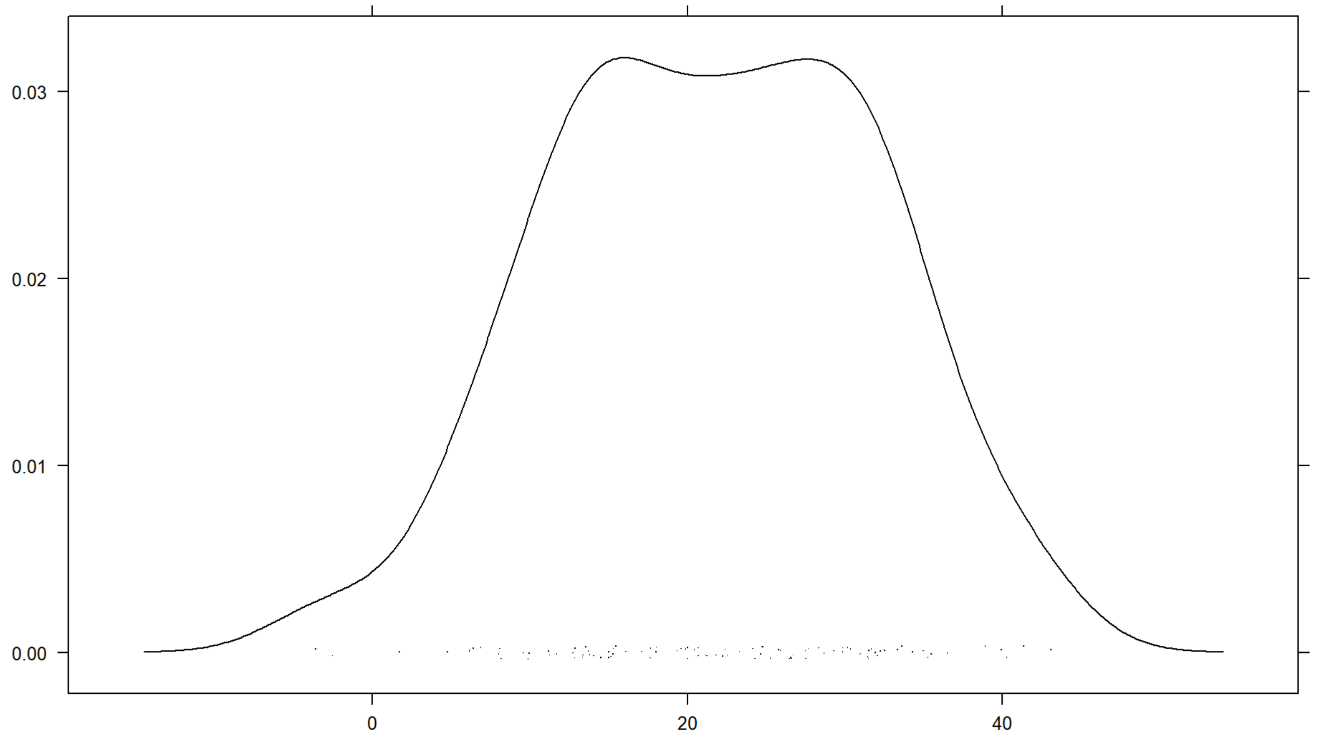
```
histogram(r2b)
```



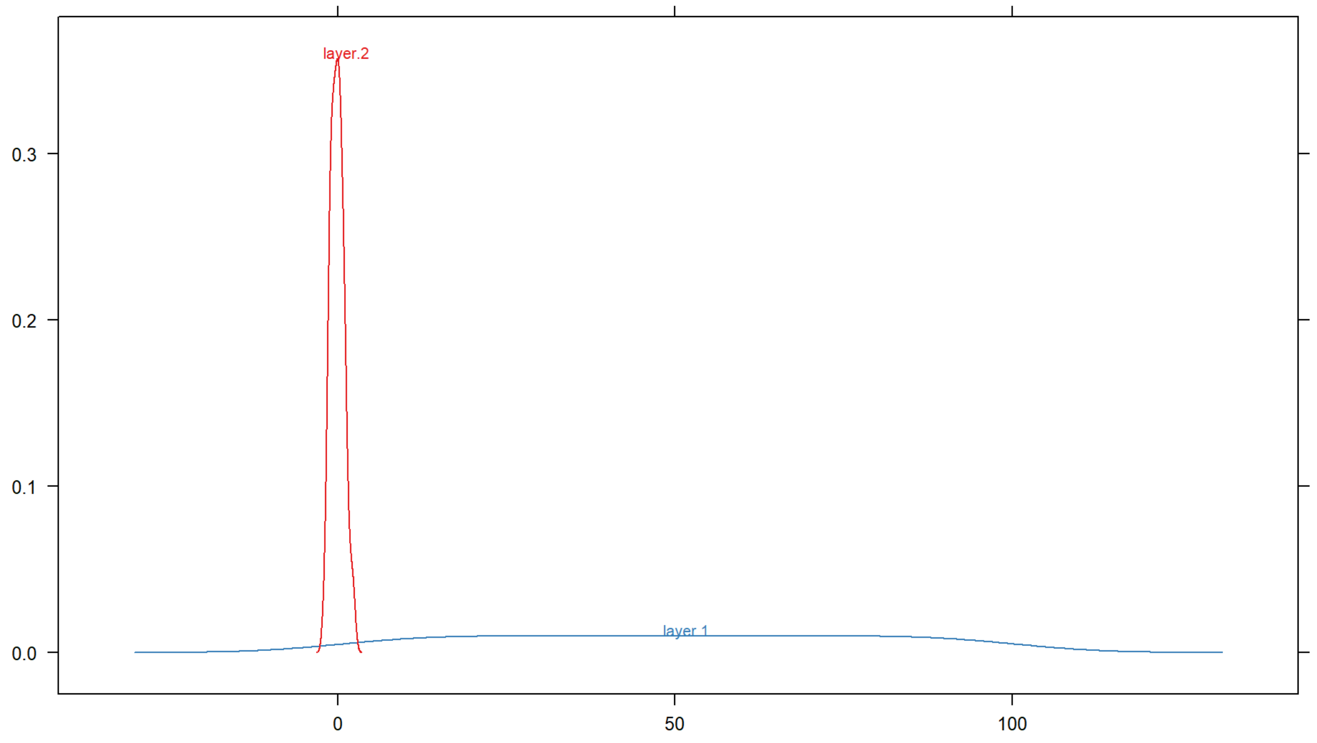
```
histogram(s2)
```



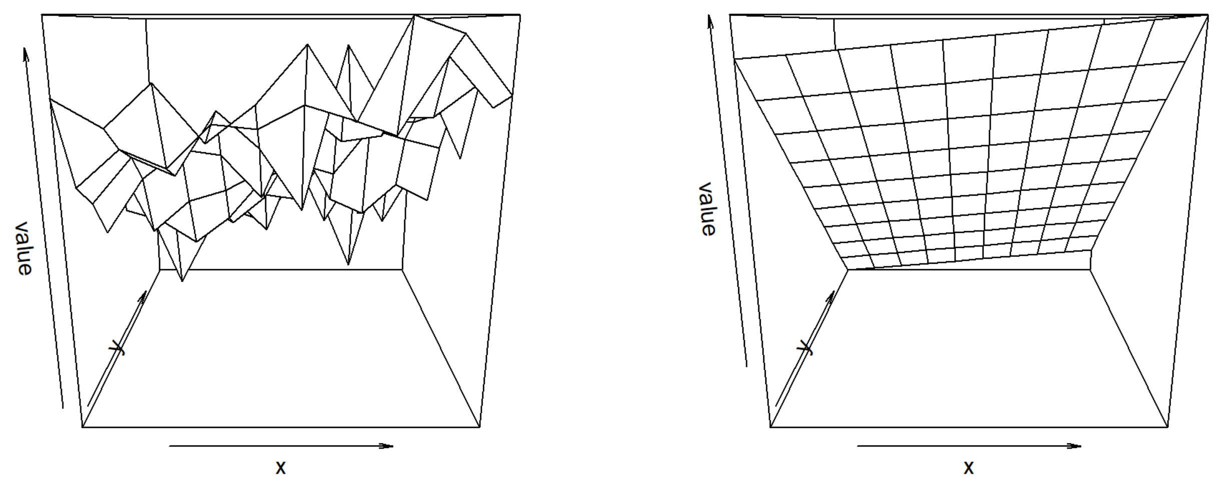
```
# Density plots: Draw kernel density plots (with lattice) of Raster objects.
# par(mfrow=c(1,2)) # 'par()': Doesn't work with 'densityplot()'
densityplot(r2b)
```



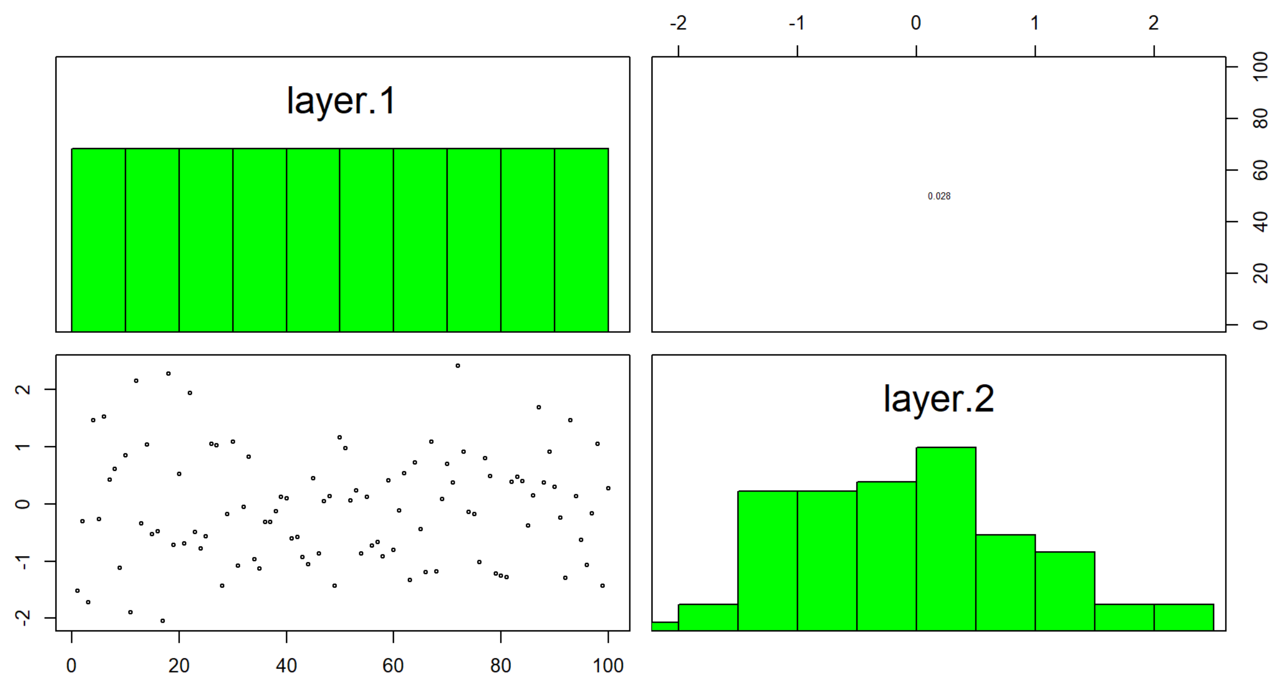
```
densityplot(s2)
```



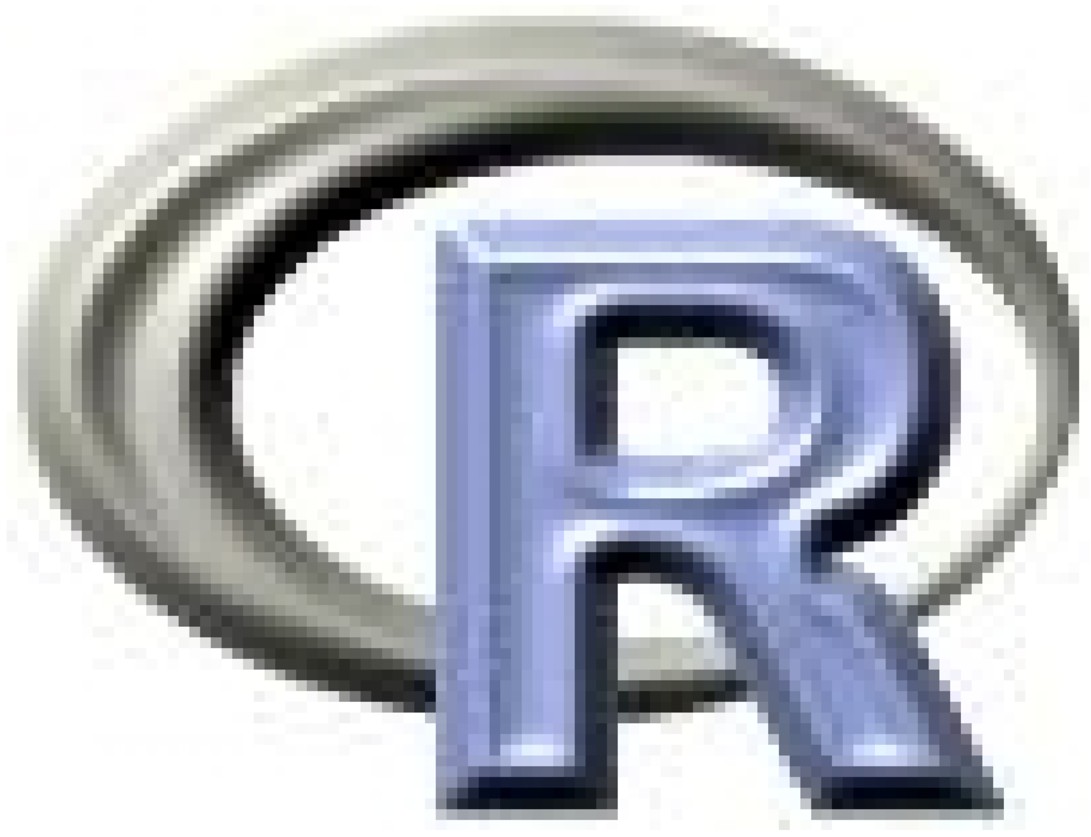
```
# Perspective plot. This is an implementation of a generic function in the graphics package.
par(mfrow=c(1,2))
persp(r2b)
persp(s2)
```



```
# For a Multiple Layers (RasterStack and RasterBrick objects) objects exclusively
# ---------------------------------------------------------------------------
# Pairs: Pair plots of layers in a RasterStack or RasterBrick.
# This is a wrapper around graphics function pairs.
par(mfrow=c(1,1))
pairs(s2)
```

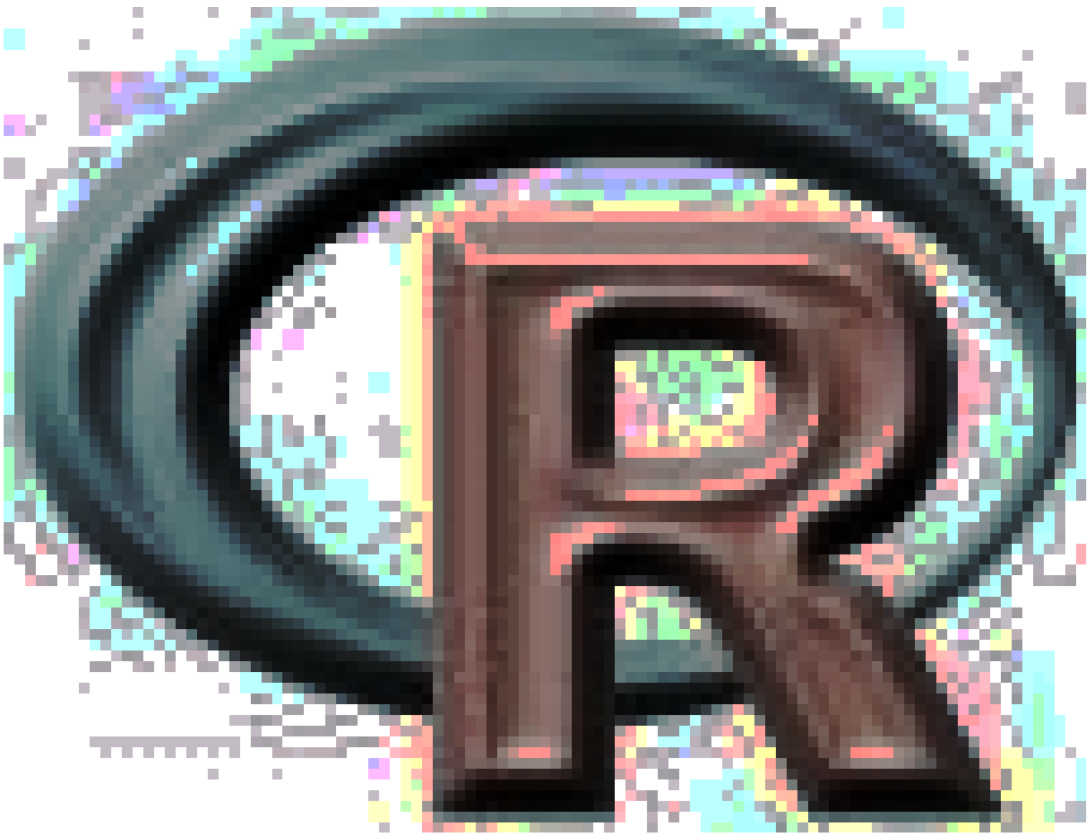


```
# plotRGB: Make a Red-Green-Blue plot based on 3 layers. (Description and Example from 'plotR
GB()' Help).
# These 3 layers are sometimes referred to as "bands" because they may represent different ba
ndwidths in the
# electromagnetic spectrum.
# The 3 layers are combined such that they represent the red, green and blue channel.
# This function can be used to make 'true (or false) color images' from Landsat and other mul
ti-band satellite images.
b = brick(system.file("external/rlogo.grd", package="raster"))
plotRGB(b)
```

