



# Geodata and algorithms in R



## Geographic raster data in R

Jannes Muenchow

DAAD summer school



# Find the slides and the code

[https://github.com/giscience-fsu/daad\\_summerschool](https://github.com/giscience-fsu/daad_summerschool)

Please install following packages:

```
install.packages(c("sf", "raster", "spData", "dplyr", "RQGIS"))
```

# Raster data model

- Continuous fields represented by pixels (cells)

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



# Raster data model

- Continuous fields represented by pixels (cells)
- One attribute value for one cell

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



# Raster data model

- Continuous fields represented by pixels (cells)
- One attribute value for one cell
- Especially suitable for continuous data without sharp borders (elevation, precipitation)

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



# Raster data model

- Continuous fields represented by pixels (cells)
- One attribute value for one cell
- Especially suitable for continuous data without sharp borders (elevation, precipitation)
- Structure: raster header (origin, resolution, ncol, nrow, crs, NAvalue) and matrix containing the data

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



# Raster data model

- Continuous fields represented by pixels (cells)
- One attribute value for one cell
- Especially suitable for continuous data without sharp borders (elevation, precipitation)
- Structure: raster header (origin, resolution, ncol, nrow, crs, NAvalue) and matrix containing the data
- Map algebra

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



# Raster data model

- Continuous fields represented by pixels (cells)
- One attribute value for one cell
- Especially suitable for continuous data without sharp borders (elevation, precipitation)
- Structure: raster header (origin, resolution, ncol, nrow, crs, NAvalue) and matrix containing the data
- Map algebra

19	38	72	18
17	31	NA	96
NA	26	16	9
14	45	50	10



Further reading: <https://geocompr.robinlovelace.net/spatial-class.html#raster-data>





# Raster data in R

Remember: the geographic raster data model is used to represent continuous surfaces. Rasters consist of a **header** and a **matrix** containing the actual values. Let's create a raster from scratch. In R we use the popular **raster** package written by Robert J. Hijmans.



# Raster data in R

Remember: the geographic raster data model is used to represent continuous surfaces. Rasters consist of a **header** and a **matrix** containing the actual values. Let's create a raster from scratch. In R we use the popular **raster** package written by Robert J. Hijmans.

```
library(raster)
elev = raster(nrow = 6, ncol = 6, res = 0.5,
              xmn = -1.5, xmx = 1.5,
              ymn = -1.5, ymx = 1.5,
              vals = 1:36)
```



# Raster data in R

elev

```
## class      : RasterLayer
## dimensions  : 6, 6, 36  (nrow, ncol, ncell)
## resolution  : 0.5, 0.5  (x, y)
## extent     : -1.5, 1.5, -1.5, 1.5  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : layer
## values     : 1, 36  (min, max)
```



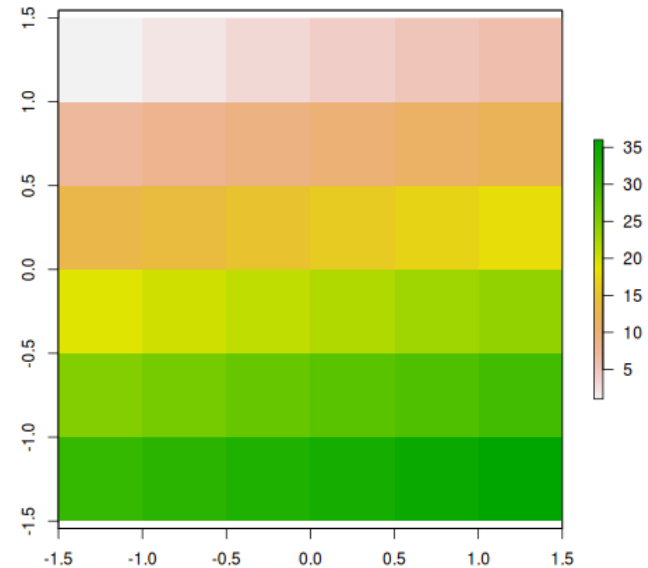
# Raster data in R

```
plot(elev)
```



# Raster data in R

```
plot(elev)
```





# Raster attribute subsetting



# Raster attribute subsetting

Since a raster is a matrix, subsetting follows the usual  $i, j$  conventions. Let's select the first and the last value.



