SISTEM INFORMASI GEOGRAFIS

PENGENALAN R SOFTWARE

JULLEND GATC

Brief Intro

- R is a powerful language and environment for statistical computing and graphics.
- It is a public domain (a so called GNU") project
- was developed at Bell Laboratories.
- 100% free vs MATLAB tidak free
- Lebih user-friendly dari pada C++ or Fortran.
- Home page: http://www.r-project.org/

Brief Intro

- **Need to be installed**
  - R-2.15.2-win.exe → core program
  - RStudio-0.97.316.exe → IDE
- **Package**
  - sp
  - maps
  - maptools
  - gstat
  - dll
- **For complete package just do**
  - `> install.views("Spatial")`

Warming Up

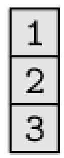
- **Set Working Dictionary**
  - `> setwd("H:/Data Spatial/Materi Praktikum/workspace/")`
- **Install Package**
  - `> install.packages("H:/Data Spatial/sp_1.0-5.zip", repos = NULL, type="source")`
  - `> library(sp)`
- **Calculator**
  - `> 10^2 + 36`
  - `> a = 4`
  - `> a`
  - `> a * 5`
  - `> a = a + 1`
  - `rm(list=ls()) #remove variable`
  - `j = NA`
- `#some comment`

Data structures

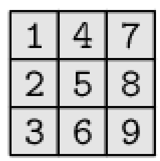
- **Scalar – single number**
  - `> B = 3`
- **Vectors – array 1D**
  - `> b=c(3,4,5)`
  - `> b`
  - `> b[2]`
  - `> b[3] = 12`
  - `> vec2 = seq(from=0, to=1, length=5)`
  - `> c(1,4,6,8,10)+ vec2`
- **Matrices – 2D**
  - `> mat=matrix(data=c(9,2,3,4,5,6),ncol=3)`
- **Data Frame**
  - `> t = data.frame(x = c(11,12,14), y = c(19,20,21), z = c(10,9,7))`
  - `> t$z`
- **Lists**
  - `> L = list(one=1, two=c(1,2), five=seq(1, 4, length=5))`
  - `> L`

Data structures

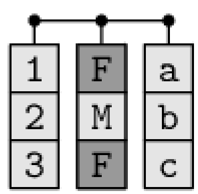
- Vector



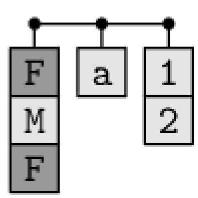
- Matric



- Data Frame



- List



- **Reading and Writing**
  - `> d = data.frame(a = c(3,4,5),  b = c(12,43,54))`
  - `> write.table(d, file="tst0.txt",  row.names = FALSE)`
  - `> d2 = read.table(file="tst0.txt", header=TRUE)`
- **Characters**
  - `> m = "apples"`
- **Dates**
  - `> date1=strptime( c("20100225230000", "20100226000000", "20100226010000"), format="%Y%m%d%H%M%S")`
- **Function**
  - `> b=c(3,4,5)`
  - `> mean(x=b)`
  - `> rnorm(10)`
- **Help and Documantation**
  - `> help(rnorm)`
  - `> example(rnorm)`

Programming tools

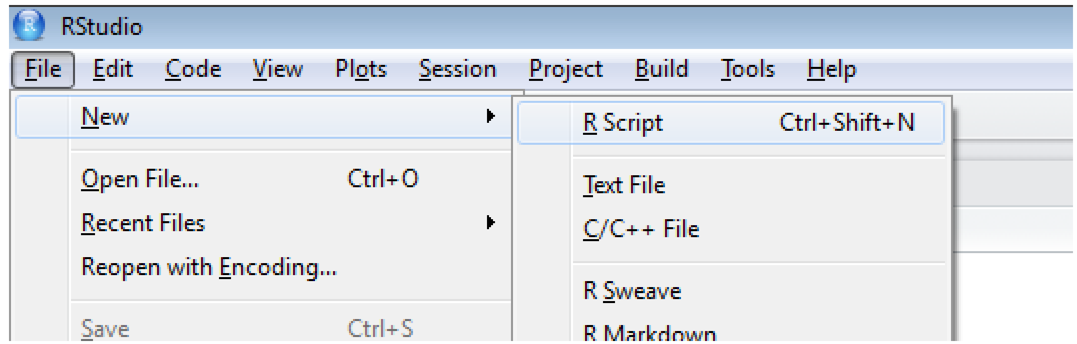
- **If-statement**
  - `> w = 3`
  - `> if( w < 5 ) {d=2}else{d=10}`
- **For-Loop**
  - `> h = seq(from = 1, to = 8)`
  - `> for(i in 1:8)`
  - `{   h[i] = h[i] * 10   }`
- **While**
  - `> z <- 0`
  - `> while(z < 5) {`

```
    z <- z + 2
    print(z)   }
```