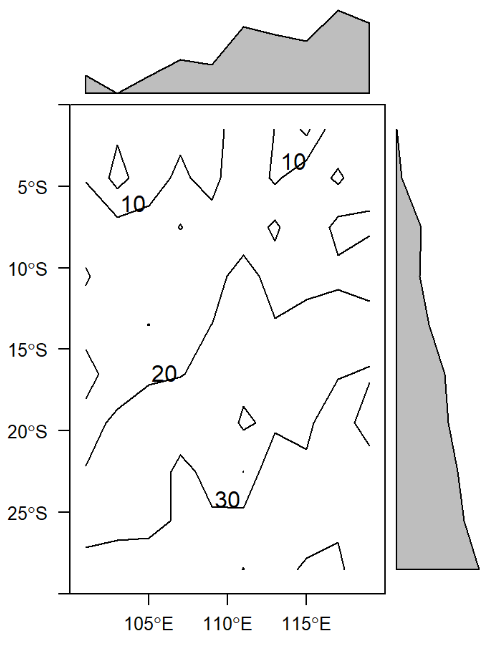


```
plotRGB(b, 3, 2, 1)
```

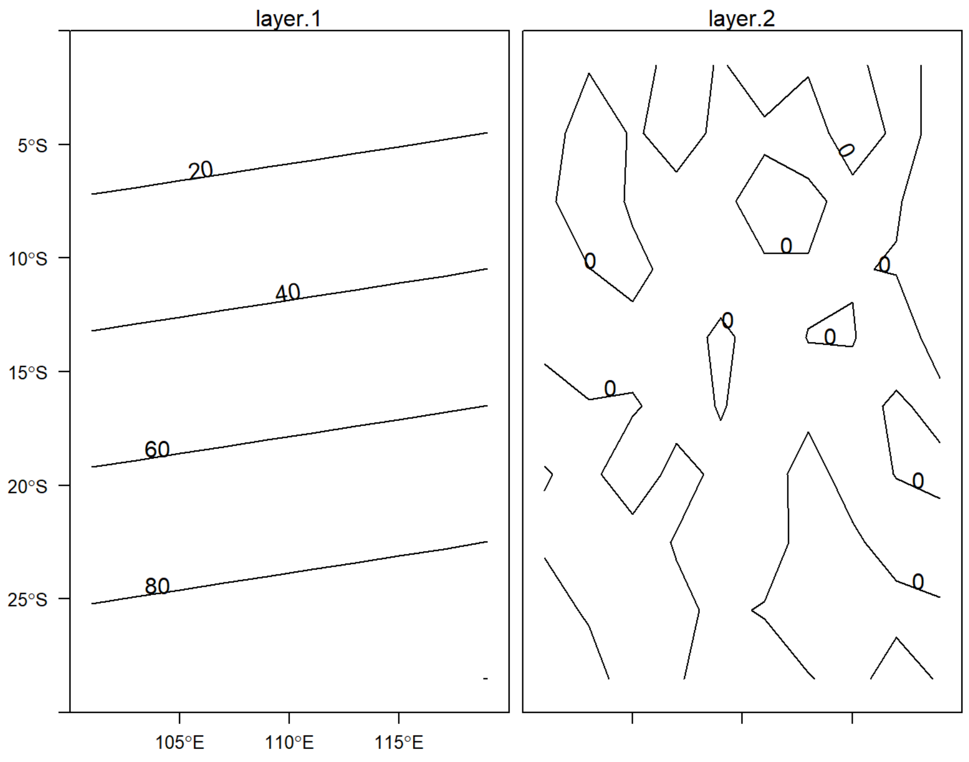
```
plotRGB(b, 3, 2, 1, stretch='hist')
```



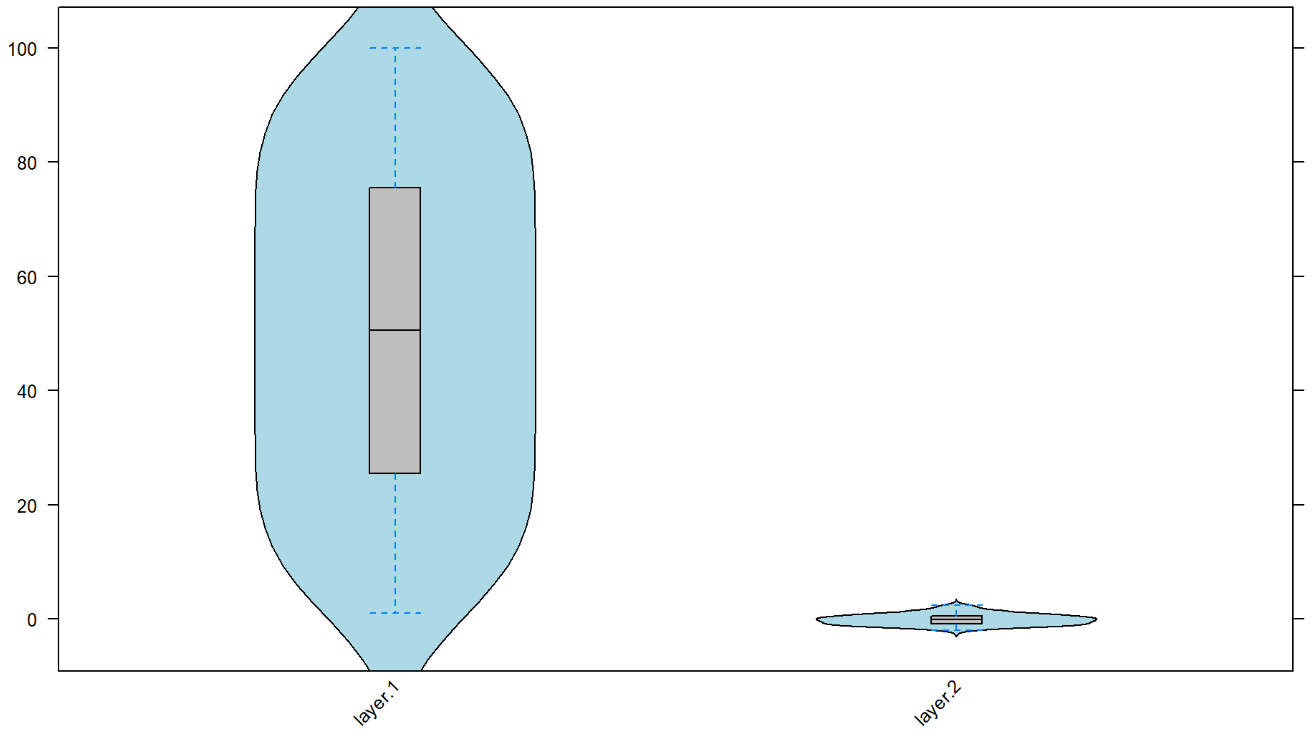
```
rm(b)



# In package 'rasterVis'
# ======================
# In addition of 'levelplot()' to draw the values of a raster (see above)



# For both a Single Layer and Multiple Layers objects
# ---------------------------------------------------
# Contour Plots of Raster objects with lattice methods and marginal plots with grid objects.
contourplot(r2b)
```



```
contourplot(s2)
```



```
# For Multiple Layers (RasterStack and RasterBrick objects) object exclusively
# ---------------------------------------------------------------------------
# Bwplots: Plots of RasterStackBrick objects using to compose the graphic a combination of:
#  panel.violin and panel.bwplot.
bwplot(s2)
```



9 Working with rasters values

9.1 Modifying individual cell values

```
# Directly (already used above to assing values to raster r2b)
r1c = r1b
r1c[5,1:5] = 6:10  # (In row 5, columns 1 to 5 = 6 to 10)
values(r1c)
```

```
##   [1]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [24]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [47]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  6  7  8  9 10  2  2  2  2
##  [70]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [93]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [116]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [139]  2  2  2  2  2  2  2  2  2  2  2  2
```

```
r1c[5,6]=NA   # Missing data
values(r1c)
```

```
##   [1]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [24]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [47]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  6  7  8  9 10 NA  2  2  2
##  [70]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [93]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [116]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [139]  2  2  2  2  2  2  2  2  2  2  2  2
```

```
r1c[5,6]=-9999    # '-9999' is the Missing Data Flag in many GIS standards (e.g. NEON)
values(r1c)
```

```
##   [1]     2     2     2     2     2     2     2     2     2     2     2
##  [12]     2     2     2     2     2     2     2     2     2     2     2
##  [23]     2     2     2     2     2     2     2     2     2     2     2
##  [34]     2     2     2     2     2     2     2     2     2     2     2
##  [45]     2     2     2     2     2     2     2     2     2     2     2
##  [56]     2     2     2     2     2     6     7     8     9    10 -9999
##  [67]     2     2     2     2     2     2     2     2     2     2     2
##  [78]     2     2     2     2     2     2     2     2     2     2     2
##  [89]     2     2     2     2     2     2     2     2     2     2     2
## [100]     2     2     2     2     2     2     2     2     2     2     2
## [111]     2     2     2     2     2     2     2     2     2     2     2
## [122]     2     2     2     2     2     2     2     2     2     2     2
## [133]     2     2     2     2     2     2     2     2     2     2     2
## [144]     2     2     2     2     2     2     2
```

```
# Combined with replacement functions
values(r1c)   # Missing data flag: -9999
```

```
##   [1]     2     2     2     2     2     2     2     2     2     2     2
##  [12]     2     2     2     2     2     2     2     2     2     2     2
##  [23]     2     2     2     2     2     2     2     2     2     2     2
##  [34]     2     2     2     2     2     2     2     2     2     2     2
##  [45]     2     2     2     2     2     2     2     2     2     2     2
##  [56]     2     2     2     2     2     6     7     8     9    10 -9999
##  [67]     2     2     2     2     2     2     2     2     2     2     2
##  [78]     2     2     2     2     2     2     2     2     2     2     2
##  [89]     2     2     2     2     2     2     2     2     2     2     2
## [100]     2     2     2     2     2     2     2     2     2     2     2
## [111]     2     2     2     2     2     2     2     2     2     2     2
## [122]     2     2     2     2     2     2     2     2     2     2     2
## [133]     2     2     2     2     2     2     2     2     2     2     2
## [144]     2     2     2     2     2     2     2
```

```
r1c[r1c==-9999] = NA  # Missing data flag changed back to: NA
values(r1c)
```

```
##   [1]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [24]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [47]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  6  7  8  9 10 NA  2  2  2
##  [70]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
##  [93]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [116]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
## [139]  2  2  2  2  2  2  2  2  2  2  2  2
```

```
r1c = (r1c==2)
values(r1c)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE    NA
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [100]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [122]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [144]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
r1c[!r1c] = 0
r1c[r1c] = 1
values(r1c)
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [24]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [47]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0 NA  1  1  1
##  [70]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [93]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## [116]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## [139]  1  1  1  1  1  1  1  1  1  1  1  1
```

9.2 Raster Calculations

9.2.1 'Raster Algebra' (also known as 'Raster Maths')

Used for small rasters and simple calculations

Directly combines rasters using:

- Operators (for more information on R Operators see this link (https://www.datamentor.io/r-programming/operator/)):
  - Arithmetic operators (+,'-,*,/,^,%%,%/%)
  - Relational operators (>, <, >=, <=, ==, !=)
  - Logical operators (!,&,&&,|,||)
- Functions (algebraic (but 'abs'), summary, and logical functions need > 1 layer):
  - Algebraic functions (e.g. abs, sum, prod)
  - Rounding functions (round, ceiling, floor, trunc)
  - Re-scaling functions (e.g. log, log10, sqrt, exp)
  - Trigonometric functions (e.g. sin, cos, tan, atan)
  - Summary functions (e.g. min, max, range)
  - Logical functions(any, all)

```
# Using 1 layer
# --------------
values(r1a)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
```

```
values((r1a * 3)^2)
```

```
##   [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
##  [36] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
##  [71] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
## [106] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
## [141] 9 9 9 9 9 9 9 9 9 9
```

```
values(r1a == 1)
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [113] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [127] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [141] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
values(!(r1a == 1))
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
values(sqrt(r1b))
```

```
##   [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##   [8] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [15] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [22] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [29] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [36] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [43] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [50] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [57] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [64] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [71] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [78] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [85] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [92] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
##  [99] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [106] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [113] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [120] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [127] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [134] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [141] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [148] 1.414214 1.414214 1.414214
```

```
values(round(sqrt(r1b)))
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
```

```
values(round(sqrt(r1b))==1)
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [113] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [127] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [141] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# Using multiple layers
# -----------------------

# With Operatros
values(r1a)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
```

```
values(r1b)
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```
r1c=r1a + r1b
values(r1c)
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
# With Functions
  # Algebraic functions
values(sum(r1a,r1b))
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
values(prod(r1a,r1b))
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```
  # Summary functions
values(max(r1a,r1b))
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```
values(min(r1a,r1b))
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
```

```
values(mean(r1a,r1b))
```

```
##   [1] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [18] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [35] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [52] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [69] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [86] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
## [103] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
## [120] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
## [137] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
```

```
  # Logical Functions
values(r1c)
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
r1c = (r1c == 1)
values(r1c)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
values(any(r1c, !r1c))
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [113] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [127] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [141] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
values(all(r1c, !r1c))
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

9.2.2 Raster Calculations using Higher Level Functions

Directly adding, subtracting, etc. layers of big raster objects is not recommended.

For big raster objects High Level Functions should be used instead.

9.2.2.1 For a Single Raster Objects

`calc` : Uses a single Raster* object and a provided formula to calculate values for a new Raster* object.

The function type will depend of the input object type: * For a uni-layer (RasterLayer) input, the function is typically a function that can take a single vector as input and return a vector of values of the same length (e.g. sqrt). * For a multi-layer (RasterStack or RasterBrick) input, the function should operate on a vector of values, with one vector for each cell.

The object type returned depends on the input object type and function type: * When input is a RasterLayer, it returns a RasterLayer. * When input is a RasterStack or RasterBrick and the Function a Summary type Function (i.e. returns a single value), it returns a RasterLayer. * When input is a RasterStack or RasterBrick and the Function returns more than one number (e.g. fun=quantile), it returns a RasterBrick.

In many cases, what can be achieved with ‘calc’, can also be accomplished with a more intuitive ‘raster-algebra’ notation. For example, the log10 of the cells in a raster can be computed as log10(r), which is simpler than using calc with this function. However, `calc` should be faster when using large datasets and/or complex formulas.

The function can even be a statistical function, such as `lm` . See `cacl` help for details.

```
# To calculate the log10 of the values in a raster can do:
 # Method 1: Directly providing the function
r1b.log10.m1 = calc(r1b, fun=log10)
r1b.log10.m1
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0.30103, 0.30103  (min, max)
```

```
 # Method 2: Writing a function and passing it on.
r1b.log10.m2 = function(r) { log10(r) }
r1b.log10.m2 = calc(r1b, fun=r1b.log10.m2)
r1b.log10.m2
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0.30103, 0.30103  (min, max)
```

```
# Replace values too high (>100) and too low (< 0) with NAs
r5 = raster(ncols=10, nrows=10)
r5[] = runif(ncell(r5), min=-10, max=110)
summary(r5)
```

```
##              layer
## Min.     -9.982466
## 1st Qu.  27.433714
## Median   57.362875
## 3rd Qu.  81.637164
## Max.    109.883174
## NA's      0.000000
```

```
replacement.f1 = function(r) { r[r<0|r>100] = NA; return(r) }
r5.NAs = calc(r5, fun=replacement.f1)
summary(r5.NAs)
```

```
##              layer
## Min.     0.6755192
## 1st Qu. 34.6801489
## Median  57.3238800
## 3rd Qu. 77.3869539
## Max.    99.4091017
## NA's    21.0000000
```

```
# Replace 'NA' with '-9999', the code for missing values in some package
 replacement.f2 = function(r) { r[is.na(r)] = -9999; return(r) }
 r5.neg9999 = calc(r5.NAs, fun=replacement.f2)
 summary(r5.neg9999)
```

```
##                layer
## Min.    -9999.00000
## 1st Qu.    12.50605
## Median     49.99301
## 3rd Qu.    71.73087
## Max.       99.40910
## NA's        0.00000
```

`stackApply` : Computes summary type layers for subsets of a RasterStack or RasterBrick. To do so it uses a summary function applied on subsets of layers of a RasterStack or RasterBrick object.These subsets are defined by vector indices. If the number of indeces in the vector is smaller than the number of layers the indices are recycled. The summary function must return a single value; thus, the number of layers in the output Raster* equals then number of unique values in the indices vector. If the output raster contains a single layer it would be a RasterLayer object. If it contains multiple layers it would be a RasterBrick object.

```
# it would be a RasterBrick object.
s = stack(r1a,r1a,r1a,r1b,r1b,r1b)

s.sA1 = stackApply(s, indices=rep(1,nlayers(s)), fun=mean)
s.sA1
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : index_1
## values      : 1.5, 1.5  (min, max)
```

```
# Providing indices for all layers. Output is a RasterBrick with 2 layers: Mean of layers 1-3
and mean of layers 4-6.
s.sA2 = stackApply(s, indices=c(1,1,1,2,2,2), fun=mean)
s.sA2
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : index_1, index_2
## min values  :       1,       2
## max values  :       1,       2
```

```
# Recycled indices. Output is a RasterBrick with 2 layers: Mean of layers 1, 3,and 5 and mean
of layers 2, 4, and 6.
s.sA3 = stackApply(s, indices=c(1,2), fun=mean)
s.sA3
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       :  index_1,  index_2
## min values  : 1.333333, 1.666667
## max values  : 1.333333, 1.666667
```

```
rm(s)
```

`overlay` : It is typically used with multiple Raster* objects, but it can also be used with a single `Raster*` object. However, it is often simpler to use `calc` (or "Raster Algebra") for single Raster* objects. See next section for its application to multi-layered raster objects.

9.2.2.2 For a Multiple Raster Objects

`ovelay` : It is more efficient that 'Raster Maths' when rasters are large and/or calculations complex and/or rasters stored in separate individual files.

NOTE: Do not confuse with the deprecated 'overlay' function in the 'sp' package. They are different.

Syntax: output.ras = overlay(input1.ras, input2.ras, fun=FunctionCalcName)

The raster objects used as inputs can be single-layered or multi-layered rasters (with the same number of layers). Both case produce similar results. In the multi-layered rasters case the function is applied to each pair of layers.