# Raster attribute subsetting

Since a raster is a matrix, subsetting follows the usual  $i, j$  conventions. Let's select the first and the last value.

```
elev[1, 1]
```

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

```
elev[6, 6]
```

```
## [1] 36
```





# Raster attribute subsetting

Since a raster is a matrix, subsetting follows the usual  $i, j$  conventions. Let's select the first and the last value.

```
elev[1, 1]
```

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

```
elev[6, 6]
```

```
## [1] 36
```

Further reading: <https://geocompr.robinlovelace.net/attr.html#raster-subsetting>



# Spatial raster operations



# Raster spatial operations - subsetting

using coordinates:

```
extract(elev, data.frame(x = 0.75, y = 0.75))
```

```
## [1] 11
```



# Raster spatial operations - subsetting

using coordinates:

```
extract(elev, data.frame(x = 0.75, y = 0.75))
```

```
## [1] 11
```

using a SpatialObject (SpatialPointsDataFrame):



# Raster spatial operations - subsetting

using coordinates:

```
extract(elev, data.frame(x = 0.75, y = 0.75))
```

```
## [1] 11
```

using a SpatialObject (SpatialPointsDataFrame):

```
library(sf)
library(dplyr)
pt = st_point(c(0.75, 0.75)) %>%
  st_sfc %>%
  st_sf %>%
  as(., "Spatial")
# use the SpatialObject for subsetting
elev[pt]
```

```
## [1] 11
```



using another raster object:

```
clip =  
  raster(nrow = 3, ncol = 3,  
        res = 0.3, xmn = 0.6,  
        xmx = 1.5, ymn = -0.45,  
        ymx = 0.45,  
        vals = rep(1, 9))  
elev[clip]
```

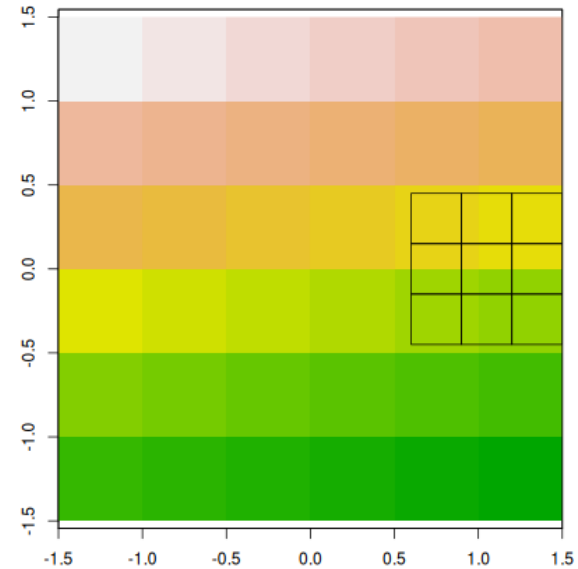
```
## [1] 17 18 23 24
```



using another raster object:

```
clip =  
  raster(nrow = 3, ncol = 3,  
        res = 0.3, xmn = 0.6,  
        xmx = 1.5, ymn = -0.45,  
        ymx = 0.45,  
        vals = rep(1, 9))  
elev[clip]
```

```
## [1] 17 18 23 24
```





# Map algebra - local operations

You may use with raster datasets:

- algebraic operators such as  $+$ ,  $-$ ,  $*$ ,  $/$
- logical operators such as  $>$ ,  $>=$ ,  $<$ ,  $=$ ,  $!$
- functions such as `abs`, `round`, `ceiling`, `floor`, `trunc`, `sqrt`, `log`, `log10`, `exp`, `cos`, `sin`, `max`, `min`, `range`, `prod`, `sum`, `any`, `all`.





# Map algebra - local operations

You may use with raster datasets:

- algebraic operators such as +, -, \*, /
- logical operators such as >, >=, <, ==, !
- functions such as abs, round, ceiling, floor, trunc, sqrt, log, log10, exp, cos, sin, max, min, range, prod, sum, any, all.

```
elev + 1  
elev^2  
elev / 4
```



# Map algebra - local operations

You may use with raster datasets:

- algebraic operators such as +, -, \*, /
- logical operators such as >, >=, <, ==, !
- functions such as abs, round, ceiling, floor, trunc, sqrt, log, log10, exp, cos, sin, max, min, range, prod, sum, any, all.

```
elev + 1  
elev^2  
elev / 4
```

Cell-by-cell operations are also called local operations. The calculation of the NDVI is one of the most popular examples.