- **Repeat**
  - `> z <- 0`
  - `> repeat {`

```
    z <- z + 1
    print(z)
    if(z > 100) break()   }
```

- **Script**
  - File - New - R Script (Ctrl + Shift + N)
    - Ketikan
    - `> A = c(1,2,3)`
    - `> B = c(3,4,5)`



  - `> source("foo.R")`

Function

- **Built in Function**
  - `> m <- mean(c(1,2,3,4,5))`
  - `> l <- length(c(1,2,3,4,5))`
  - lapply() - When you want to apply a function to each element of a list in turn and get a list back.
  - sapply() - When you want to apply a function to each element of a list in turn, but you want a vector back, rather than a list.
  - `> x <- list(a = 1, b = 1:3, c = 10:100)`
  - `> lapply (a, length)`
  - `> sapply(a,length)`
- **Write Functions**
  - `> fun1 = function(arg1, arg2 )`
  - `{`
  - `w = arg1 ^ 2`
  - `return(arg2 + w)`
  - `}`
  - `> fun1(arg1 = 3, arg2 = 5)`

Function

- **Some Common Function**
  - `cbind( )` and `rbind( )` : Combine R Objects by Rows or Columns
    - `> cbind(c(1,2),c(4,5))`
  - `a1 <- as.character(99/8)`
  - Functions to convert between character to calendar dates and times.
    - `z <- strptime("1jan1960", "%d%b%Y")`
    - `as.POSIXlt(z, "GMT")`
  - Sorting data; order() return the index of sorted data
    - `o <- order(c(4,3,2,5,1,1,16,6))`
  - `str(a)` display the internal *str*ucture of an R object; max.level maximal level of nesting
  - `t()` : transpose
  - `identical()` : To `wholly' compare two vecors
  - `nrow()` : Returns number of rows
  - `match()` : matching 2 vector; ex: match(c(1,2,3),c(4,5,6,7,3,4,4,4,1,5,4,3,6))
  - `row.names(x)` : Get and Set Row Names for Data Frames

- > x <- rnorm(50,2,3)
- > plot(1:50,cumsum(x)) #plot Cumulative Sum
- > curve(x^3 - 3*x, -2, 2)
- > x <- scan("scandata.txt")
- > write(x,"scandata.txt")

Class

- A class is a static entity written as program code designed to represent objects of a certain type using slots (which in turn have other classes etc.)
- Can be S3 (old-style) or S4
- To create an S3 method write a function with the name *generic_function*.**class**,
  - Don't be confused!!! `state.pupolation` can be a variable
  - `> state.population <- c(1,2,3,4,5)`
- Information in S4 classes is organized into slots. Each slot is named and requires a specified class.

S3 Class example

- **# Create a class object infant**
- ```
  > infant2 <- function(ID, sex, age, ht, wt) {
    out <- list(ID=ID, sex=sex, data=data.frame(Age=age,
  HT.cm=ht, WT.kg=wt))
    class(out) <- "infant2"
    invisible(out)
  }
  ```
- **# Print method for infant class**
- ```
  > print.infant2 <- function(object) {
    cat("hahahahID =", object$ID, "\nSex =", object$sex, "\n")
    print(object$data)
  }
  ```
- **# create an object of infant2**
- `> x <- infant2(1, "male", age, male.ht, male.wt)`
- `> print(x)`