```
# Input single-layered objects (RasterLayer)
# .........................................
values(r1a)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
```

```
values(r1b)
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```
# Adding Rasters
 # Using Raster Algebra with Operators
r1c=r1a + r1b
values(r1c)
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
 # Using 'overlay()'
r1d = overlay( r1a, r1b, fun=function(r1,r2) {return(r1+r2)} )
values(r1d)
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
values(r1c==r1d)  # Compare individual cells, are they equal?
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [113] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [127] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [141] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
all(values(r1c==r1d))  # Are all cells equal (in 1 answer)?
```

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

```
# Using a function
silly.f = function(x,y) {
                silly.ras = x* round((y)^(2/3))
                return(silly.ras)
                }
r1e = overlay(r1a, r1b, fun=silly.f)
values(r1e)
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2
```

```
# Input multi-layered objects (RasterStack/RasterBrick)
# ....................................................
s1a = stack(r1a,r1a)
s1b = stack(r1b,r1b)
s1ab = stack(r1a,r1b)
s1a; s1b
```

```
## class       : RasterStack 
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## names       : layer.1, layer.2 
## min values  :       1,       1 
## max values  :       1,       1
```

```
## class       : RasterStack 
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## names       : layer.1, layer.2 
## min values  :       2,       2 
## max values  :       2,       2
```

```
# Adding rasters
s1d1 = overlay( s1a, s1b, fun=function(r1,r2) {return(r1+r2)} )
s1d2 = overlay( s1ab, s1ab, fun=function(r1,r2) {return(r1+r2)} )
r1d; s1d1; s1d2
```

```
## class       : RasterLayer 
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer 
## values      : 3, 3  (min, max)
```

```
## class       : RasterBrick 
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer.1, layer.2 
## min values  :       3,       3 
## max values  :       3,       3
```

```
## class       : RasterBrick 
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer.1, layer.2 
## min values  :       2,       4 
## max values  :       2,       4
```

```
# Using a function
s1e1 = overlay(s1a, s1b, fun=silly.f)
s1e2 = overlay(s1ab, s1ab, fun=silly.f)
r1e; s1e1; s1e2
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 2, 2  (min, max)
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       2,       2
## max values  :       2,       2
```

```
## class       : RasterBrick
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer.1, layer.2
## min values  :       1,       4
## max values  :       1,       4
```

```
rm(r1d, r1e, s1d1, s1d2, s1e1, s1e2)
```

NOTE: Rasters in operations including multiple rasters must have the same: Dimensions, Projection, Extent, and Resolution. E.g.: Create a new raster with the same Dimensions and Projection than r1a and r1b, but different Extent and Resolution

```
r3 = raster(nrow=10, ncol=15, xmn=100, xmx=130, ymn=-30, ymx=0)
values(r3) = 5;
r1a; r1b; r3
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 2, 2  (min, max)
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 130, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 5, 5  (min, max)
```

```
values(r1a+r1b)   # Works
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
```

```
#values(r1a+r3)     # Does Not Work
rm(r3)
```

9.3 Raster Classification

Aim: To reclassify values in the raster; for example from 6 different land-types, ecosystems to 3

```
r3 = raster(nrow=10, ncol=10)
values(r3) = sample(x=1:6, size=ncell(r3), replace=TRUE)
values(r3)
```

```
##   [1] 1 2 2 1 3 6 5 4 6 6 3 1 1 6 4 3 2 4 1 6 3 4 4 1 4 2 2 5 2 4 4 1 5 6 4
##  [36] 5 2 6 1 5 1 2 3 6 6 5 3 4 6 4 4 5 3 1 1 5 1 3 3 3 3 3 2 5 6 6 6 2 5 4
##  [71] 1 5 2 3 6 5 1 4 4 5 4 1 3 5 4 2 5 2 6 1 3 5 1 2 1 5 2 3 4 2
```

```
old.vals = sort(unique(values(r3)))
old.vals
```

```
## [1] 1 2 3 4 5 6
```

```
reclass.m = matrix(c(old.vals, c(1,rep(2,3),3,3)), ncol=2)
reclass.m
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    2
## [4,]    4    2
## [5,]    5    3
## [6,]    6    3
```

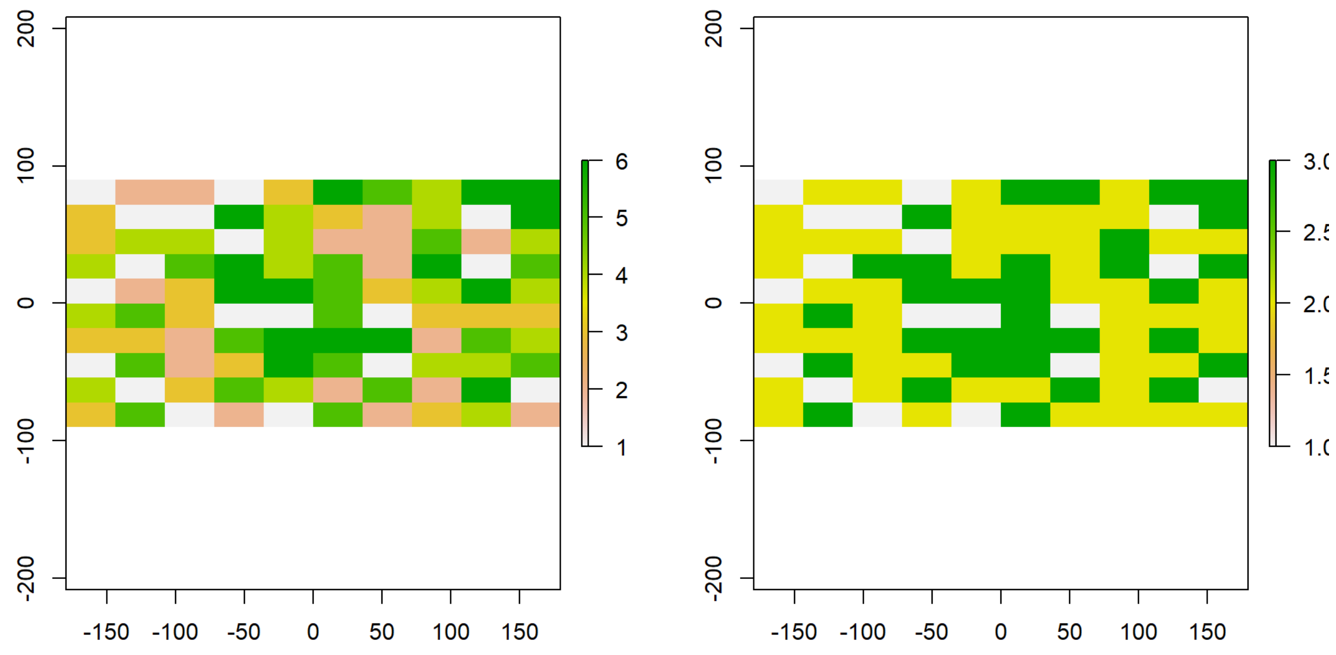
```
r3.reclass = reclassify(r3, rcl=reclass.m)
values(r3)
```

```
##   [1] 1 2 2 1 3 6 5 4 6 6 3 1 1 6 4 3 2 4 1 6 3 4 4 1 4 2 2 5 2 4 4 1 5 6 4
##  [36] 5 2 6 1 5 1 2 3 6 6 5 3 4 6 4 4 5 3 1 1 5 1 3 3 3 3 3 2 5 6 6 6 2 5 4
##  [71] 1 5 2 3 6 5 1 4 4 5 4 1 3 5 4 2 5 2 6 1 3 5 1 2 1 5 2 3 4 2
```

```
values(r3.reclass)
```

```
##   [1] 1 2 2 1 2 3 3 2 3 3 2 1 1 3 2 2 2 2 1 3 2 2 2 1 2 2 2 3 2 2 2 1 3 3 2
##  [36] 3 2 3 1 3 1 2 2 3 3 3 2 2 3 2 2 3 2 1 1 3 1 2 2 2 2 2 2 3 3 3 3 2 3 2
##  [71] 1 3 2 2 3 3 1 2 2 3 2 1 2 3 2 2 3 2 3 1 2 3 1 2 1 3 2 2 2 2
```

```
par(mfrow=c(1,2))
plot(r3)
plot(r3.reclass)
```



10 Other ways to Combine multiple Raster objects: ‘mask()’ and ‘cover()’

10.1 'mask()'

`mask()` : It creates a new Raster* object that has the same values as x, except for the cells that are NA (or other maskvalue) in a 'mask'. These cells become NA (or other updatevalue). The mask can be: A Raster* object of the same extent and resolution, or a Spatial* object (e.g. SpatialPolygons) in which case all cells that are not covered by the Spatial object are set to updatevalue. You can use `inverse=TRUE` to set the cells that are not NA (or other maskvalue) in the mask, or not covered by the Spatial* object, to NA (or other updatevalue).

```
# Exlore Original Raster
r2b; r2b[]
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
##   [1]  4.780121  1.736944  6.889259  8.038875  6.171172 16.102196  9.951327
##   [8] -2.523808 18.019132 13.707639  9.590426 -3.583128  6.199381 11.708418
##  [15]  6.413912 19.344206  8.185034 17.106459  8.098741 17.678986 14.052422
##  [22] 13.361022 12.853425 20.442539 14.500877 13.704217 21.826718 11.201254
##  [29] 23.300787 21.121238 21.235934 12.744511 15.280282 11.247760 15.461039
##  [36] 24.732835 15.306017 12.881048 17.647537 15.022321 15.017614 13.350011
##  [43]  9.901972 13.815558 20.022739 22.240784 20.692412 27.484770 25.901197
##  [50] 24.642700 24.789695 13.547285 18.037063 19.605525 22.402358 26.727755
##  [57] 29.865164 29.867752 29.287679 31.001823 15.145634 22.541422 26.552205
##  [64] 25.791529 20.017065 31.688927 27.509136 28.339053 35.486548 25.257955
##  [71] 20.706820 26.596051 24.286767 32.242393 27.680065 19.901360 39.923136
##  [78] 31.443303 31.526053 35.502415 24.155372 28.685411 26.468013 31.504920
##  [85] 30.950129 33.623521 32.070314 30.168423 38.902838 33.343174 34.976939
##  [92] 31.924456 36.501460 30.356163 34.299321 40.280377 32.520743 43.064039
##  [99] 41.332182 35.267483
```

```
# Create a Raster Mask
r2b.Mask = r2b
r2b.Mask[] = 1
r2b.Mask[5:6,] = NA
r2b.Mask[,c(1,9)] = NA
r2b.Mask; r2b.Mask[]
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
##   [1] NA  1  1  1  1  1  1  1 NA  1 NA  1  1  1  1  1  1  1 NA  1 NA  1  1
##  [24]  1  1  1  1  1 NA  1 NA  1  1  1  1  1  1  1 NA  1 NA NA NA NA NA NA
##  [47] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  1  1  1  1  1  1  1 NA
##  [70]  1 NA  1  1  1  1  1  1  1 NA  1 NA  1  1  1  1  1  1  1 NA  1 NA  1
##  [93]  1  1  1  1  1  1 NA  1
```

```
# r2b masked out
r2b.MaskedOut = mask(r2b, r2b.Mask)
r2b.MaskedOut; r2b.MaskedOut[]
```

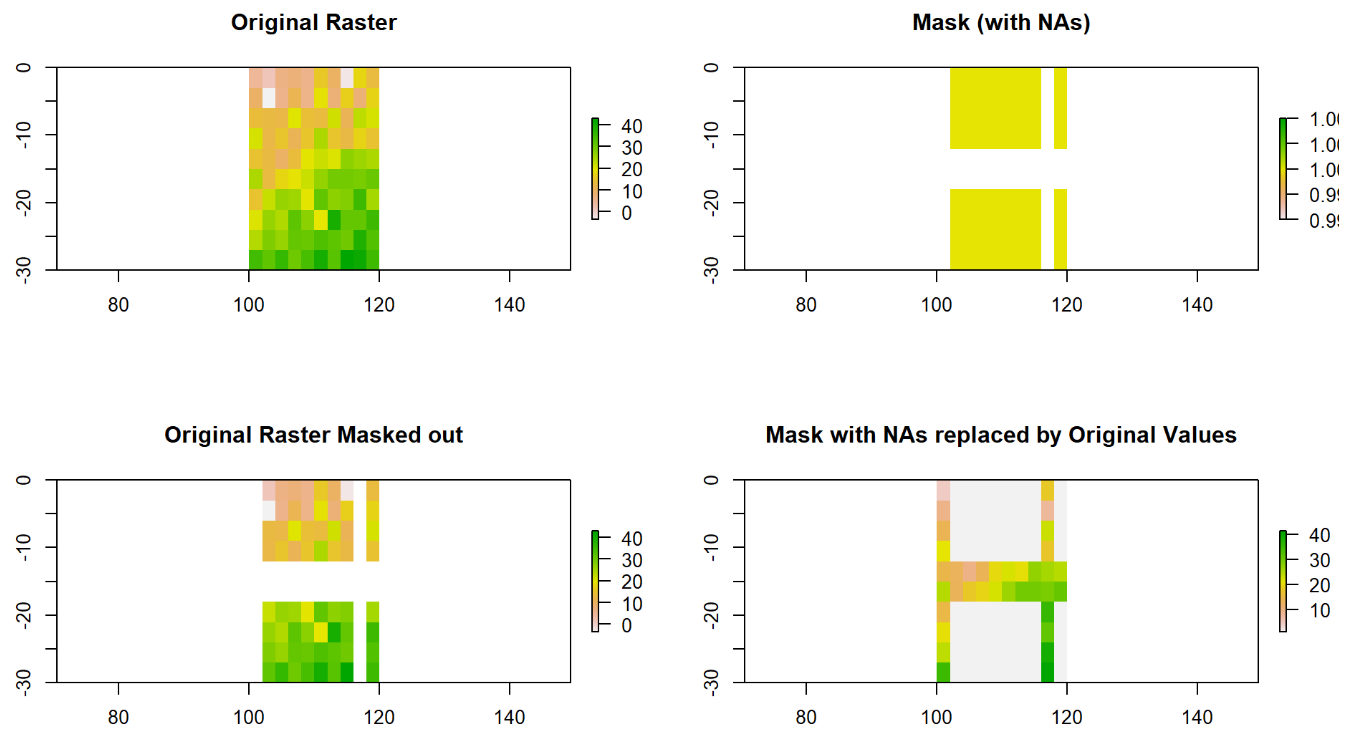
```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
##   [1]        NA  1.736944  6.889259  8.038875  6.171172 16.102196  9.951327
##   [8] -2.523808        NA 13.707639        NA -3.583128  6.199381 11.708418
##  [15]  6.413912 19.344206  8.185034 17.106459        NA 17.678986        NA
##  [22] 13.361022 12.853425 20.442539 14.500877 13.704217 21.826718 11.201254
##  [29]        NA 21.121238        NA 12.744511 15.280282 11.247760 15.461039
##  [36] 24.732835 15.306017 12.881048        NA 15.022321        NA        NA
##  [43]        NA        NA        NA        NA        NA        NA        NA
##  [50]        NA        NA        NA        NA        NA        NA        NA
##  [57]        NA        NA        NA        NA        NA 22.541422 26.552205
##  [64] 25.791529 20.017065 31.688927 27.509136 28.339053        NA 25.257955
##  [71]        NA 26.596051 24.286767 32.242393 27.680065 19.901360 39.923136
##  [78] 31.443303        NA 35.502415        NA 28.685411 26.468013 31.504920
##  [85] 30.950129 33.623521 32.070314 30.168423        NA 33.343174        NA
##  [92] 31.924456 36.501460 30.356163 34.299321 40.280377 32.520743 43.064039
##  [99]        NA 35.267483
```

10.2 ‘cover()’

`cover()` : Combines layers by taking the values of the first layer except where these are NA. For Raster* objects it replaces NA values in the first Raster object (x) with the values of the second (y), and so forth for additional Rasters. If x has multiple layers, the subsequent Raster objects should have the same number of layers, or have a single layer only (which will be recycled).

```
# Substitute the Missing Rows and Columns (i.e. containing NAs) in the Mask, with the Rows #
 and Columns in the Original Raster to create a Combined Raster.
r2b.coverMaskOrig = cover(r2b.Mask, r2b)



# Plot Original, Mask, Mask Out, and Combined Rasters.
par(mfrow=c(2,2))
plot(r2b, main='Original Raster')
plot(r2b.Mask, main='Mask (with NAs)')
plot(r2b.MaskedOut, main='Original Raster Masked out')
plot(r2b.coverMaskOrig, main='Mask with NAs replaced by Original Values')
```



11 Changing the essential Spatial Attributes of a Raster

11.1 Changing Spatial Extent

The Spatial Extent of Raster Objects can be modified using several funtions:

11.1.1 'crop()'

`crop()` : The most commonly used function to modify the extent of a raster object. It returns a geographic subset of an object as specified by an Extent object (or any object from which an extent object can be extracted/created). The Extent object represent our region of interest. When the object to crop is a Raster* object, the Extent is aligned to this object. Areas included in the Extent object, but outside the extent of the object to crop are ignored.

`drawExtent` : Allows to subset a raster object by clicking on two pionts of a plot (raster). Currently, it returns an error when used in Rmarkdown files (used to prepare this tutorial). However, you can give it a try by using the code below. Simply, remove the comment symbol (`#`) in front of the commands containing `drawExtent` or the raster object rb2.crop3.

