Plotting and Spatial data

**Some plotting functions:**

plot(): Generic plotting, works for a variety of different object types

hist(): Plot a histogram

abline(): Draw a line on the plot with specified slope and intercept

points(), segments(), arrows), polygons():Draw points, line segments, arrows or polygons on an existing plot.

par(): Change everything about how a plot is drawn. Run it before running the plot function; the settings will apply to all plots until a new window is started. Here are the par() settings I typically use.

par(font.axis = 6,font.lab = 6,font.main = 8,las=1,cex.axis = 1.2,cex.lab = 1.6,cex.main = 2,mar = c(5,5,2,1),lend = 2)

This sets the font sizes for the axis, labels and title, causes the y-axis to be labeled with un-rotated numbers (las), slightly enlarges axis numbers (by a factor of 1.2), enlarges axis labels (by a factor of 1.6) and enlarges the main title (by a factor of 2). Then, I reduce the plot margins somewhat and set all lines to end with square ends (rather than rounded ends).

**Some spatial functions**

readShapePoly(): Read a .shp and associated files, creating a SpatialPolygonsDataFrame

raster(): Create a raster object, by reading from a file or by defining it in R

click(): Brings up the active plot window, and returns the raster value at where you click.

zoom(): Click twice to select the new zoomed region

pointDistance(): Calculates the distance between two points (or two sets of points).

extract(): Extracts raster values or calculates statistics for a point or polygon.

cellFromXY(), cellFromRowCol(), colFromX(), xFromCol() etc.: Functions for converting among row/column, cell and x/y formats for identifying cells

project(), projectExtent(), projectRaster(), spTransform(): Functions for geographical projections

**Required packages:**

maptools

rgdal

raster

Part 1: Plotting

**1**. Create two correlated variables x and y, using x = rnorm(200) and y = x + rnorm(200).

a. Plot a histogram of x. Change the color of the bars (using the col argument) and the color of the borders (using the border argument)

b. Use abline() to plot the mean of x as a vertical, dotted line

c. Plot x vs. y, and play around with the plotting parameters. Use different settings in par(), change the axis labels and plot title, and change the plotting characters.

d. Use par(mfrow = c(2,3)) to create a grid of 2 rows and 3 columns to plot into. Plot x and y into these windows using 6 plot commands with different colors and plotting symbols.

Part 2: Basics of shapes and rasters

**2**. Read a spatial polygon map of the world (Political Map.shp). To figure out how things in the shape file are named, use names(), and try using str() to look at the structure. It is kind of a mess, but can still be useful. Try plotting the world map using plot(). Change the colors of the countries and borders using col and border.

Make a vector with the names of all countries. How many countries are there? Plot a world map where each country has a unique color. Which entry in the list is Denmark? Make a new spatial polygon object that is just the polygons for Denmark. Plot the map of Denmark, and add some lines of longitude and latitude using abline().

**3**. Use raster() to read in the MAT.tif raster, a map of mean annual temperature for some of northern Europe. Look at the object you created for the raster – note that it gives only summary information and no actual data. This is because R does not load a raster into memory when you read it

To get any of those particular pieces of summary information, generally just use the name of the line as the function (for example ncol(yourRaster) or ymin(yourRaster) )

**4**. Plot the raster, using plot(yourRaster). Give it different color schemes, using the rgb() function, the rainbow() function and divPalette() in the package fBasics.

**5**. The coordinates of Aarhus are 10.216 E, 56.150 N. Plot this as a point on the MAT map. Find the value of the MAT raster in Aarhus using three methods

a. Zoom in to the area and use the click() function to click on Aarhus

b. Determine which cell Aarhus is in (using cellFromXY()) and use that cell to index the raster

c. Determine which row and column of the raster Aarhus is in using rowFromY() and colFromX(), and use that row and column to index the raster.

**6**. extract() can get all raster values from within a polygon, or calculate some function of these within the polygon. Use it and your polygon map of Denmark from question 1 to calculate the mean MAT for all cells within Denmark.

Part 3: Projections

**7.** For this part, we will work on projecting the polygon of Denmark you extracted in question 1 into a new coordinate system.

a. The coordinate system of the polygon is currently undefined. You can see that by passing your polygon object to the function projection(). Define it by using projection(your\_object) = CRS(“+proj=longlat +datum=WGS84”)

b. Now use spTransform() to project the polygon into the Behrmann equal area projection (“+proj=cea +lat\_ts=30”)

c. Finally, get the coordinates for the corners of the Jutland portion of the Denmark polygon into a 2-column matrix (this will require you to dig through the structure of the object!). Jutland is the 8th polygon in the object. Project those x,y coordinates into the Behrmann projection using project() from the rgdal package and check that they match the transformed polygon you made in b.

**8**. Read in the “Global MAP (0.5 degree).tif” raster, which is a map of mean annual precipitation from worldclim, aggregated to 0.5 degree resolution.

a. Project it into a Behrmann Equal Area projection (cylindrical equal area, latitude of true scale = 30) using projectRaster()

b. Project it by projecting the x,y coordinates of a Behrmann grid back into latitude and longitude coordinates and sampling the original raster at these coordinates.

c. Use Sys.time()to determine which method is faster.

d. Do the two methods produce identical results? If not, why?

e. Your Behrmann raster probably has a resolution of 48202.75 m. Verify that this is the width of a cell at 30°N in a lat-long grid with resolution of 0.5° (use pointDistance()). (Note – the pointDistance function calculates the distance along the great circle, not along the line of latitude. Thus, it will produce a (very) slightly different distance.)