S3 Class example

- Methods to construct S3 Classes

`class(x)`	Get or set the class attributes of `x`
`unclass(x)`	Remove class attributes of `x`
`methods(generic.function)`	All available S3 methods for a generic function
`methods(class="`*class*`")`	Get all S3 methods for a particular class
`is(object)`	Return `object`'s class and all super-classes
`is(object, class2)`	Tests if `object` is from `class2`
`str(object)`	Display the internal structure of an `object`

S4 Class Example

 # Define an infant class

- `> setClass("infant", representation(ID = "numeric", sex = "character", data = "data.frame"))`
- `> getClass("infant")`
- `> getSlots("infant")`

 # Create an object of class infant

- `> x <- new("infant", data=data.frame(age, male.wt, male.ht), sex="male", ID=1)`
- `> x@data` **# Use the @ sign to extract a specific slot**

 # Show method, similar to print for S3 classes

- > setMethod(f = "show", signature = "infant",

```
        definition = function(object) {
         cat("ID =", object@ID, "\nSex =", object@sex, "\n")
          print(object@data)
        })
```

S4 Class Example

- Methods to construct S4 Classes

`setClass()`	Create a new class
`setMethod()`	Create a new method
`setGeneric()`	Create a new generic function
`new()`	Generate a new object for a given class
`getClass()`	Get the class definition
`getMethod()`	Get the method definition
`getSlots()`	Get the name and class of each slot
`@`	Get or replace the contents of a slot
`validObject()`	Test the validity of an object

Classes for Spatial Data in R

- Using **sp** package
- Point

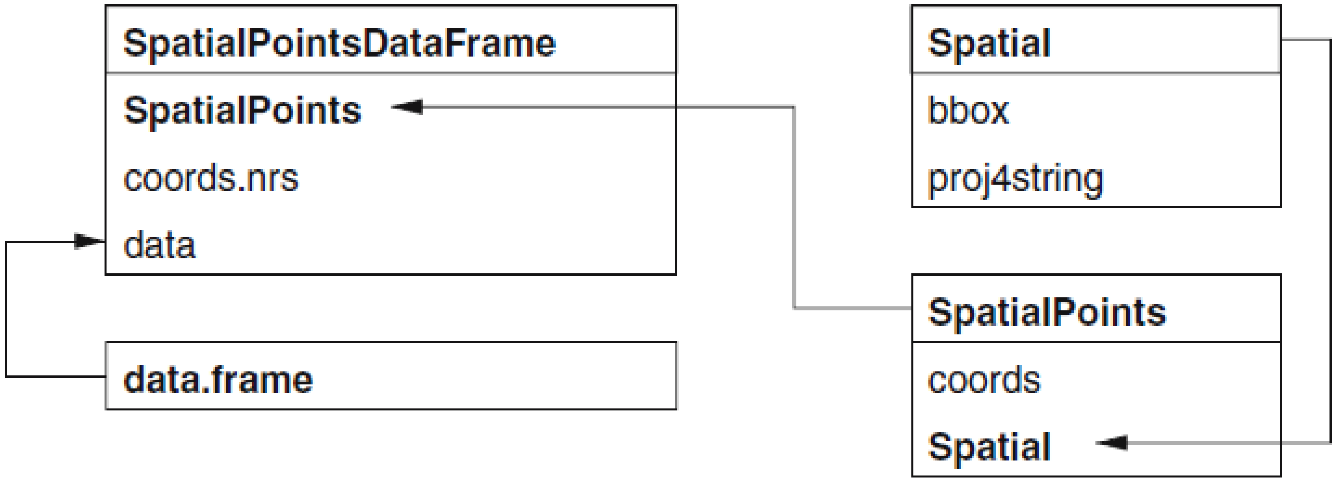


Fig. 2.2. Spatial points classes and their slots; arrows show subclass extensions

Spatial Class

- **Consists of 2 slots**
  - bbox → matrix coordinate of area study
  - crs → Coordinate Reference System

- `> library(sp)`
- `> getClass("Spatial")`

- `> m <- matrix(c(0, 0, 1, 1), ncol = 2, dimnames = list(NULL,c("min", "max")))`
- `> crs <- CRS(projargs = as.character(NA))`
- `> S <- Spatial(bbox = m, proj4string = crs)`