```
# It can be used in multiple ways:
r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
# Providing the coordinates for the limits of the Extent object
r2b.crop1 = crop(r2b, extent(100,115,-40,-15))
r2b.crop1
```

```
## class       : RasterLayer
## dimensions  : 5, 8, 40  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 116, -30, -15  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 13.54729, 43.06404  (min, max)
```

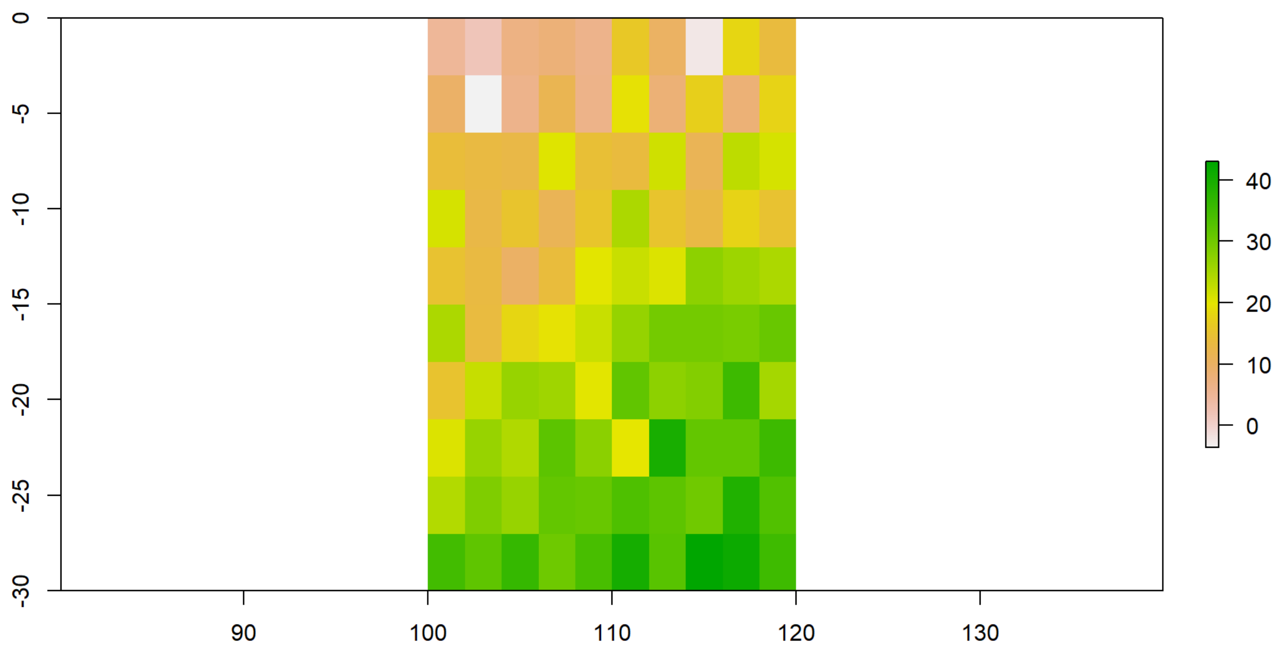
```
# Obtaining the limits of the Extent object from another Spatial Object
# (a Raster, SpatiaPolygon*, SpatialLines*, SpatialPoints* object)
r2c  = raster(nrow=10, ncol=10, xmn=110, xmx=125, ymn=-20, ymx=5)
r2c
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 1.5, 2.5  (x, y)
## extent      : 110, 125, -20, 5  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

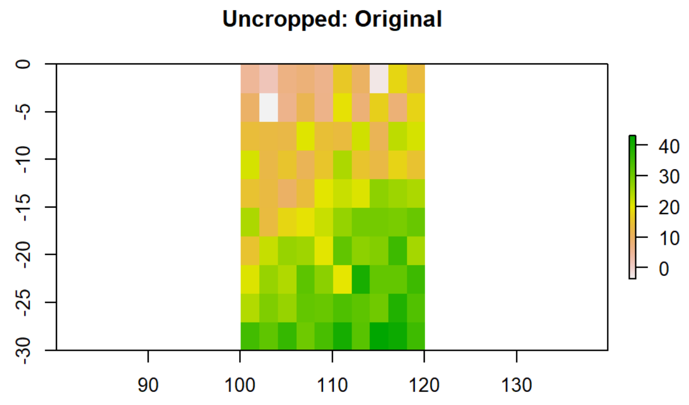
```
r2b.crop2 = crop(r2b, extent(r2c))
r2b.crop2
```

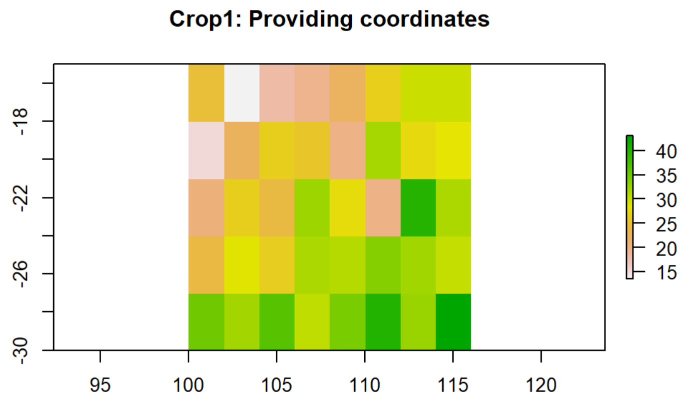
```
## class       : RasterLayer
## dimensions  : 7, 5, 35  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 110, 120, -21, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -2.523808, 35.48655  (min, max)
```

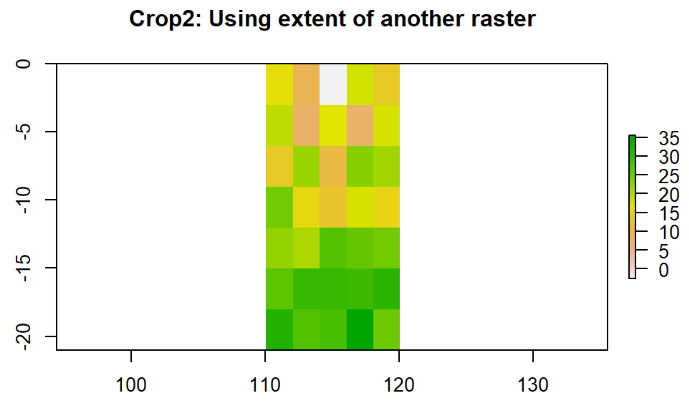
```
# Plotting the RasterLayer and Manually Drawing  the limits of the Extent object with 'drawEx
tent()'
# To try it out remove the comment symbol ('#') from the front of the commands below
plot(r2b)
```



```
#new.extent = drawExtent()  # To Select the Limits of the new Extent object click twice on th
e map
#r2b.crop3 = crop(r2b, new.extent)
#r2b.crop3

# Plot Original and 3 Cropped Rasters
par(mfrow=c(2,2))
plot(r2b, main="Uncropped: Original")
plot(r2b.crop1, main="Crop1: Providing coordinates")
plot(r2b.crop2, main="Crop2: Using extent of another raster")
#plot(r2b.crop3, main="Crop3: Cropped interactively")
```







11.1.2 'merge()'

`merge()` : Merges 2 or more Raster* objects to create a single a new Raster object with a larger (or equal) spatial extent.

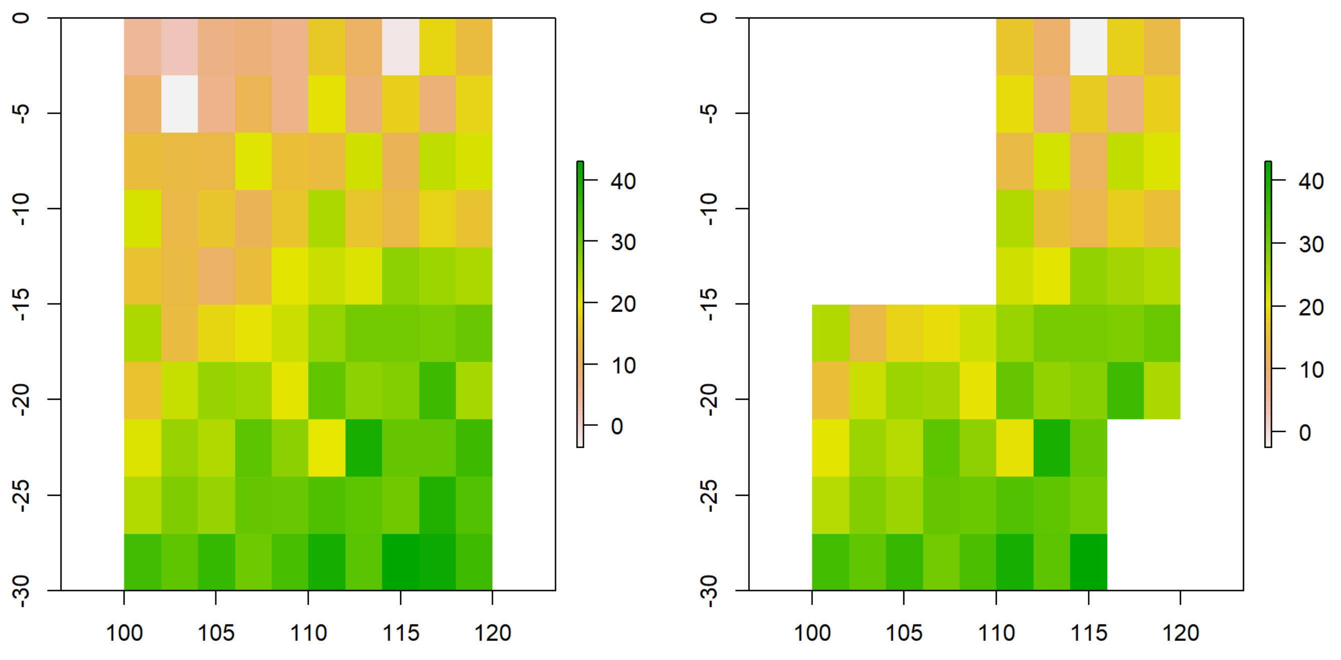
The Raster objects must have the same origin and resolution (so their cells neatly fit into a single larger raster). If this is not the case, first adjust one of the Raster objects with use (dis)aggregate or resample (see below).

If objects overlap, the values get priority in the same order as the arguments (i.e. the values of the Raster object that is first in the sequence of arguments will be retained).

NA values are ignored (except when `overlap=FALSE`).

`merge` can directly save the resulting Raster object into a file.

```
r2b.merge.c1andc2 = merge(r2b.crop1, r2b.crop2)
par(mfrow=c(1,2))
plot(r2b)
plot(r2b.merge.c1andc2)
```



11.1.3 ‘trim()’ and ‘extend()’

11.1.3.1 ‘trim()’

Trim (shrink) a Raster* object by removing outer rows and columns that contain the same value (e.g. NA). The multi-layer case is analogous to the single-layer case.

```
# Single-layer case
# ~~~~~~~~~~~~~~~~

# Create raster and Add NAs
r1a.NAs = r1a
r1a.NAs[1:2,] = NA
r1a.NAs[nrow(r1a.NAs),] = NA

# Trim
r1a.trim = trim(r1a.NAs, values=NA)

# Visualize results
dim(r1a); dim(r1a.NAs); dim(r1a.trim)
```

```
## [1] 10 15  1
```

```
## [1] 10 15  1
```

```
## [1]  7 15  1
```

```
summary(r1a)[6]; summary(r1a.NAs)[6]; summary(r1a.trim)[6]  # Count of NAs
```

```
## [1] 0
```

```
## [1] 45
```

```
## [1] 0
```

```
# Multi-layer case
# ~~~~~~~~~~~~~~~~
s1a.NAs = stack(r1a.NAs, r1a.NAs)

# Trim
s1a.trim = trim(s1a.NAs, values=NA)

# Visualize results
dim(s1a); dim(s1a.NAs); dim(s1a.trim)
```

```
## [1] 10 15  2
```

```
## [1] 10 15  2
```

```
## [1]  7 15  2
```

```
# summary(s1a)[6]; summary(s1a.NAs)[6]; summary(s1a.trim)[6]  # Summary doesn't work for mult
i-layered objects
```

11.1.3.2 ‘extend()’

Extend returns a Raster* object with a larger spatial extent, by adding new rows and/or columns with a give value (e.g.NA). The output Raster object has the outer minimum and maximum coordinates of the input Raster and Extent arguments. Thus, all of the cells of the original raster are included.

The new extent is provided via the `y` argument. It can be provided in two different ways: * Providing and extent object. This should be a numeric vector of 1, 2, or 4 elements. * Providing an object from which an extent can be extracted * Providing a numeric vector with the number of rows and columns to be added at each side. This vector can have length 2 (i.e.rows and columns) or length 1 if the number of rows and columns is equal.

The argument `value` is used to assign a value to the new cells.

The multi-layer case is analogous to the single-layer case

This function has replaced function “expand” (to avoid a name conflict with the Matrix package).

```
# Single-layer case
# ~~~~~~~~~~~~~~~~

# This function has replaced function "expand" (to avoid a name conflict with the Matrix pack
age).
r1a
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
r2a
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

```
# Method 1: Providing a new extent object
new.extent = extent(90, 125, -45, 10)
r2a.extended.m1 = extend(r2a, y=new.extent, value=NA)
r2a.extended.m1
```

```
## class       : RasterLayer
## dimensions  : 18, 17, 306  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 90, 124, -45, 9  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

```
# Method 2: Providing an object from which an extent can be extracted
r2a.extended.m2 = extend(r2a, y=r1a, value=NA)
r2a.extended.m2
```

```
## class       : RasterLayer
## dimensions  : 60, 180, 10800  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

```
# Method 3: Providing a numeric vector with the number of rows and columns to be added at eac
h side.
r2a.extended.m3 = extend(r2a, y=c(3,5), value=NA)
r2a.extended.m3
```

```
## class       : RasterLayer
## dimensions  : 16, 20, 320  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 90, 130, -39, 9  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

```
# Multi-layer case
# ~~~~~~~~~~~~~~~~

# Create multi-layer objects
s1a = stack(r1a, r1a)
s2a = stack(r2a, r2a)

# This function has replaced function "expand" (to avoid a name conflict with the Matrix pack
age).
s1a
```

```
## class       : RasterStack
## dimensions  : 10, 15, 150, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2
## min values  :       1,       1
## max values  :       1,       1
```

```
s2a
```

```
## class       : RasterStack
## dimensions  : 10, 10, 100, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       : layer.1, layer.2
## min values  :       1,       1
## max values  :     100,     100
```

```
# Method 1: Providing a new extent object
new.extent = extent(90, 125, -45, 10)
s2a.extended.m1 = extend(s2a, y=new.extent, value=NA)
s2a.extended.m1
```

```
## class       : RasterBrick 
## dimensions  : 18, 17, 306, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 90, 124, -45, 9  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer.1, layer.2 
## min values  :       1,       1 
## max values  :     100,     100
```

```
# Method 2: Providing an object from which an extent can be extracted
s2a.extended.m2 = extend(s2a, y=s1a, value=NA)
s2a.extended.m2
```

```
## class       : RasterBrick 
## dimensions  : 60, 180, 10800, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer.1, layer.2 
## min values  :       1,       1 
## max values  :     100,     100
```

```
# Method 3: Providing a numeric vector with the number of rows and columns to be added at eac
h side.
s2a.extended.m3 = extend(s2a, y=c(3,5), value=NA)
s2a.extended.m3
```

```
## class       : RasterBrick 
## dimensions  : 16, 20, 320, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 2, 3  (x, y)
## extent      : 90, 130, -39, 9  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 
## data source : in memory
## names       : layer.1, layer.2 
## min values  :       1,       1 
## max values  :     100,     100
```

11.2 Changing Resolution

11.2.1 Changing the Resolution by an Integer Factor (e.g. 2 times smaller)

It can be achieved with the functions: `aggregate` and `disaggregate` .

Changing the Resolution by an Integer Factor is not always possible

11.2.1.1 ‘aggregate()’

`aggregate` : Aggregates a Raster* object to create a new RasterLayer or RasterBrick with a lower resolution (i.e. fewer larger cells). Aggregation groups rectangular areas to create this larger cells. The value for the resulting cells is computed with a user-specified function (e.g. `mean`). Different aggregation functions can be used in the x and y direction.

```
r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
r2b.aggrfactor2 = aggregate(r2b, fact=2, fun=mean)
r2b.aggrfactor2
```

```
## class       : RasterLayer
## dimensions  : 5, 5, 25  (nrow, ncol, ncell)
## resolution  : 4, 6  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 3.131091, 37.21142  (min, max)
```

11.2.1.2 'disaggregate()'

`disaggregate` : Does the opposite than 'aggregate'. It disaggregates a RasterLayer to create a new RasterLayer with a higher resolution (i.e. more smaller cells). The values in the new RasterLayer are the same as in the larger original cells; unless you specify method="bilinear", in which case values are locally interpolated (using the resample function).

```
r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
r2b.aggrfactor2
```

```
## class       : RasterLayer
## dimensions  : 5, 5, 25  (nrow, ncol, ncell)
## resolution  : 4, 6  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 3.131091, 37.21142  (min, max)
```

```
r2b.aggrf2.disaggrf2a = disaggregate(r2b.aggrfactor2, fact=2)  # Cells same values as origina
l
r2b.aggrf2.disaggrf2a
```

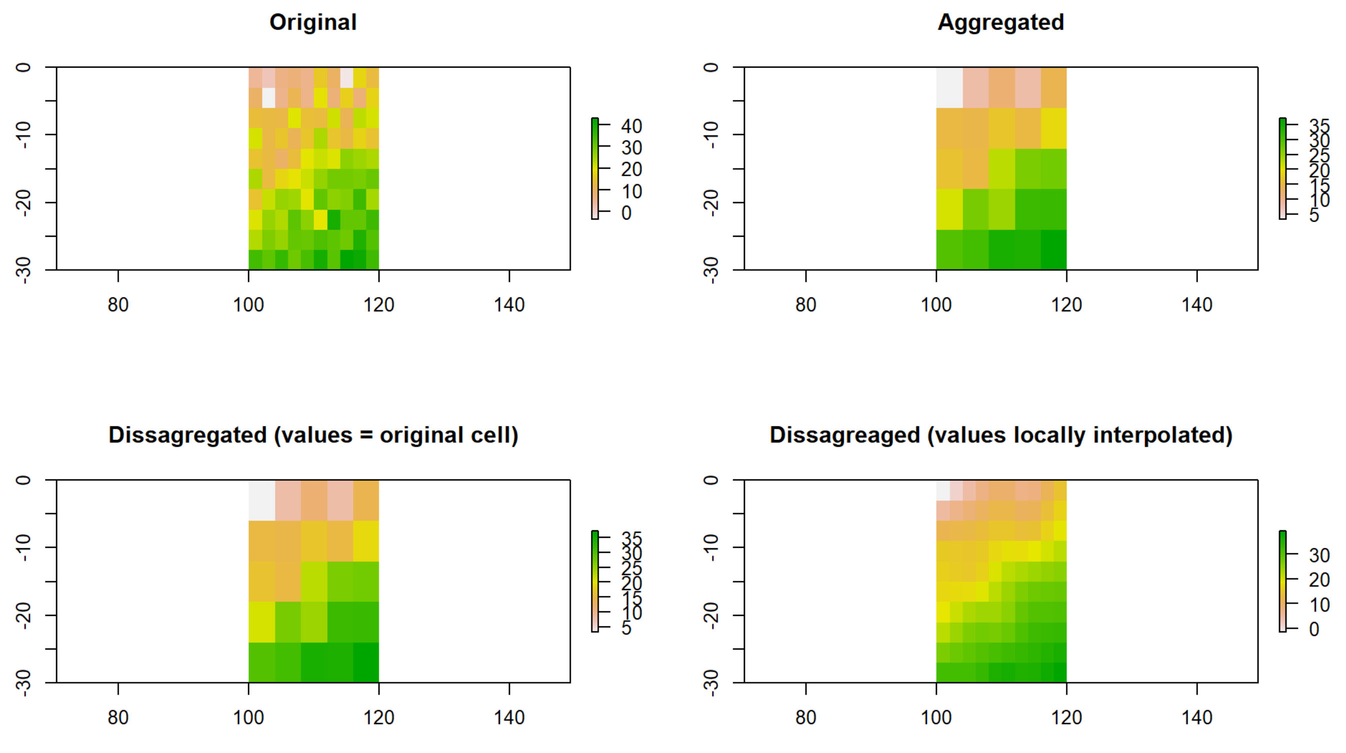
```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 3.131091, 37.21142  (min, max)
```

```
r2b.aggrf2.disaggrf2b = disaggregate(r2b.aggrfactor2, fact=2, method='bilinear')  # Cells wit
h locally interpolated values
r2b.aggrf2.disaggrf2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -1.534625, 39.38085  (min, max)
```