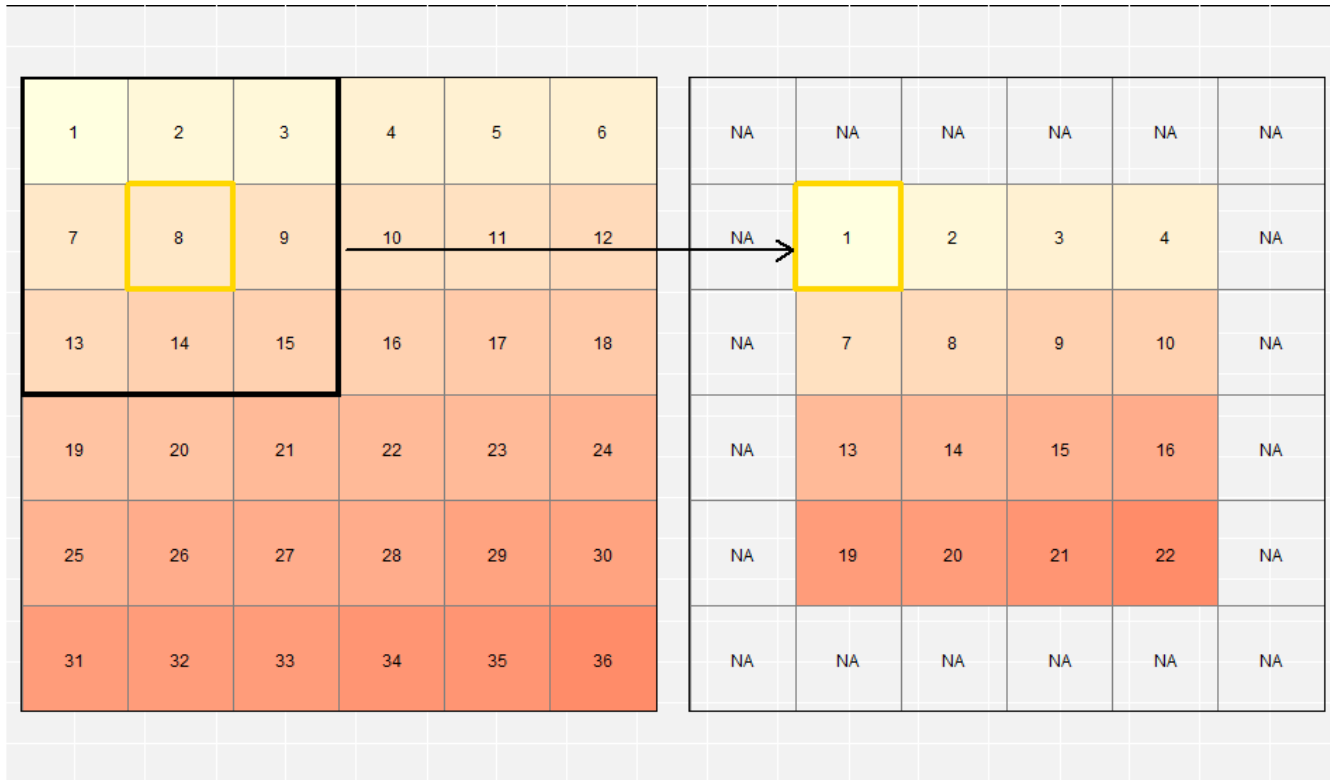
# Map algebra - focal operations

While local functions operate on one cell, though possibly from multiple layers, **focal** operations take into account a central cell and its neighbors. The neighborhood (also named kernel, filter or moving window) under consideration is typically of size 3-by-3 cells (that is the central cell and its eight surrounding neighbors) but can take on any other (not necessarily rectangular) shape as defined by the user.



# Map algebra - focal operations

```
r_focal = focal(elev, w = matrix(1, nrow = 3, ncol = 3), fun = min)
```





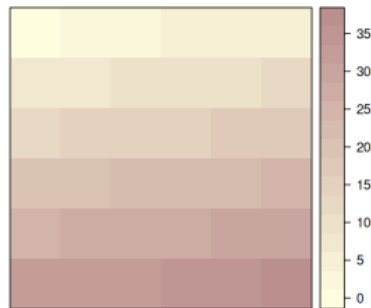
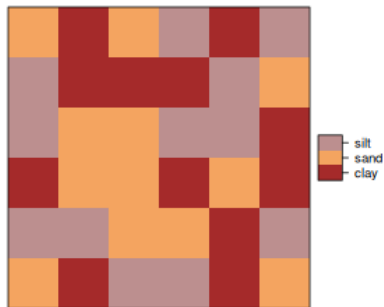
# Map algebra - zonal operations

Zonal operations are similar to focal operations. The difference is that zonal filters can take on any shape instead of just a predefined window. Let's compute the mean elevation for different soil grain size classes.