Spatial Point Class

- `> CRAN_df <- read.table("CRAN051001a.txt", header = TRUE)`
- `> CRAN_mat <- cbind(CRAN_df$long, CRAN_df$lat)`
- `> row.names(CRAN_mat) <- 1:nrow(CRAN_mat)`

- `> llCRS <- CRS("+proj=longlat +ellps=WGS84")`
- `> CRAN_sp <- SpatialPoints(CRAN_mat, proj4string = llCRS)`
- `> summary(CRAN_sp)`

Additional Information

- CRS : WGS84 (A new World Geodetic System: WGS 84)

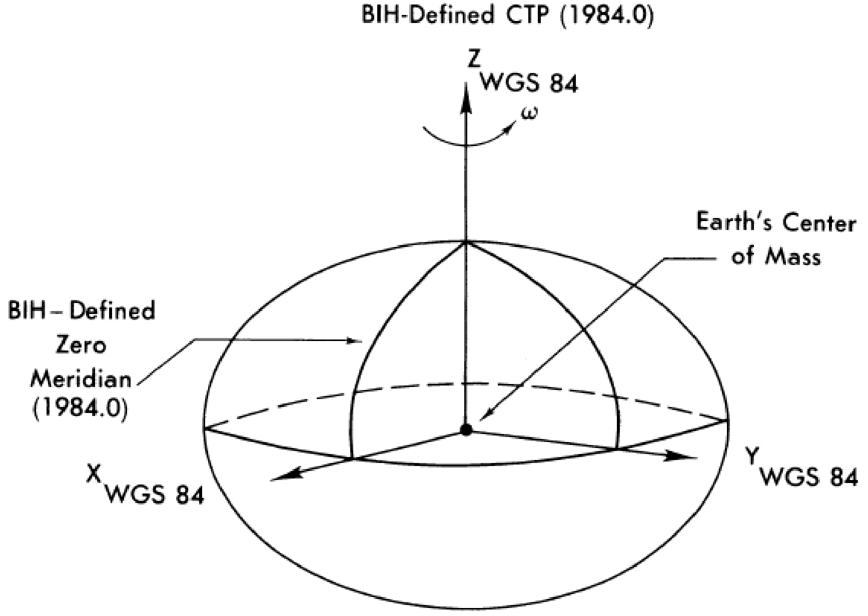


Figure 1.1. WGS 84 Reference Frame

Method in Spatial Points

- Get bounding box
  - `> bbox(CRAN_sp)`
- Get projection string
  - `> proj4string(CRAN_sp)`
- Get a specific location
  - `> brazil <- which(CRAN_df$loc == "Brazil")`
  - `> coordinates(CRAN_sp)[brazil, ]`
- Access spatial point with specific index
  - `> summary(CRAN_sp[brazil, ])`
  - `> south_of_equator <-which(coordinates(CRAN_sp)[,2]<0)`
  - `> summary(CRAN_sp[-south_of_equator, ])`

Spatial Points Data Frame

- SpatialPointsDataFrame(coords, data, coords.nrs = numeric(0),
  proj4string = CRS(as.character(NA)), match.ID = TRUE)

- `> CRAN_spdf1 <- SpatialPointsDataFrame(CRAN_mat, CRAN_df, proj4string = llCRS, match.ID = TRUE)`
- `> CRAN_spdf1[4, ]`

Spatial Points Data Frame

- `turtle_df <- read.csv("seamap105_mod.csv")`
- `summary(turtle_df)`
- `timestamp <- as.POSIXlt(strptime(as.character(turtle_df$obs_date), "%m/%d/%Y %H:%M:%S"), "GMT")`
- `turtle_df1 <- data.frame(turtle_df, timestamp = timestamp)`
- `turtle_df1$lon <- ifelse(turtle_df1$lon < 0, turtle_df1$lon + 360, turtle_df1$lon)`
- `turtle_sp <- turtle_df1[order(turtle_df1$timestamp),]`
- `coordinates(turtle_sp) <- c("lon", "lat")`
- `proj4string(turtle_sp) <- CRS("+proj=longlat +ellps=WGS84")`