Plot original, aggregated, and disaggregated rasters.

```
par(mfrow=c(2,2))
plot(r2b, main="Original")
plot(r2b.aggrfactor2, main="Aggregated")
plot(r2b.aggrf2.disaggrf2a, main="Dissagregated (values = original cell)")  # Looks the same
 as disaggregated cells have the same value than the original cell
plot(r2b.aggrf2.disaggrf2b, main="Dissagreaged (values locally interpolated)")
```



11.2.2 Changing the Resolution in Other Cases

11.2.2.1 Shift

`shift` : Moves the location of a Raster* (or Spatial*) object in the x and/or y direction. Important arguments:

- x : Numeric argument containing the shift in the horizontal (i.e. longitude) direction
- y : Numeric argument containing the shift in the vertical (i.e. latitude) direction

```
# Shift raster 10 degress of longitude and 5 degrees of latitude
r1a
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
r1a.shift.x10.y5 = shift(r1a, x=10, y=5)
r1a.shift.x10.y5
```

```
## class       : RasterLayer
## dimensions  : 10, 15, 150  (nrow, ncol, ncell)
## resolution  : 24, 18  (x, y)
## extent      : -170, 190, -85, 95  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

11.2.2.2 Resample

`resample` : Transfers values between Raster* objects with different origin and/or resolution. If Raster* objects also have different projections (i.e. coordinate reference system), `projectRaster` can be used to re-project one of the objects (see next section).

Before using 'resample', the use of other raster functions should be considered (i.e. `aggregate` , `disaggregate` , `crop` , `extend` , `merge`) as they might be more appropriate to achieve the required task.

`method` : Argument indicating the method to use to compute the value for the new RasterLayer:

- "bilinear" for bilinear interpolation
- "ngb" for nearest neighbor

```
# Rasters with different dimensions and resolutions, but equal extents and projections
# ------------------------------------------------------------------------------------

# Create and Fill raster to be resampled
r.10x10 = raster(nrow=5, ncol=5)
r.10x10[] = 1:ncell(r.10x10)

# Create raster to be resampled to
r.15x15 = raster(nrow=15, ncol=15)
r.10x10
```

```
## class       : RasterLayer
## dimensions  : 5, 5, 25  (nrow, ncol, ncell)
## resolution  : 72, 36  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 25  (min, max)
```

```
r.15x15
```

```
## class       : RasterLayer
## dimensions  : 15, 15, 225  (nrow, ncol, ncell)
## resolution  : 24, 12  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

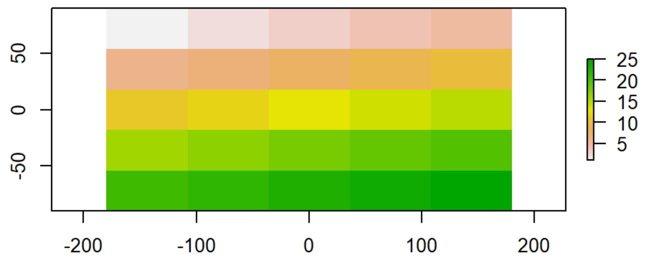
```
# Resample
r.15x15.bl = resample(r.10x10, r.15x15, method='bilinear')



# Rasters with different dimensions, resolutions and extents, but equal projections
# ---------------------------------------------------------------------------------

# Create raster to be resampled by shifting the previous raster (now also different extents)
r.10x10.shift.x25.y15 = shift(r.10x10, x=25, y=15) # Shift raster 10 degs. of long and 15 deg
s. of lat.
r.10x10.shift.x25.y15
```

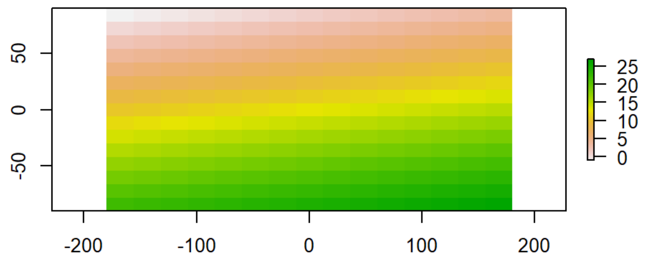
```
## class       : RasterLayer
## dimensions  : 5, 5, 25  (nrow, ncol, ncell)
## resolution  : 72, 36  (x, y)
## extent      : -155, 205, -75, 105  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 25  (min, max)
```

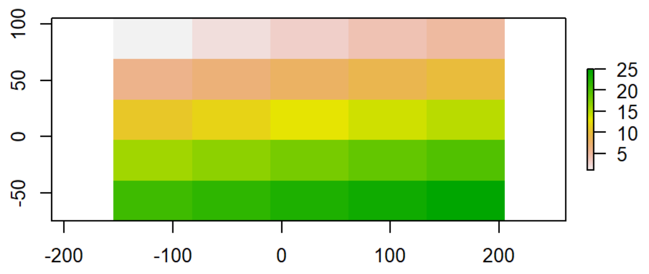
```
# Create raster to be resampled to
r.20x20 = raster(nrow=20, ncol=20)
r.20x20
```

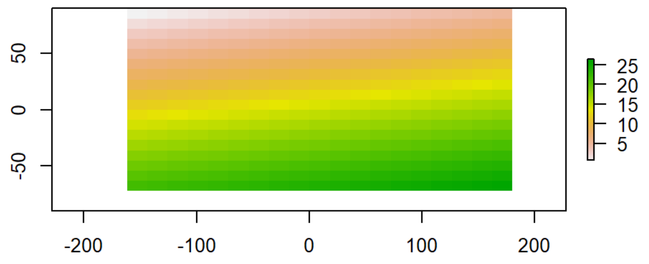
```
## class       : RasterLayer
## dimensions  : 20, 20, 400  (nrow, ncol, ncell)
## resolution  : 18, 9  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
# Resample
r.20x20.sbl = resample(r.10x10.shift.x25.y15, r.20x20, method='bilinear')



# Plot rasters
# ------------
par(mfrow=c(2,2))
plot(r.10x10)
plot(r.15x15.bl)
plot(r.10x10.shift.x25.y15)
plot(r.20x20.sbl)
```









```
# For rasters with different dimensions, resolutions, extents, and projections use
# `projectRaster` to reproject raster, see next.
```

11.3 Defining & Changing Spatial Projections

11.3.1 Define a Spatial Projection

`projection` : Get or set the Cordinate Reference System (CRS) of a Raster* object. It Obtains or Adds information for a Raster object, but doesn't change the Projection.

```
r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
projection(r2b)
```

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

```
r2d = raster(ncol=10, nrow=10, xmn=-1000, xmx=1000, ymn=-100, ymx=900)
r2d  # Provided an strange Spatial Extent, so CRS = NA
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 200, 100  (x, y)
## extent      : -1000, 1000, -100, 900  (xmin, xmax, ymin, ymax)
## coord. ref. : NA
```

```
new.CRS = crs("+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84")
projection(r2d) =new.CRS
r2d
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 200, 100  (x, y)
## extent      : -1000, 1000, -100, 900  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84
```

11.3.2 Changing a Spatial Projection

`projectRaster` : Project the values of a Raster* object to a new Raster* object with another projection (i.e. CRS).

It can be done by:

- *Providing the new projection as a single argument*: The function sets the extent and resolution of the new object.
- *Providing a Raster* object with the properties that the input data should be projected to*: Has more control over the transformation (e.g. to assure that the new object lines up with other datasets).

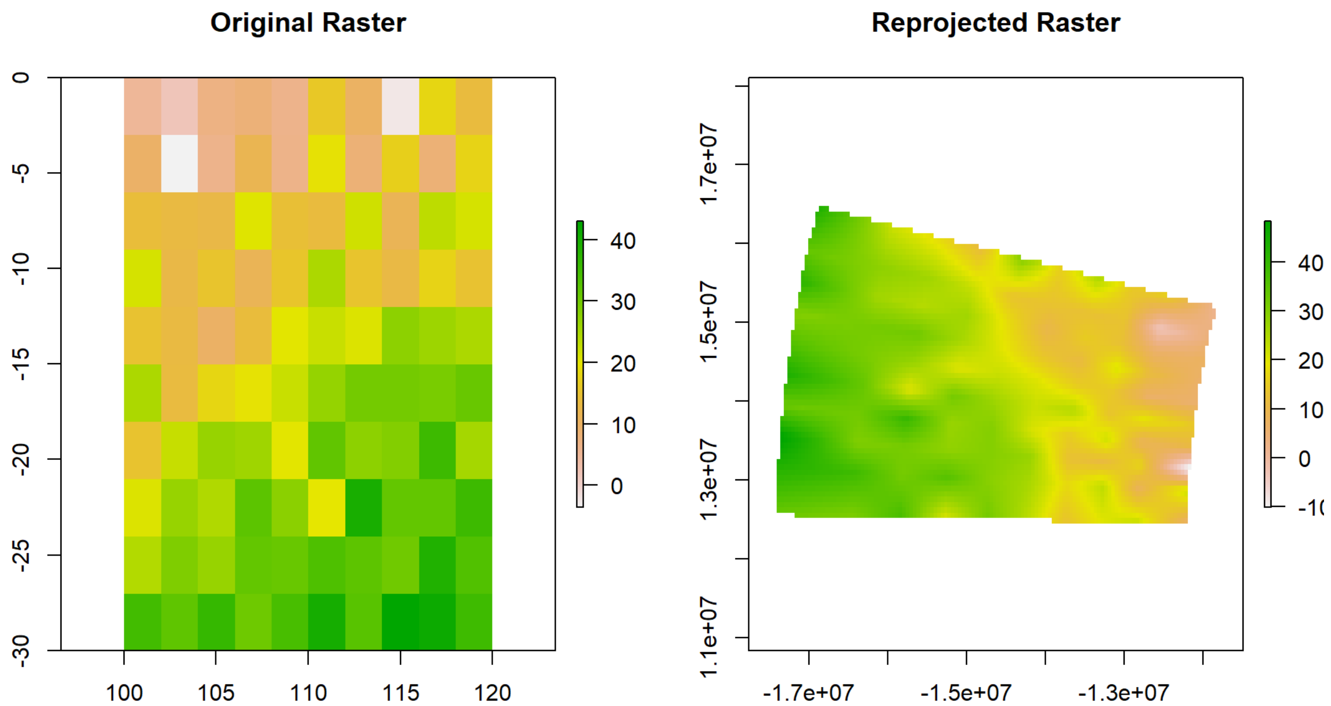
```
r2b.newCRS = projectRaster(r2b, crs=new.CRS)
r2b
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 3  (x, y)
## extent      : 100, 120, -30, 0  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -3.583128, 43.06404  (min, max)
```

```
r2b.newCRS
```

```
## class       : RasterLayer
## dimensions  : 69, 142, 9798  (nrow, ncol, ncell)
## resolution  : 44200, 68200  (x, y)
## extent      : -17765520, -11489120, 12121026, 16826826  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84
## data source : in memory
## names       : layer
## values      : -10.12194, 48.40965  (min, max)
```

```
 # Plot the Original and Re-projected Rasters
par(mfrow=c(1,2))
plot(r2b, main="Original Raster")
plot(r2b.newCRS, main="Reprojected Raster")
```



11.4 Setting the Spatial Attributes of All Rasters to the same values at once

The Extent, Resolution, and Projection can be set to be the same in all rasters using the `spatial_sync_raster` function from the `spatial.tools` package.

12 Raster Prediction

12.1 Interpolation

Interpolation predicts values for the cells in a raster from the values of a limited number of sample data points (cells).

The raster function `interpolate` returns a raster object with predictions, and requires (at least): a raster object and a model object. The models uses the cell coordinates as predictor variables and the cell values as the dependent variable. Here we will use a Thin Plate Spline Model to create (i.e. fit) the model object. Multiple models can be fitted using:

- `gstat` package: inverse distance weighted (IDW), Kriging, ordinary Kriging, universal Kriging, co-Kriging
- `fields` package: thin plate spline, cubic splines, Kriging
- Could write your own function to do so

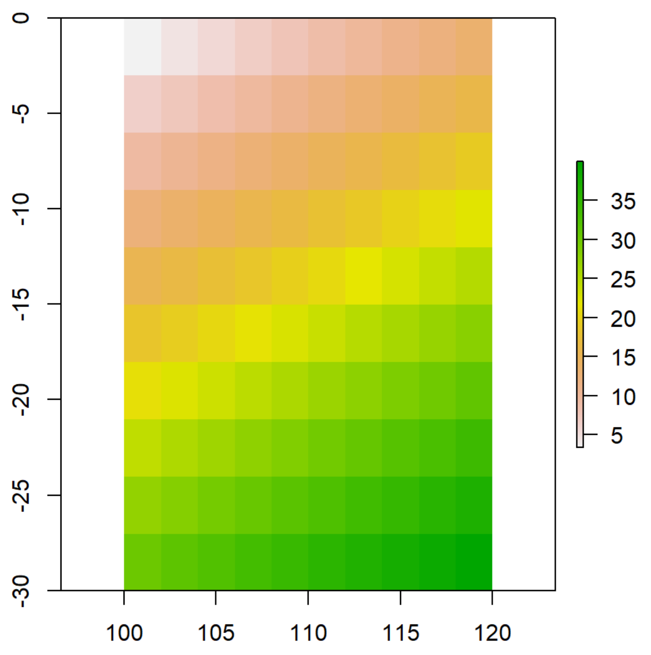
```
library(fields)   # For function 'Tps': Thin plate spline (Tps)

# Agregate r2b layer to create a corse layer
r2b.aggr2 = aggregate(r2b, fact=2)
r2b.aggr4 = aggregate(r2b, fact=4)
# Get raster Coordinates
r2b.aggr2.xy = data.frame(xyFromCell(r2b.aggr2, 1:ncell(r2b.aggr2)))
r2b.aggr4.xy = data.frame(xyFromCell(r2b.aggr4, 1:ncell(r2b.aggr4)))
# Get rater values
r2b.aggr2.vals = getValues(r2b.aggr2)
r2b.aggr4.vals = getValues(r2b.aggr4)
# Fit Thin Plate Spline (smoothing parameter chosen by genralized cross-valuations; see Tps h
elp)
r2b.aggr2.tps = Tps(r2b.aggr2.xy, r2b.aggr2.vals)
```

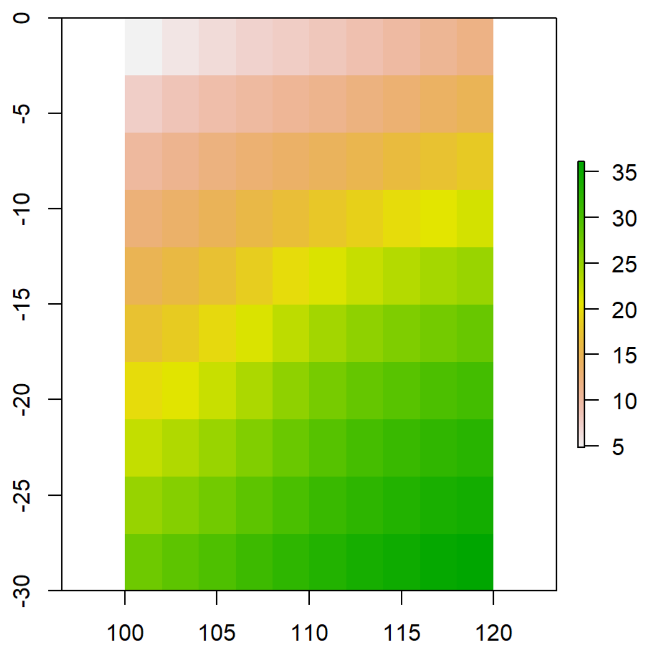
```
## Warning:
## Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
##   (GCV) Generalized Cross-Validation
##    minimum at  right endpoint  lambda  =  182.9905 (eff. df= 3.001008 )
```

```
r2b.aggr4.tps = Tps(r2b.aggr4.xy, r2b.aggr4.vals)
```