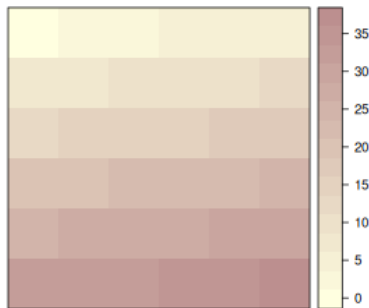
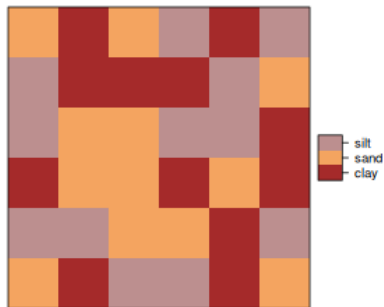
# Map algebra - zonal operations

Zonal operations are similar to focal operations. The difference is that zonal filters can take on any shape instead of just a predefined window. Let's compute the mean elevation for different soil grain size classes.



# Map algebra - zonal operations

Zonal operations are similar to focal operations. The difference is that zonal filters can take on any shape instead of just a predefined window. Let's compute the mean elevation for different soil grain size classes.



```
library(spData)
zonal(elev, grain, fun = "mean")
```

##		zone	mean
##	[1,]	1	17.75
##	[2,]	2	18.50
##	[3,]	3	19.25



# Map algebra - global operations

Global operations are a special case of zonal operations with the entire raster dataset representing a single zone. The most common global operations are descriptive statistics for the entire raster dataset such as the minimum or maximum.

```
cellStats(elev, min)
```

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

```
cellStats(elev, max)
```

```
## [1] 36
```

```
cellStats(elev, sd)
```

```
## [1] 10.53565
```





# Map algebra - global operations

Global operations are a special case of zonal operations with the entire raster dataset representing a single zone. The most common global operations are descriptive statistics for the entire raster dataset such as the minimum or maximum.

```
cellStats(elev, min)
```

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

```
cellStats(elev, max)
```

```
## [1] 36
```

```
cellStats(elev, sd)
```

```
## [1] 10.53565
```

Further reading: <https://geocompr.robinlovelace.net/spatial-operations.html#spatial-ras>



# Your turn

- `Attach data("dem", package = "RQGIS")`. Retrieve the altitudinal values of the 10th row.
- Sample randomly 10 coordinates of dem with the help of the `sp::coordinates()`-command, and extract the corresponding altitudinal values.
- `Attach data("random_points", package = "RQGIS")` and find the corresponding altitudinal values. Plot altitude against `spri`.
- Compute the hillshade of dem (hint: `?hillShade`). Overlay the hillshade with dem while using an appropriate level of transparency.



# Geometric operations on raster data



# Intersecting geometry

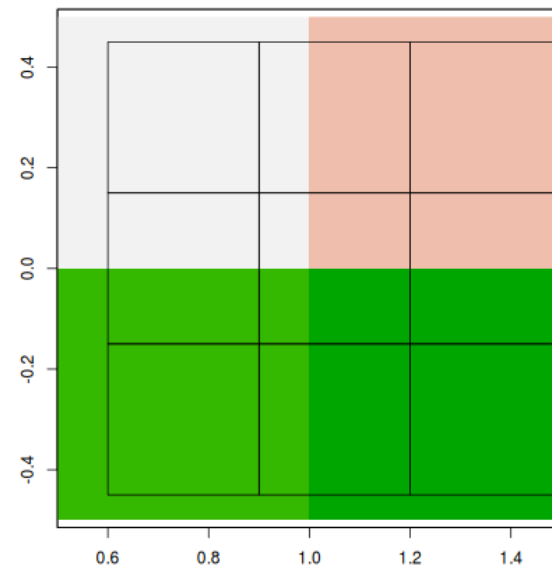
If you want the intersecting geometry of two rasters, use the spatial subsetting syntax and set the drop-parameter to FALSE.