Lines

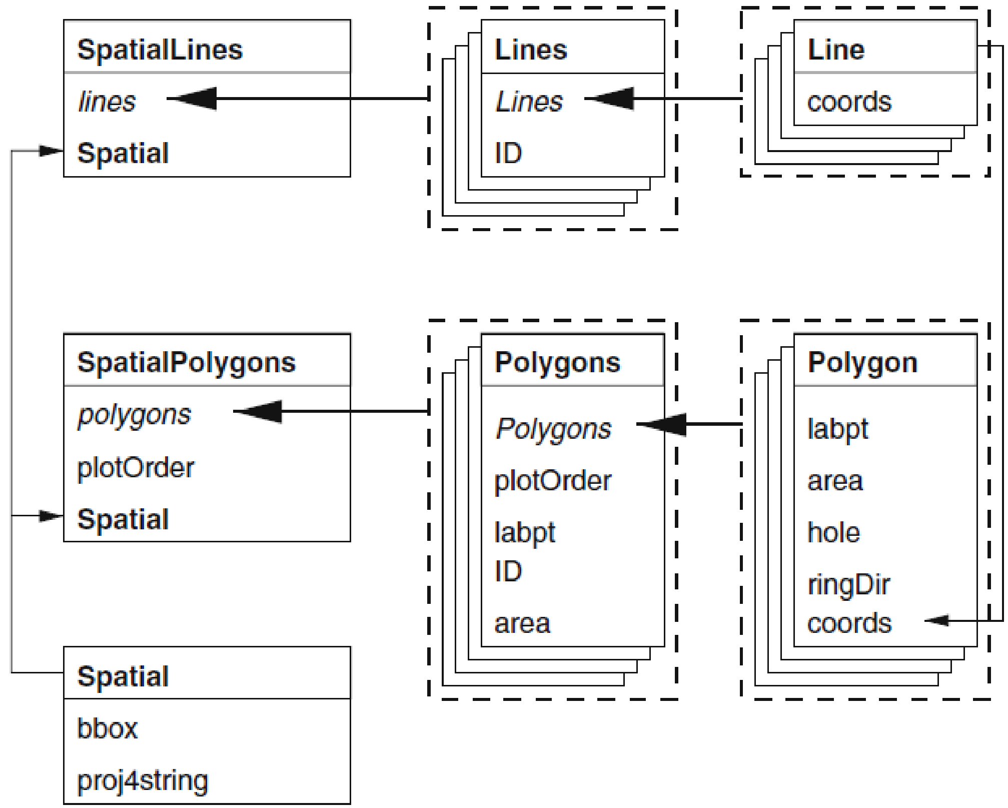


Fig. 2.4. SpatialLines and SpatialPolygons classes and slots; thin arrows show subclass extensions, thick arrows the inclusion of lists of objects

Spatial Lines

- `> library(maps)`
- `> library(maptools)`
- `> japan <- map("world", "japan", plot = FALSE)`
- `> p4s <- CRS("+proj=longlat +ellps=WGS84")`
- `> SLjapan <- map2SpatialLines(japan, proj4string = p4s)`
- `> str(SLjapan, max.level = 2)`
- `> Lines_len <- sapply(slot(SLjapan, "lines"),function(x) length(slot(x, "Lines")))`
- `> table(Lines_len)`

Spatial Line Data Frame

- `volcano_c <- contourLines(volcano)` #Calculate contour lines for a given set of data
- `volcano` : default variable in R; This data set gives topographic information for Maunga Whau on a 10m by 10m grid.
- `> volcano_sl <- ContourLines2SLDF(volcano_c)`
- `> t(slot(volcano_sl, "data"))`

Spatial Poligons

- Closed line
- `> getClass("Polygon")`
- `> getClass("Polygons")`
- `> getClass("SpatialPolygons")`

- `auckland_mapgen.dat` : Auckland Shoreline data set in Mapgen format.
- **# Import Mapgen to Spatial lines**
  - `> llCRS <- CRS("+proj=longlat +ellps=WGS84")`
  - `> auck_shore <- MapGen2SL("auckland_mapgen.dat", llCRS)`
  - `> summary(auck_shore)`