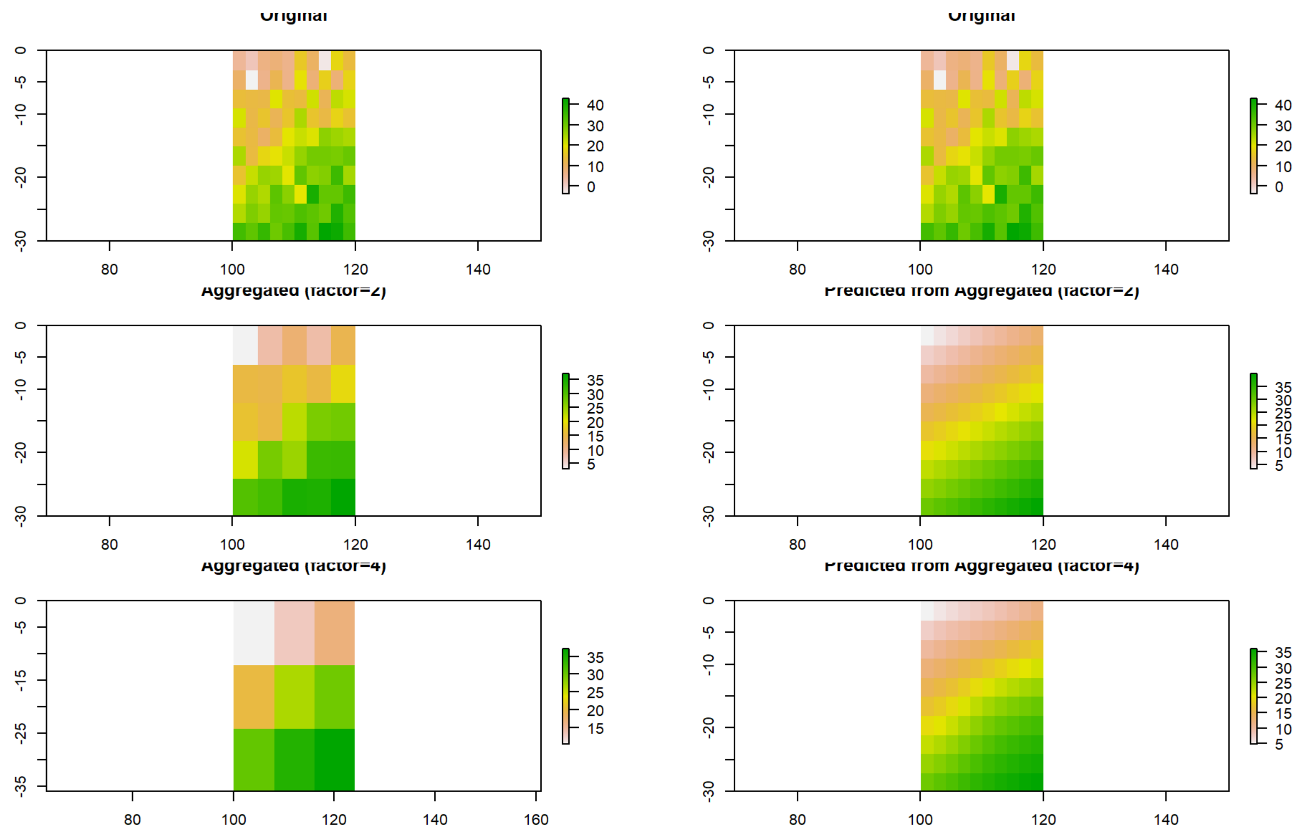
```
## Warning:
## Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
##   (GCV) Generalized Cross-Validation
##    minimum at  right endpoint  lambda  =  0.0007376219 (eff. df= 8.549996
## )
```

```
# Create 'blank' rasters (i.e. spatial features of r2b, but values=NA) to contain Pred. Value
s (i.e. Interpolation)
r2b.aggr2.pred = r2b.aggr4.pred = raster(r2b)
# Interpolate
r2b.aggr2.pred = interpolate(r2b.aggr2.pred, r2b.aggr2.tps)
r2b.aggr4.pred = interpolate(r2b.aggr4.pred, r2b.aggr4.tps)
# View Interpolated Surfaces
par(mfrow=c(1,2))
plot(r2b.aggr2.pred); plot(r2b.aggr4.pred)
```





```
# Mask Interpolated Sufaces to Extent & Resolution of Initial Raster
r2b.aggr2.pred = mask(r2b.aggr2.pred, r2b)
r2b.aggr4.pred = mask(r2b.aggr4.pred, r2b)
# Plot Original Raster, Aggregated Rasters (by factor of 2 and 3), and Predicted (Spline Inte
rpolated) Rasters
par(mfrow=c(3,2))
plot(r2b, main="Original")
plot(r2b, main="Original")
plot(r2b.aggr2, main="Aggregated (factor=2)")
plot(r2b.aggr2.pred, main="Predicted from Aggregated (factor=2)")
plot(r2b.aggr4, main="Aggregated (factor=4)")
plot(r2b.aggr4.pred, main="Predicted from Aggregated (factor=4)")  # Similar results because
 of regular filling
```



```
# Remove Objects containing the string "aggr" (i.e. those created and used in this section)
rm(list = ls(pattern="aggr"))
```

12.2 Prediction

As with the function `interpolate`, the function `predict` also returns a raster object with spatial predictions and has as inputs a raster object and a model object. However, these are different than in the function ‘interpolate’:

- *Raster object*: is typically a multi-layer raster (i.e. RasterStack or RasterBrick)
- *Model object*: is a model of any class that has a ‘predict’ method, including: glm, gam, randomForest, and is fitted to raster data.

This approach of predicting from a fitted model to raster data is commonly used in remote sensing (e.g. for the classification of satellite images) and in ecology (e.g. for species distribution modelling).

For further informatin see help pages for the predict function in the package raster (http://127.0.0.1:25588/library/raster/html/predict.html) and the package dismo (http://127.0.0.1:25588/library/dismo/html/dismo-package.html).

```
# Temperature Raster: With Gradient from West to East.
temp.ras = raster(nrow=10, ncol=16)
temp.ras[] = colFromCell(temp.ras,1:ncell(temp.ras))*10/4 + rnorm(ncell(temp.ras),sd=1)
temp.ras
```

```
## class       : RasterLayer
## dimensions  : 10, 16, 160  (nrow, ncol, ncell)
## resolution  : 22.5, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1.389249, 41.37007  (min, max)
```

```
# Humidity Raster: With Gradient from North to South.
humid.ras = raster(nrow=10, ncol=16)
humid.ras[] = 20 + rowFromCell(humid.ras,1:ncell(humid.ras)) * 7.5 + rnorm(ncell(humid.ras),s
d=2)
humid.ras
```

```
## class       : RasterLayer
## dimensions  : 10, 16, 160  (nrow, ncol, ncell)
## resolution  : 22.5, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 24.47817, 97.41214  (min, max)
```

```
# Create a RasterBrick object with Environmental Conditions (Temperature & Humidity)
EnvCond.brick = brick(temp.ras,humid.ras)
names(EnvCond.brick) = c("Temperature","Humidity")
EnvCond.brick
```

```
## class       : RasterBrick
## dimensions  : 10, 16, 160, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 22.5, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : Temperature,  Humidity
## min values  :    1.389249, 24.478169
## max values  :    41.37007,  97.41214
```

```
# Presence & Absence Cells
  # Present: High Temp & High Humidity
p = cbind(floor(runif(15,min=7,max=11)),floor(runif(15,min=9,max=17)))
  # Absent: 1-5: Low Temp & Low Humidity; 6-10: Low Temp & High Humidity; 11-15: High Temp &
 Low Humidity
a = cbind( floor(c(runif(10,min=1,max=6),runif(5,min=6,max=11))),
           floor(c(runif(5,min=1,max=9),runif(5,min=9,max=17),runif(5,min=1,max=9))) )

# Presence/Absece Raster (for Visualization)
pa.ras = raster(nrow=10, ncol=16)
pa.ras[] = NA
pa.ras[p] = 1
pa.ras[a] = 0
pa.ras
```

```
## class       : RasterLayer
## dimensions  : 10, 16, 160  (nrow, ncol, ncell)
## resolution  : 22.5, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0, 1  (min, max)
```

```
pa.ras[]
```

```
##   [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA  0 NA NA NA  0 NA NA NA NA
##  [24]  0 NA NA NA  0 NA NA NA NA NA NA NA NA NA NA  0 NA  0 NA NA  0  0 NA
##  [47] NA NA NA  0 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  0 NA
##  [70] NA NA NA NA NA NA NA NA NA NA NA NA  0 NA  0 NA NA NA NA NA NA NA NA
##  [93] NA NA NA NA NA NA NA NA  0 NA NA NA  1 NA  1  1 NA NA NA  1 NA NA NA
## [116] NA NA NA NA NA NA NA  1 NA NA NA NA  1  0 NA NA NA  0 NA NA NA  1 NA
## [139] NA  1 NA NA  1 NA NA NA NA NA NA NA NA NA NA NA  1 NA NA  1 NA NA
```

```
# Plot Rasters
par(mfrow=c(2,2))
plot(temp.ras, main="Temperature (Â°C): Longitudinal gradient")
plot(humid.ras, main="Humidity (%): Latitudinal gradient")
plot(pa.ras, main="Presence & Absences for Training")

# Extract Values for Points
pa.xy = rbind(cbind(1,p), cbind(0,a))
pa.xy
```

```
##       [,1] [,2] [,3]
##  [1,]    1    8   11
##  [2,]    1    7    9
##  [3,]    1    9   12
##  [4,]    1    9    9
##  [5,]    1    7   12
##  [6,]    1   10   14
##  [7,]    1    9    9
##  [8,]    1    8   11
##  [9,]    1    8   16
## [10,]    1    7   12
## [11,]    1    9   15
## [12,]    1    7   11
## [13,]    1   10   11
## [14,]    1    7   11
## [15,]    1    7   16
## [16,]    0    2    8
## [17,]    0    4    2
## [18,]    0    5    4
## [19,]    0    3    7
## [20,]    0    2    3
## [21,]    0    1   15
## [22,]    0    3   12
## [23,]    0    3    9
## [24,]    0    2   12
## [25,]    0    3   13
## [26,]    0    9    5
## [27,]    0    6    4
## [28,]    0    7    5
## [29,]    0    6    2
## [30,]    0    9    1
```

```
cells.indices = cellFromRowCol(EnvCond.brick, row=pa.xy[,2], col=pa.xy[,3])
pa.envcond.df = data.frame(cbind(pa=pa.xy[,1], extract(EnvCond.brick, cells.indices)))
#pa.envcond.df = data.frame(cbind(pa=pa.xy[,1], extract(EnvCond.brick, pa.xy[,2:3])))
pa.envcond.df
```

```
##    pa Temperature Humidity
## 1   1   26.014427 80.13996
## 2   1   22.973031 71.21403
## 3   1   28.062903 87.38564
## 4   1   22.646710 86.80553
## 5   1   29.343879 72.73097
## 6   1   35.378989 95.12276
## 7   1   22.646710 86.80553
## 8   1   26.014427 80.13996
## 9   1   39.878039 82.41568
## 10  1   29.343879 72.73097
## 11  1   35.498587 87.75041
## 12  1   26.168327 76.05656
## 13  1   28.108070 93.54173
## 14  1   26.168327 76.05656
## 15  1   40.958117 69.99511
## 16  0   18.120375 35.15829
## 17  0    6.397417 50.43451
## 18  0   10.864054 57.95063
## 19  0   17.182124 43.61141
## 20  0    8.003834 35.32117
## 21  0   36.724121 26.99667
## 22  0   28.571414 43.90640
## 23  0   23.235162 43.47022
## 24  0   30.200287 33.07525
## 25  0   32.180682 42.81642
## 26  0   11.720406 83.00840
## 27  0   11.039122 67.02925
## 28  0   11.792528 77.87521
## 29  0    4.365097 71.29406
## 30  0    1.389249 90.20378
```

```
# Fit Model (GLM)
glm.fit = glm(formula=pa~., data=pa.envcond.df)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = pa ~ ., data = pa.envcond.df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.46075  -0.18358  -0.01935   0.21358   0.43986
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.171086   0.185473  -6.314 9.29e-07 ***
## Temperature  0.025643   0.004411   5.813 3.47e-06 ***
## Humidity     0.016038   0.002311   6.939 1.86e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.06500629)
##
##     Null deviance: 7.5000  on 29  degrees of freedom
## Residual deviance: 1.7552  on 27  degrees of freedom
## AIC: 7.9774
##
## Number of Fisher Scoring iterations: 2
```

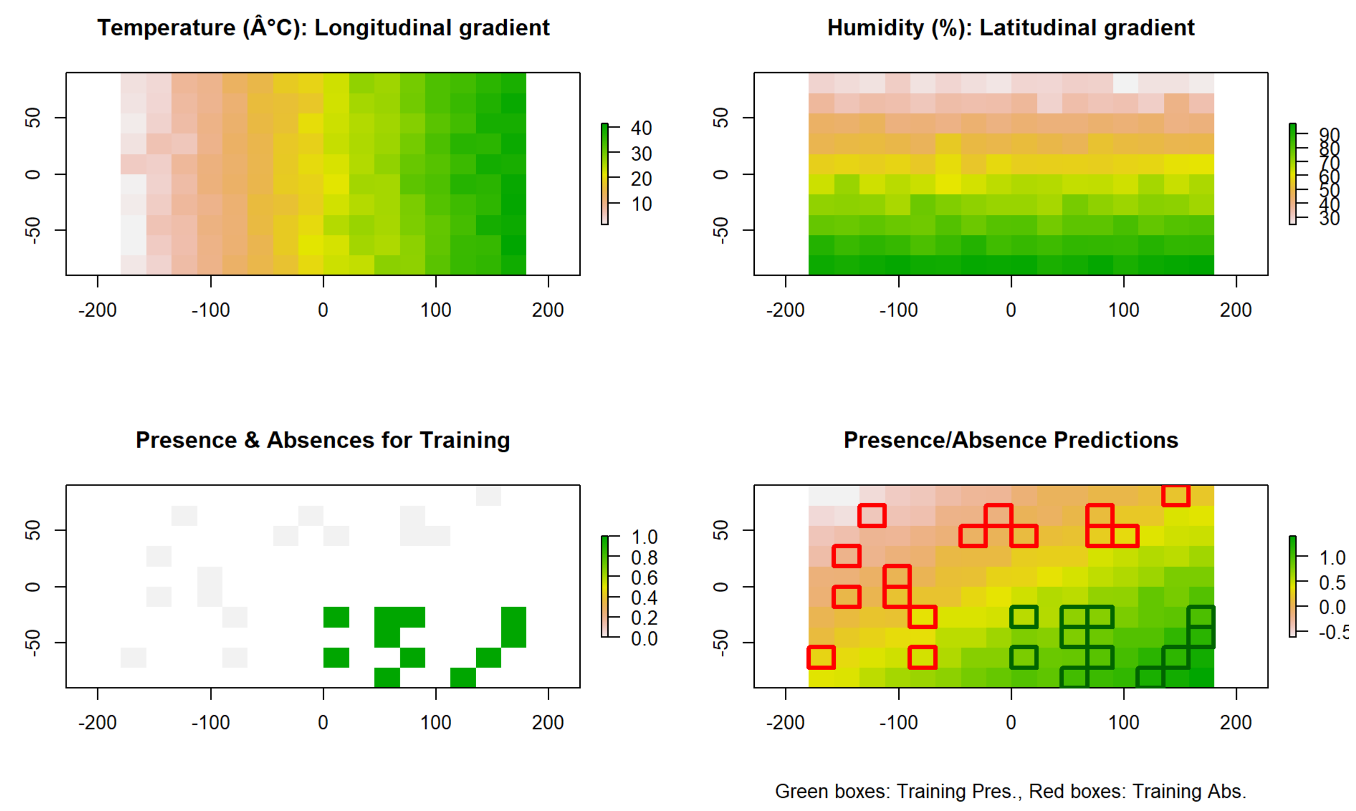
```
# Predict to a Raster
pred.ras = predict(EnvCond.brick, glm.fit, progress='text')
```

```
##
  |
  |                                                                 |   0%
  |
  |================                                                 |  25%
  |
  |================================                                 |  50%
  |
  |=================================================                |  75%
  |
  |=================================================================| 100%
##
```

```
pred.ras
```

```
## class       : RasterLayer
## dimensions  : 10, 16, 160  (nrow, ncol, ncell)
## resolution  : 22.5, 18  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -0.6204496, 1.404305  (min, max)
```

```
# Plot Predictions & PA Points
plot(pred.ras, main="Presence/Absence Predictions",
     sub="Green boxes: Training Pres., Red boxes: Training Abs.")
p.rc = rowColFromCell(pred.ras, cells.indices[1:15])
a.rc = rowColFromCell(pred.ras, cells.indices[16:30])
#p.rc; a.rc
for( cnt in 1:15 ) {
    plot(extent(pred.ras, p.rc[cnt,1], p.rc[cnt,1], p.rc[cnt,2], p.rc[cnt,2]), add=TRUE, col=
'darkgreen', lwd=3)
    plot(extent(pred.ras, a.rc[cnt,1], a.rc[cnt,1], a.rc[cnt,2], a.rc[cnt,2]), add=TRUE, col=
'red', lwd=3)
}
```



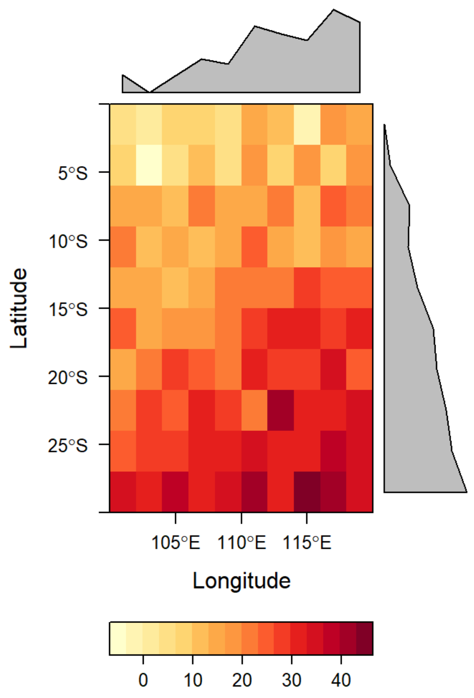
13 Other GIS operations with rasters in R

First we create a dataset with simulated elevation values to be used in this section.

```
# rb2 created above (see 'Visualising Rasters') contains Latitudinal and Longitudinal gradien
ts,
# with the former 3 times stronger.
r2b[]
```

```
##   [1]  4.780121  1.736944  6.889259  8.038875  6.171172 16.102196  9.951327
##   [8] -2.523808 18.019132 13.707639  9.590426 -3.583128  6.199381 11.708418
##  [15]  6.413912 19.344206  8.185034 17.106459  8.098741 17.678986 14.052422
##  [22] 13.361022 12.853425 20.442539 14.500877 13.704217 21.826718 11.201254
##  [29] 23.300787 21.121238 21.235934 12.744511 15.280282 11.247760 15.461039
##  [36] 24.732835 15.306017 12.881048 17.647537 15.022321 15.017614 13.350011
##  [43]  9.901972 13.815558 20.022739 22.240784 20.692412 27.484770 25.901197
##  [50] 24.642700 24.789695 13.547285 18.037063 19.605525 22.402358 26.727755
##  [57] 29.865164 29.867752 29.287679 31.001823 15.145634 22.541422 26.552205
##  [64] 25.791529 20.017065 31.688927 27.509136 28.339053 35.486548 25.257955
##  [71] 20.706820 26.596051 24.286767 32.242393 27.680065 19.901360 39.923136
##  [78] 31.443303 31.526053 35.502415 24.155372 28.685411 26.468013 31.504920
##  [85] 30.950129 33.623521 32.070314 30.168423 38.902838 33.343174 34.976939
##  [92] 31.924456 36.501460 30.356163 34.299321 40.280377 32.520743 43.064039
##  [99] 41.332182 35.267483
```

```
levelplot(r2b, par.settings=YlOrRdTheme)
```



```
# Making all values positive and stretched out by 10 (to recreate some decent 'elevation')
r2b.elev = (r2b + abs(cellStats(r2b, min)) + 0.1) * 10
r2b.elev[]
```

```
##   [1]  84.63248  54.20071 105.72386 117.22003  98.54299 197.85323 136.34455
##   [8]  11.59320 217.02259 173.90766 132.73554   1.00000  98.82508 153.91545
##  [15] 100.97039 230.27333 118.68162 207.89587 117.81868 213.62114 177.35550
##  [22] 170.44150 165.36552 241.25666 181.84005 173.87345 255.09846 148.84382
##  [29] 269.83915 248.04366 249.19061 164.27638 189.63409 149.30887 191.44166
##  [36] 284.15963 189.89145 165.64176 213.30664 187.05449 187.00741 170.33138
##  [43] 135.85099 174.98686 237.05867 259.23911 243.75539 311.67898 295.84325
##  [50] 283.25827 284.72822 172.30413 217.20191 232.88653 260.85486 304.10883
##  [57] 335.48291 335.50879 329.70807 346.84951 188.28762 262.24550 302.35333
##  [64] 294.74657 237.00193 353.72054 311.92263 320.22181 391.69676 289.41083
##  [71] 243.89947 302.79179 279.69894 359.25520 313.63192 235.84487 436.06264
##  [78] 351.26430 352.09180 391.85543 278.38500 323.68539 301.51140 351.88048
##  [85] 346.33257 373.06649 357.53442 338.51550 425.85965 370.26302 386.60066
##  [92] 356.07584 401.84588 340.39290 379.82448 439.63505 362.03870 467.47166
##  [99] 450.15309 389.50610
```

```
cellStats(r2b.elev, min)
```

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

13.1 Distance & Direction

13.1.1 Distance functions

Attention must be paid to the raster projection as it can noticiably affect the results (small areas often use a UTM projection, while large areas/world oftne use '+proj=longlat'). Illustrations of the different results obtained using different projections can be seen in the examples for the functions `pointDistance` and `gridDistance` below.