# Intersecting geometry

If you want the intersecting geometry of two rasters, use the spatial subsetting syntax and set the drop-parameter to FALSE.

```
elev[clip, drop = FALSE]
```





# Intersecting geometry

which in fact is the same as using `intersect()`:

```
raster::intersect(elev, clip)
```

```
## class      : RasterLayer
## dimensions : 2, 2, 4  (nrow, ncol, ncell)
## resolution : 0.5, 0.5  (x, y)
## extent     : 0.5, 1.5, -0.5, 0.5  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : layer
## values     : 17, 24  (min, max)
```



# Aggregation and disaggregation

Change the resolution of a raster:

```
elev_agg =  
  aggregate(elev, fact = 2,  
            fun = mean)
```

Use `dissaggregate()` for  
increasing the spatial resolution of a  
raster

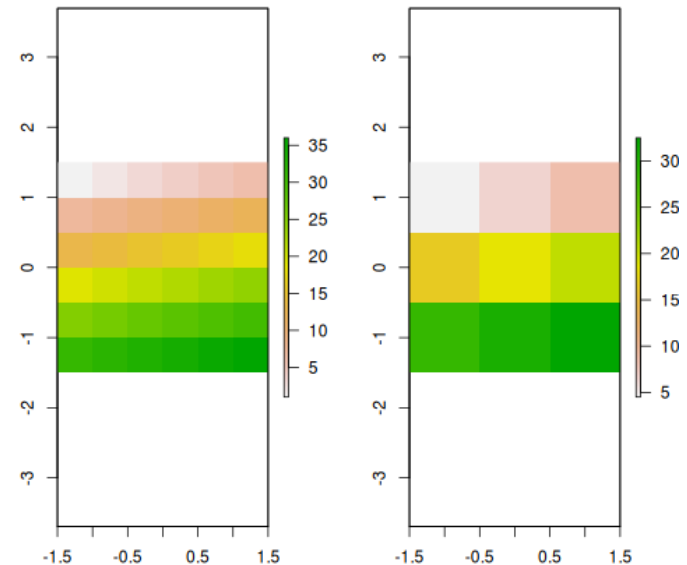


# Aggregation and disaggregation

Change the resolution of a raster:

```
elev_agg =  
  aggregate(elev, fact = 2,  
            fun = mean)
```

Use `dissaggregate()` for  
increasing the spatial resolution of a  
raster







# Changing the CRS of a raster

- To change the CRS of a raster use `projectRaster()`.
- EPSG codes are not accepted, use a proj4string instead.



# Changing the CRS of a raster

- To change the CRS of a raster use `projectRaster()`.
- EPSG codes are not accepted, use a proj4string instead.

```
library(spDataLarge)  
proj4string(nz_elev)
```

```
## [1] "+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000
```



# Changing the CRS of a raster

- To change the CRS of a raster use `projectRaster()`.
- EPSG codes are not accepted, use a proj4string instead.

```
library(spDataLarge)
proj4string(nz_elev)
```

```
## [1] "+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000
```

```
projectRaster(nz_elev, crs = st_crs(4326)$proj4string)
```

```
## class      : RasterLayer
## dimensions : 1483, 1248, 1850784  (nrow, ncol, ncell)
## resolution : 0.0119, 0.00901  (x, y)
## extent     : 164.9573, 179.8085, -47.53651, -34.17468  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : elevation
## values     : 0, 3195.91  (min, max)
```



# Changing the CRS of a raster

- To change the CRS of a raster use `projectRaster()`.
- EPSG codes are not accepted, use a proj4string instead.

```
library(spDataLarge)
proj4string(nz_elev)
```

```
## [1] "+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000
```

```
projectRaster(nz_elev, crs = st_crs(4326)$proj4string)
```

```
## class      : RasterLayer
## dimensions : 1483, 1248, 1850784  (nrow, ncol, ncell)
## resolution : 0.0119, 0.00901  (x, y)
## extent     : 164.9573, 179.8085, -47.53651, -34.17468  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : elevation
## values     : 0, 3195.91  (min, max)
```

Further reading on geometric raster operations:

<https://geocompr.robinlovelace.net/geometric-operations.html#geo-ras>



# Your turn

- Decrease the resolution of dem (`data("dem", package = "RQGIS")`) by a factor of 10. Plot the result.
- Reproject dem into WGS84. Plot the output next to the original object.
- Randomly select three points of `random_points` (`data("random_points", package = "RQGIS")`). Convert these into a polygon (hint: `st_cast`). Extract all altitudinal values falling inside the created polygon Use the polygon to clip dem. What is the difference between `intersect` and `mask`. Hint: `sf` objects might not work as well with **raster** commands as `SpatialObjects`. Assuming your polygon of class `sf` is named `poly`, convert it into a `SpatialObject` with `as(sf_object, "Spatial")`.



# Recap

We learned about:

- raster attribute operations
- spatial raster operations
- geometric raster operations