- Get the lines
  - `> lns <- slot(auck_shore, "lines")`
  - `> table(sapply(lns, function(x) length(slot(x, "Lines"))))`
- Check is it an island? The start point and end point should be the same.
  - ```
    > islands_auck <- sapply(lns, function(x) {
        crds <- slot(slot(x, "Lines")[[1]], "coords")
        identical(crds[1, ], crds[nrow(crds), ])
    })
    ```
  - `#p = slot(lns[10][[1]],'Lines')`
  - `> table(islands_auck)`
    ```

- > islands_sl <- auck_shore[islands_auck]
- > list_of_Lines <- slot(islands_sl, "lines")
- > islands_sp <- SpatialPolygons(lapply(list_of_Lines, function(x) {

```
  Polygons(list(Polygon(slot(slot(x, "Lines")[[1]],
                   "coords"))), ID = slot(x, "ID"))
          })
    , proj4string = CRS("+proj=longlat +ellps=WGS84"))
```

- > summary(islands_sp)

Spatial Polygons Data Frame

- `> state.map <- map("state", plot = FALSE, fill = TRUE)`
- `> IDs <- sapply(strsplit(state.map$names, ":"), function(x) x[1])`
- `> state.sp <- map2SpatialPolygons(state.map, IDs = IDs,`
- `proj4string = CRS("+proj=longlat +ellps=WGS84"))`
- `> sat <- read.table("state.sat.data_mod.txt", row.names=5, header=TRUE)`
- `> id <- match(row.names(sat), sapply(slot(state.sp, "polygons"),`
- `function(x) slot(x, "ID")))`
- `> row.names(sat)[is.na(id)]`
- `> state.spdf <- SpatialPolygonsDataFrame(state.sp, sat[!is.na(id),])`
- `> str(slot(state.spdf, "data"))`

Spatial Grid and Spatial Pixel

- This representation is typical for remote sensing and raster GIS,
- used widely for storing data in regular rectangular cells,
  - such as digital elevation models, satellite imagery, and interpolated data from point measurements, as well as image processing

- `> getClass("GridTopology")`
- Data Manitoulin Island vector data set.
  - `> load("high.RData")`
  - `> manitoulin_sp <- high$SP`

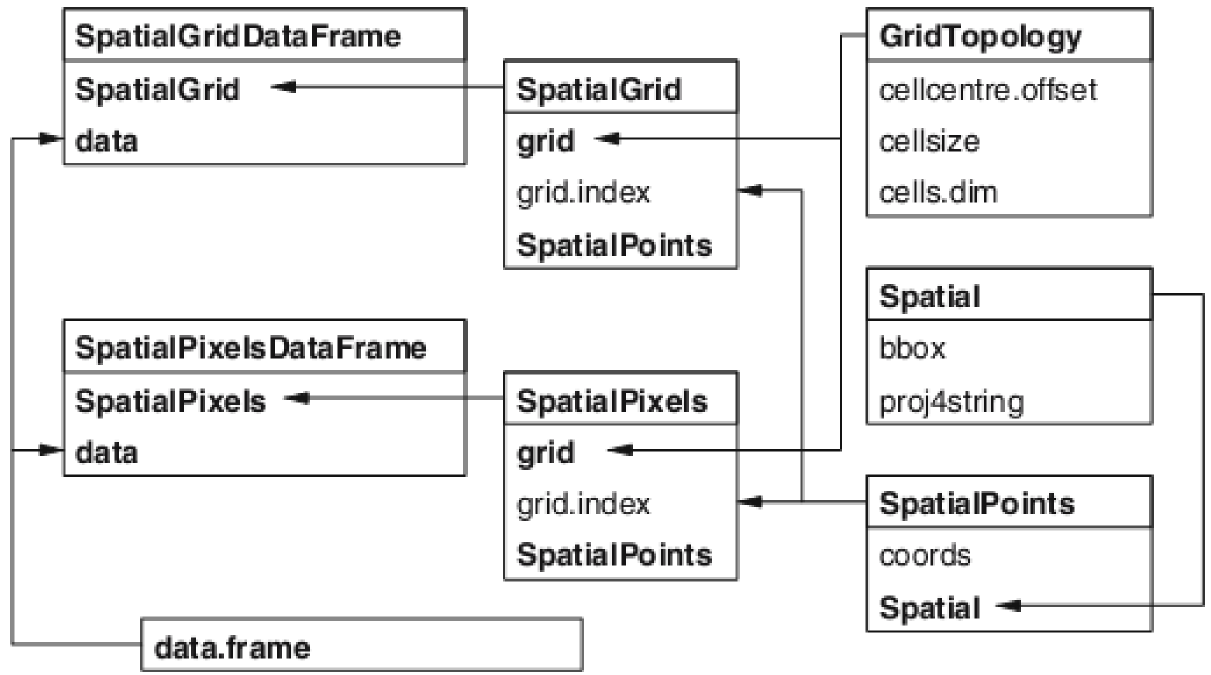


Fig. 2.8. `SpatialGrid` and `SpatialPixel` classes and their slots; arrows show subclass extensions

Spatial Grid

- `> bb <- bbox(manitoulin_sp) #get bounding box`
- `> bb`
- `> cs <- c(0.01, 0.01) # cell size`
- `> cc <- bb[,1]+(cs/2) # smallest coordinate`
- `> cd <- ceiling(diff(t(bb))/cs) #dimention`
- `> manitoulin_grd <- GridTopology(cellcentre.offset=cc, cellsize=cs, cells.dim=cd)`
- `> manitoulin_grd`

- `> getClass("SpatialGrid")`

- `> p4s <- CRS(proj4string(manitoulin_sp))`
- `> manitoulin_SG <- SpatialGrid(manitoulin_grd, proj4string=p4s)`
- `> summary(manitoulin_SG)`