The units of the distances returned by all distance functions in the package `raster` are:

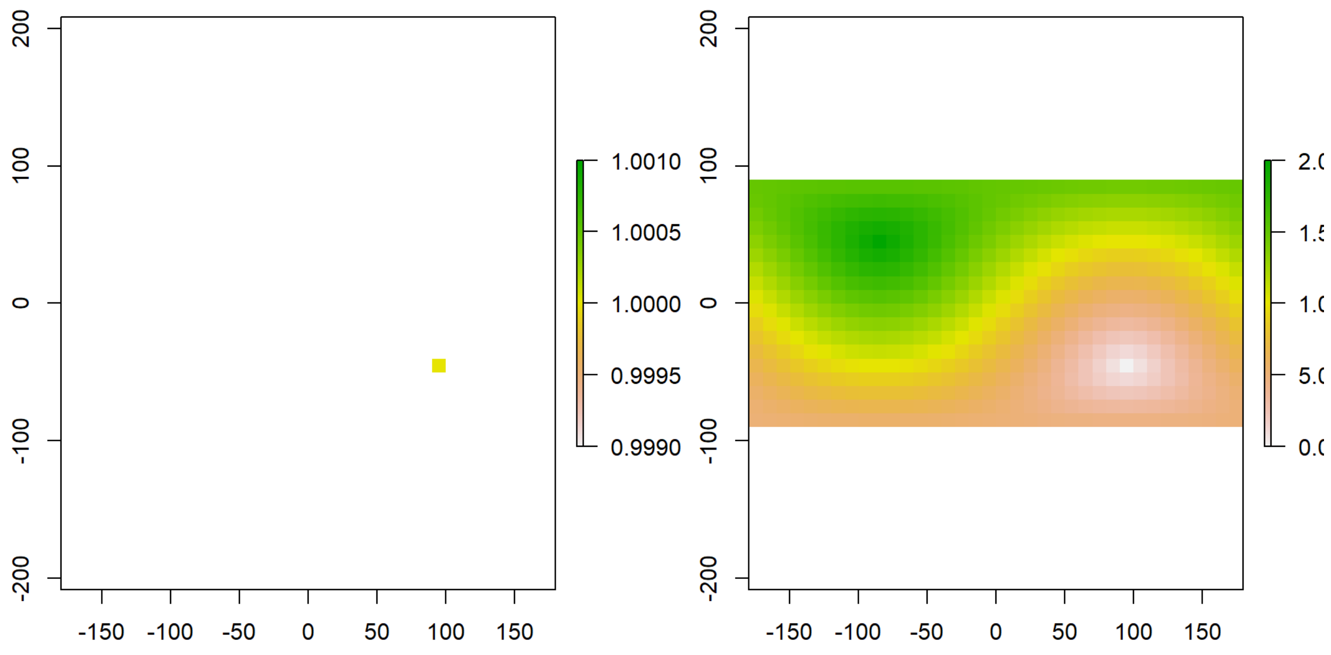
- *Meters* if the RasterLayer is not projected (i.e. '+proj=longlat')
- *Map units (typically also meters)* if the RasterLayer is projected.

More advanced distance function can be found in other packages, including:

- `gdistance` : Computes Distances and Routes on Geographical Grids. It includes advanced distances, such as: cost distance and resistance distance. See package vignette here (https://cran.r-project.org/web/packages/gdistance/vignettes/gdistance1.pdf). A tutorial on how to calculate Cummulative Distances usign this package can be found here (http://personal.colby.edu/personal/m/mgimond/Spatial/Distance_rook_vs_queen_vs_knight.html)
- `geosphere` : Uses spherical trigonometry for geographic applications. It computes great-circle distances (and related measures)for angular (i.e. spherical longitude/latitude coordinates) locations. See package vignette here (https://cran.r-project.org/web/packages/geosphere/vignettes/geosphere.pdf)

`distance` : Computes the distance from all NA cells to cells that are not NA.

```
# Create and Fill raster with values (all cells = NA, but one 'half-way' in the Souther hemis
phere
r = raster(ncol=18*2,nrow=9*2) # 1 cell for each 10x10 degrees
r[] = NA
r[cellFromXY(r, cbind(90,-45))] = 1

# Compute distances. NOTE: Using a representation of 'Earth` (i.e. ~ Spheric), so Distances f
irst increase and they decrease.
r.dists = distance(r)

# Plot rasters
par(mfrow=c(1,2))
plot(r)
plot(r.dists)
```



`distanceFromPoints` : Computes the distance from a set of points to all cells of a Raster* object.

```
# Distance from 1 point
# ====================
 # Projection = 'longlat'
 # ----------------------
r = raster(ncol=18*2,nrow=9*2) # 1 cell for each 10x10 degrees
point1.xy = c(90,-45)
d.point1.longlat = distanceFromPoints(r, point1.xy)
d.point1.longlat
```

```
## class       : RasterLayer
## dimensions  : 18, 36, 648  (nrow, ncol, ncell)
## resolution  : 10, 10  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 394171.6, 19626586  (min, max)
```

```
 # Projection = 'utm'
 # ------------------
crs(r) = '+proj=utm +zone12 +datum=WGS84' # Change projection
d.point1 = distanceFromPoints(r, point1.xy)
d.point1.utm = distanceFromPoints(r, point1.xy)
d.point1.utm
```

```
## class       : RasterLayer
## dimensions  : 18, 36, 648  (nrow, ncol, ncell)
## resolution  : 10, 10  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 5, 295.1694  (min, max)
```

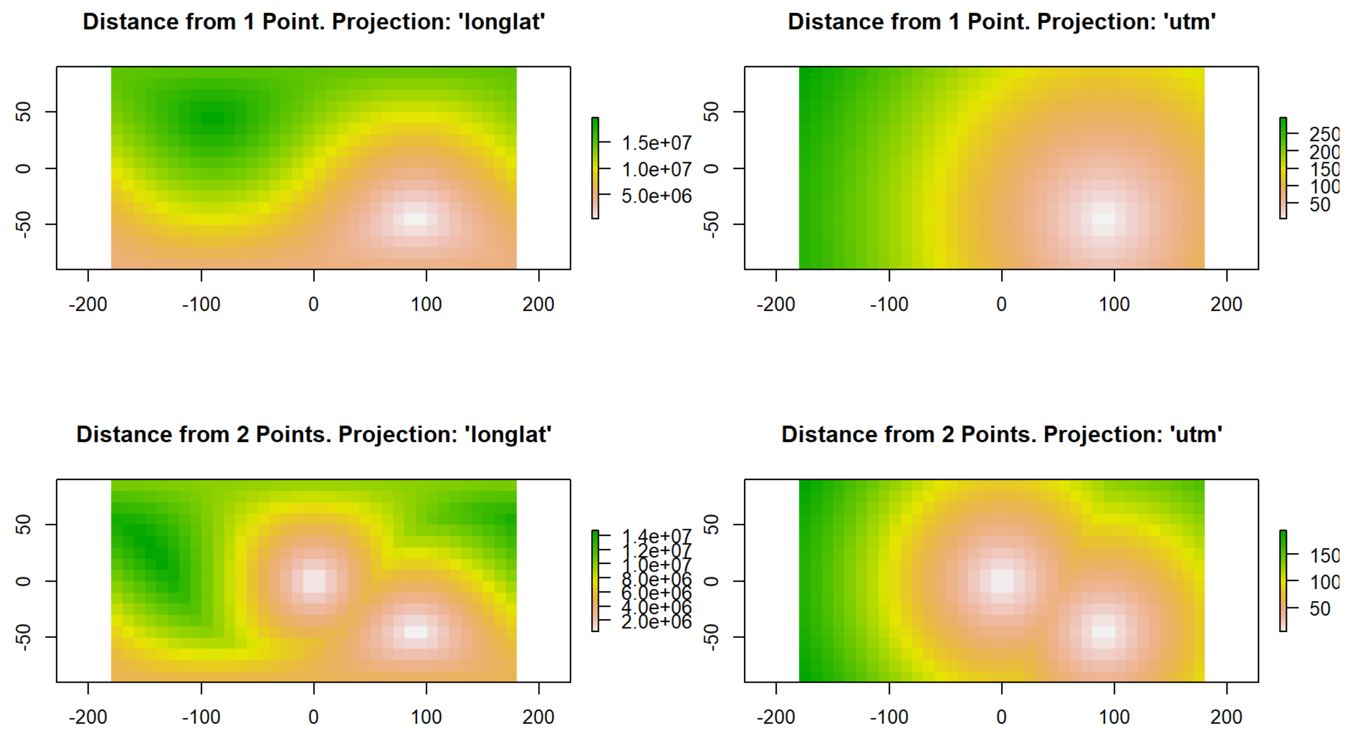
```
# Distance from 2 points
# =====================
 # Projection = 'longlat'
 # ----------------------
r = raster(ncol=18*2,nrow=9*2) # 1 cell for each 10x10 degrees
points2n3.xy = cbind(c(0,90),c(0,-45))
d.points2n3.longlat = distanceFromPoints(r, points2n3.xy)
d.points2n3.longlat
```

```
## class       : RasterLayer
## dimensions  : 18, 36, 648  (nrow, ncol, ncell)
## resolution  : 10, 10  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 394171.6, 14705105  (min, max)
```

```
 # Projection = 'utm'
 # ------------------
crs(r) = '+proj=utm +zone12 + datum=WGS84' # Change projection
d.points2n3.utm = distanceFromPoints(r, points2n3.xy)
d.points2n3.utm
```

```
## class       : RasterLayer
## dimensions  : 18, 36, 648  (nrow, ncol, ncell)
## resolution  : 10, 10  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +ellps=WGS84
## data source : in memory
## names       : layer
## values      : 5, 194.5508  (min, max)
```

```
# Plot Distances
# ==============
par(mfrow=c(2,2))
plot(d.point1.longlat, main="Distance from 1 Point. Projection: 'longlat'")
plot(d.point1.utm, main="Distance from 1 Point. Projection: 'utm'")
plot(d.points2n3.longlat, main="Distance from 2 Points. Projection: 'longlat'")
plot(d.points2n3.utm, main="Distance from 2 Points. Projection: 'utm'")
```



`pointDistance` : Computes the shortest geographic distance between two (set of) points. When sets of points are given, if both sets have:

- *Equal number of points*: The distance between each point and the corresponding point in the other set is calculated, except if the argument allpairs is TRUE (i.e. `allpairs=TRUE`).
- *Different number of points*: The distance between each possible pair of points among sets is calculated.

Argument `lonlat` : Logical. If:

- TRUE: Coordinates should be in degrees. The distance is calculated on the World Geodetic System (WGS) ellipsoid. The units of the calculated distances are meters.
- FALSE: Coordinates represent Euclidian space (e.g. units of meters). The distance is calculate d on a plane. The units of the calculated distances are map units (often also meters).

```
# Create Sets of Points
points.set1 = cbind(c(30,60,90),c(15,30,45))
points.set2 = cbind(c(120,150,180),c(60,75,90))
points.set3 = cbind(c(120,180),c(60, 90))
points.set1; points.set2; points.set3
```

```
##      [,1] [,2]
## [1,]   30   15
## [2,]   60   30
## [3,]   90   45
```

```
##      [,1] [,2]
## [1,]  120   60
## [2,]  150   75
## [3,]  180   90
```

```
##      [,1] [,2]
## [1,]  120   60
## [2,]  180   90
```

```
# Calculate Distances
  # Between Points
pointDistance(c(0,0), c(10,10), longlat=TRUE)   # Distances on WGS ellipsoid
```

```
## [1] 1565109
```

```
pointDistance(c(0,0), c(10,10), longlat=FALSE)   # Distances on a plane
```

```
## [1] 14.14214
```

```
sqrt(10^2+10^2)  # Distances on a plane are the Euclidian Distances
```

```
## [1] 14.14214
```

```
  # Between a Point and a Set of Points: Calculate all possible pairs (here 3 pairs)
pointDistance(c(0,0), points.set2, longlat=TRUE)   # Distances on WGS ellipsoid
```

```
## [1] 11621024 11448656 10001966
```

```
pointDistance(c(0,0), points.set2, longlat=FALSE)   # Distances on a plane
```

```
## [1] 134.1641 167.7051 201.2461
```

```
  # Between two Sets of Points with equal number of points: Calculate distances single pairs
 of points across both sets
pointDistance(points.set1, points.set2, longlat=TRUE)   # Distances on WGS ellipsoid: All res
ults different
```

```
## [1] 8574772 6807739 5017021
```

```
pointDistance(points.set1, points.set2, longlat=FALSE)   # Distances on a plane: All results
 equal
```

```
## [1] 100.6231 100.6231 100.6231
```

```
   # Between two Sets of Points with different number of points: Calculate distances all poss
ible pairs across both sets
pointDistance(points.set1, points.set3, longlat=TRUE)   # Distances on WGS ellipsoid
```

```
##         [,1]    [,2]
## [1,] 8574772 8342976
## [2,] 5511860 6681852
## [3,] 2594667 5017021
```

```
pointDistance(points.set1, points.set3, longlat=FALSE)   # Distances on a plane
```

```
##           [,1]     [,2]
## [1,] 100.62306 167.7051
## [2,]  67.08204 134.1641
## [3,]  33.54102 100.6231
```

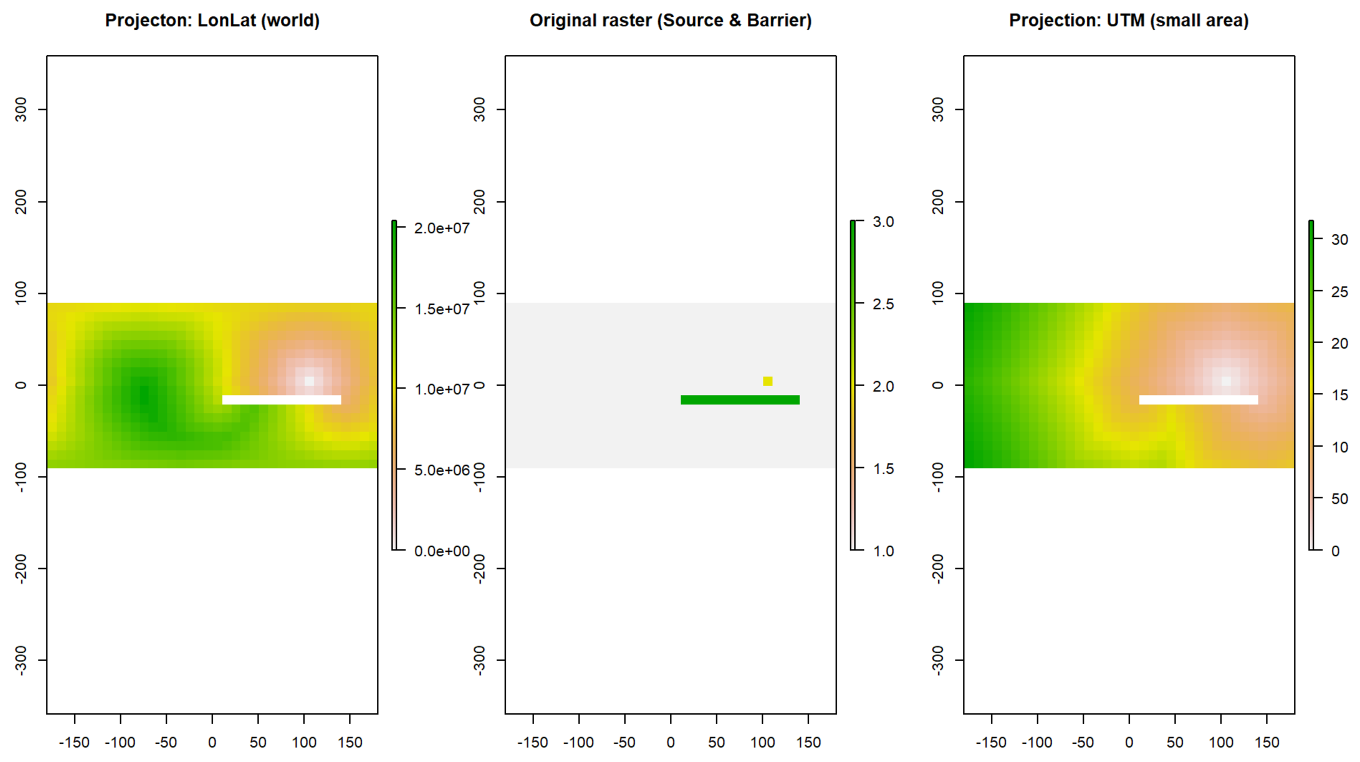
`gridDistance` : Computes the distance from a cell or set of cells to cells in a RasterLayer. The distances are computed by summing the local distances between the center of the cells. The path can follow any of the 8 neighboring cells (i.e. the 'queen' case, which is currently the only one implemented). It can be specified that certain types of cells (i.e. with a given value) cannot be traversed by the path. For example, water bodies can be excluded by providing the cell value representing them.