Spatial Grid Data Frame

- Data
  - SRTM elevation data in metres for the Auckland
  - `> fname <- unzip(zipfile = "70042108.zip", files="70042108.tif")`
  - `> library(rgdal)`
  - `> auck_el1 <- readGDAL("70042108.tif")`
  - `> class(auck_el1)`
  - `> slot(auck_el1, "grid")`
  - `> slot(auck_el1, "grid.index")`
  - `> slot(auck_el1, "bbox")`
  - `> object.size(auck_el1)`
  - `> object.size(slot(auck_el1, "data"))`

SpatialPixelsDataFrame

- we can coerce our SpatialGridDataFrame object to a SpatialPixelsDataFrame object
  - `> auck_el2 <- as(auck_el1, "SpatialPixelsDataFrame")`
  - `> object.size(auck_el2)`
  - `> object.size(slot(auck_el2, "data"))`

Plotting Points, Lines, Polygons, and Grids

- R has 2 plotting systems:
  - the ‘traditional’ plotting system
  - the Trellis Graphics system (**lattice**)

Traditional Plotting

- **Points,**
  - `> library(sp)`
  - `> data(meuse)`
  - `> coordinates(meuse) <- c("x", "y")`
  - `> plot(meuse)`
  - `> title("points")`
- **Lines,**
  - `> cc <- coordinates(meuse)`
  - `> m.sl <-SpatialLines(list(Lines(list(Line(cc)), "1")))`
  - `> plot(m.sl)`
  - `> title("lines")`

Traditional Plotting

- **Polygons,**
  - `> data(meuse.riv) #Meuse River`
  - `> meuse.lst <- list(Polygons(list(Polygon(meuse.riv)), "meuse.riv"))`
  - `> meuse.sr <- SpatialPolygons(meuse.lst)`
  - `> plot(meuse.sr, col = "grey")`
  - `> title("polygons")`
- **and Grids**
  - `> data(meuse.grid)`
  - `> coordinates(meuse.grid) <- c("x", "y")`
  - `> meuse.grid <- as(meuse.grid, "SpatialPixels")`
  - `> image(meuse.grid, col = "grey")`
  - `> title("grid")`
- **Combination**
  - `> image(meuse.grid, col = "lightgrey")`
  - `> plot(meuse.sr, col = "grey", add = TRUE)`
  - `> plot(meuse, add = TRUE)`

Traditional Plotting

- Axes
  - `> layout(matrix(c(1, 2), 1, 2))`
  - `> plot(meuse.sr, axes = TRUE)`
  - `> plot(meuse.sr, axes = FALSE)`
  - `> axis(1, at = c(178000 + 0:2 * 2000), cex.axis = 0.7)`
  - `> axis(2, at = c(326000 + 0:3 * 4000), cex.axis = 0.7)`
  - `> box()`
- plotting a scale bar and a north arrow
  - `> plot(meuse)`
  - `> plot(meuse.sr, col = "lightgrey", add = TRUE)`
  - `> SpatialPolygonsRescale(layout.scale.bar(), offset = c(180200,329600), scale = 1000, fill=c("transparent","black"), plot.grid = FALSE)`
  - `> text(x = c(180200,181200), y = rep(329750, 2), c("0", "1 km"))`
  - `> SpatialPolygonsRescale(layout.north.arrow(), offset = c(178750,332500), scale = 400, plot.grid = FALSE)`
  - `> box()`