If the RasterLayer to be processed is big, it will be processed in chunks. This may lead to errors when of complex objects spread over different chunks (e.g. meandering rivers). Varying the chunk size might help solving these issues (see function `setOptions`).

Important arguments:

- `x` : A RasterLayer
- `origin``: Value(s) of the cell(s) from which the distance is calculated
- `omit` : Value(s) of the cell(s) that cannot be traversed.

```
# Projection: 'lonlat' (world)
# ---------------------------
r = raster(ncol=18*2,nrow=9*2) # 1 cell for each 10x10 degrees
r[] = 1
r[317] = 2
r[380:392] = 3
dist.lonlat = gridDistance(r,origin=2,omit=3)

# Projection: 'utm' (small area)
# -----------------------------
projection(r) = "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
dist.utm = gridDistance(r,origin=2,omit=3)

# Plot results
# ------------
par(mfrow=c(1,3))
plot(dist.lonlat, main="Projecton: LonLat (world)")
plot(r, main="Original raster (Source & Barrier)")
plot(dist.utm, main="Projection: UTM (small area)")
```

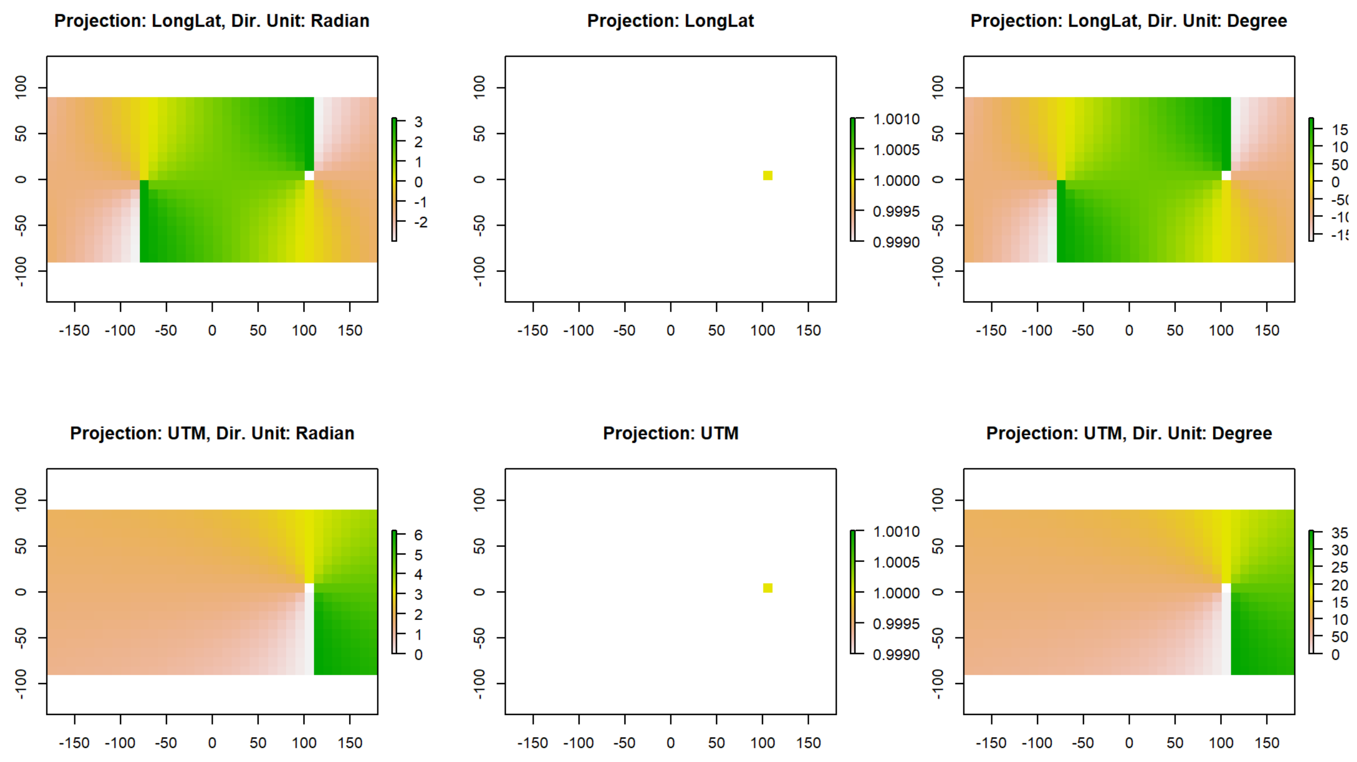


13.1.2 Direction functions

`direction` : Computes the direction (azimuth) toward (or from) the nearest cell that is not NA. Important arguments:

- ‘from’:
  - FALSE (default): Computes the direction to the nearest cell that is not NA.
  - TRUE: Computes the direction from the nearest cell that is not NA.
- `degrees` :
  - FALSE (default): Computes the direction in radians (i.e. unit of direction: radians).
  - TRUE: Computes the direction in degrees (i.e. unit of direction: degrees).

```
# Projection: LongLat
r.longlat = raster(ncol=18*2,nrow=9*2) # 1 cell for each 10x10 degrees
r.longlat[] = NA
r.longlat[317] = 1
dir.longlat.rad = direction(r.longlat)  # Computes Direction in Radians
dir.longlat.deg = direction(r.longlat, degrees=TRUE) # Computes Direction in Degrees

# Projection: UTM
r.utm = r.longlat
projection(r.utm) = "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
dir.utm.rad = direction(r.utm)  # Computes Direction in Radians
dir.utm.deg = direction(r.utm, degrees=TRUE) # Computes Direction in Degrees

# Plot results
par(mfrow=c(2,3))
plot(dir.longlat.rad, main="Projection: LongLat, Dir. Unit: Radian")
plot(r.longlat, main="Projection: LongLat")
plot(dir.longlat.deg, main="Projection: LongLat, Dir. Unit: Degree")
plot(dir.utm.rad, main="Projection: UTM, Dir. Unit: Radian")
plot(r.utm, main="Projection: UTM")
plot(dir.utm.deg, main="Projection: UTM, Dir. Unit: Degree")
```



13.2 Spatial autocorrelation Raster

To compute measures of autocorrelation in a RasterLayer, two autocorrelation measures are available: Moran’s I and Geary’s C.Both can be computed: Globally (returns a number) or Locally (returns a RasterLayer object). They use weights defined by a matrix, as in ‘focal’ (see above ‘Summarizing values in a “moving window”’). The Weights Shape and Size can be changed. The default is: ‘w=matrix(c(1,1,1,1,0,1,1,1,1), 3,3)’; i.e. a 3x3 matrix.

```
# Global autocorrelation
Moran(r2b.elev)
```

```
## [1] 0.6938611
```

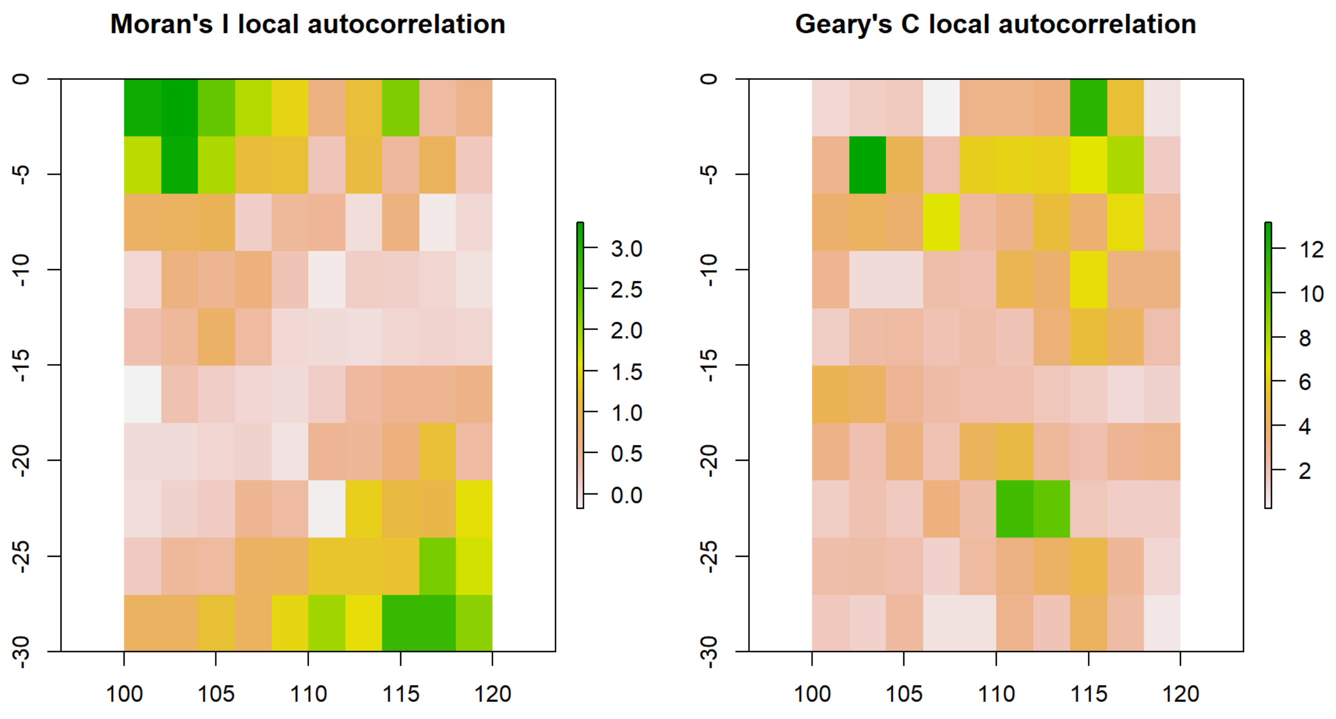
```
Moran(r2b.elev, w=matrix(c(rep(1,12),0,rep(1,12)), nrow=5))
```

```
## [1] 0.580636
```

```
Geary(r2b.elev)
```

```
## [1] 0.229886
```

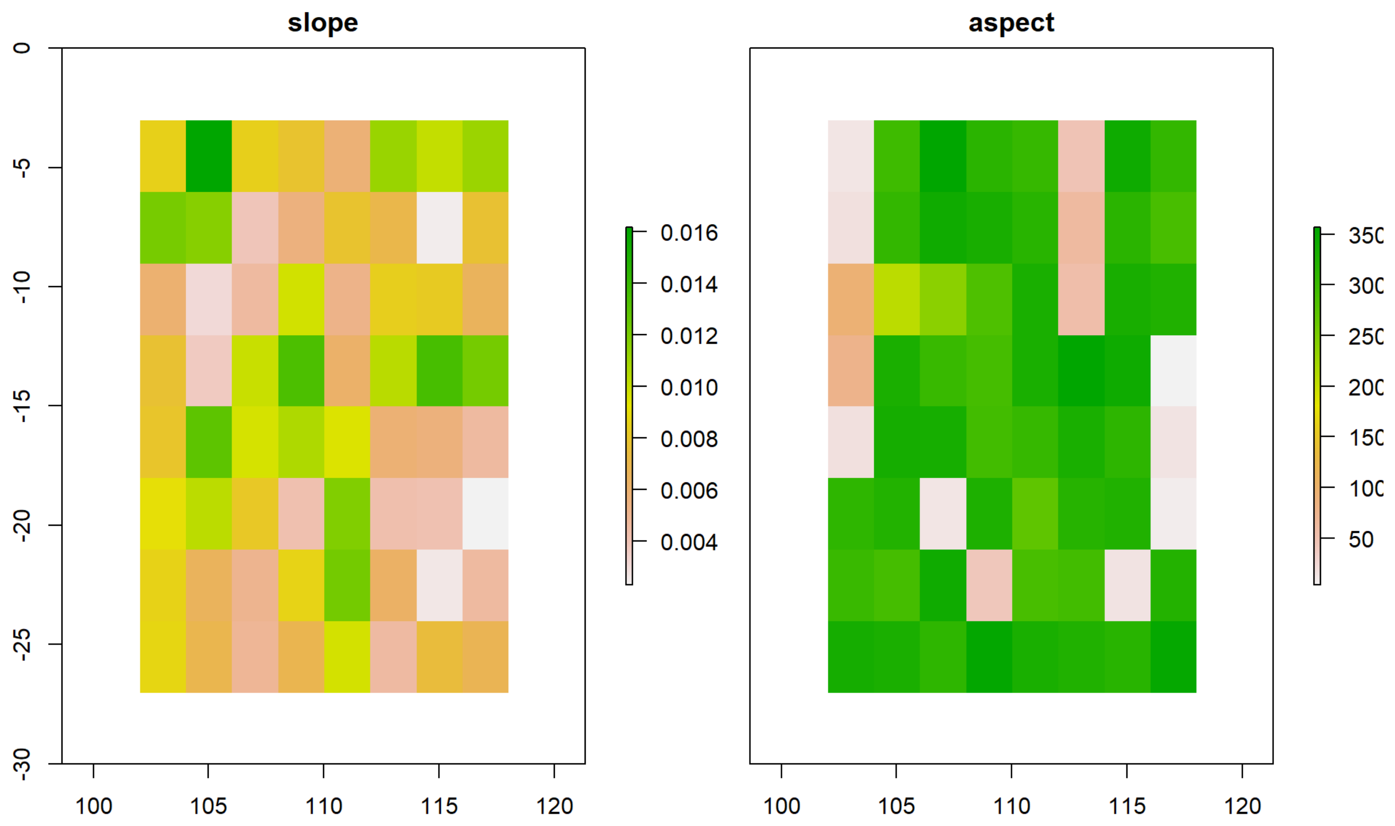
```
# Local autocorrelation
par(mfrow=c(1,2))
plot(MoranLocal(r2b.elev), main="Moran's I local autocorrelation" )
plot(GearyLocal(r2b.elev), main="Geary's C local autocorrelation"  )
```



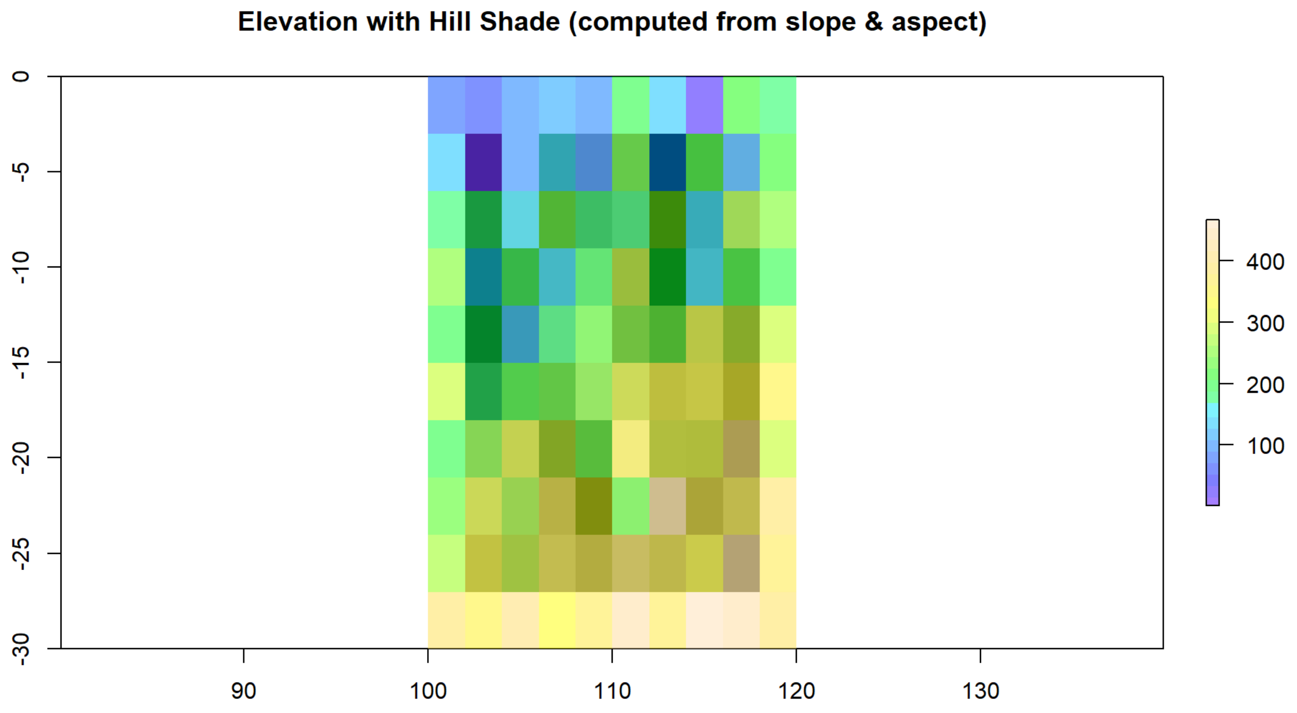
13.3 Elevation, slope, aspect

Slope & aspect can be computed with function `terrain`.

```
# Basic maps
r2b.slo.asp.deg.ras = terrain(r2b.elev, opt = c("slope", "aspect"), unit = "degrees")
#plot(r2b.elev)
plot(r2b.slo.asp.deg.ras)
```



```
# Compute: Slope & Aspect
r2b.slope = terrain(r2b.elev, opt = "Slope")
r2b.aspect = terrain(r2b.elev, opt = "Aspect")

# Nicer plot
  # Compute hill shade from Slope and Aspect layers (both in radians).
r2b.hillShade = hillShade(r2b.slope, r2b.aspect, 40, 270)
  # Plot Hill Shade layer as a backdrop and then Elevation on top as a semi-transparent laye
r.
plot(r2b.hillShade, col = grey(0:100/100), legend = FALSE, main="Elevation with Hill Shade (c
omputed from slope & aspect)")
plot(r2b.elev, col = topo.colors(25, alpha = 0.5), add = TRUE) # 'alpha': Transparency channe
l, range=[0,1] (0=transparent, 1=opaque)
```



Elevation with Hill Shade (computed from slope & aspect)
0
-5
-10
-15
-20
-25
-30
90
100
110
120
130
400
300
200
100

14 THE END