- Degrees in Axis
  - > library(maptools)
  - > library(maps)
  - > wrld <- map("world", interior = FALSE, xlim = c(-179, 179), ylim = c(-89, 89), plot = FALSE)
  - > wrld_p <- pruneMap(wrld, xlim = c(-179, 179))
  - > llCRS <- CRS("+proj=longlat +ellps=WGS84")
  - > wrld_sp <- map2SpatialLines(wrld_p, proj4string = llCRS)
  - > prj_new <- CRS("+proj=moll")
  - > library(rgdal)
  - > wrld_proj <- spTransform(wrld_sp, prj_new)
  - > wrld_grd <- gridlines(wrld_sp, easts = c(-179, seq(-150, 150, 50), 179.5), norths = seq(-75, 75, 15), ndiscr = 100)
  - > wrld_grd_proj <- spTransform(wrld_grd, prj_new)
  - > at_sp <- gridat(wrld_sp, easts = 0, norths = seq(-75, 75, 15), offset = 0.3)
  - > at_proj <- spTransform(at_sp, prj_new)
  - > plot(wrld_proj, col = "grey60")
  - > plot(wrld_grd_proj, add = TRUE, lty = 3, col = "grey70")
  - > text(coordinates(at_proj),pos = at_proj$pos, offset = at_proj$offset, labels = parse(text = as.character(at_proj$labels)), cex = 0.6)

Traditional Plotting

- Plotting Attribute and Map Lagend
  - `> data(meuse.grid)`
  - `> coordinates(meuse.grid) <- c("x", "y")`
  - `> gridded(meuse.grid) <- TRUE`
  - `> library(gstat)`
  - `> zn.idw <- krige(log(zinc) ~ 1, meuse, meuse.grid)`
  - `> cols <- grey.colors(4, 0.95, 0.55, 2.2)`
  - `> image(zn.idw, col = cols, breaks=log(c(100,200,400,800,1800)))`
  - `> plot(meuse.sr, add = TRUE)`
  - `> plot(meuse, pch = 1, cex = sqrt(meuse$zinc)/20, add = TRUE)`
  - `> legVals <- c(100, 200, 500, 1000, 2000)`
  - `> legend("left", legend=legVals, pch = 1, pt.cex = sqrt(legVals)/20, bty = "n",`
  - `> title="measured, ppm", cex=0.8, y.inter=0.9)`
  - `> legend("topleft", fill = cols, legend=c("100-200","200-400","400-800“,"800-1800"), bty = "n", title = "interpolated, ppm", cex=0.8, y.inter=0.9)`
  - `> title("measured and interpolated zinc")`

Trellis/Lattice Plots with ssplot

- **A Straight Trellis Example**
  - `> library(gstat)`
  - `> library(sp)`
  - `> data(meuse)`
  - `> coordinates(meuse) <- ~x+y #sets spatial coordinates;c("x","y")`
  - `> data(meuse.grid)`
  - `> coordinates(meuse.grid) <- ~x+y`
  - `> gridded(meuse.grid) <- T`
  - `> zn <- krige(zinc~1,meuse,meuse.grid)`
  - `> zn$direct <- zn$var1.pred`
  - `> zn$log <- exp(krige(log(zinc)~1,meuse,meuse.grid)$var1.pred)`
  - `> library(lattice)`
  - `> print(levelplot(z~x+y|name, spmap.to.lev(zn[c("direct", "log")]), asp = "iso", cuts=4, col.regions=grey.colors(5, 0.90, 0.50, 2.2)), split = c(1,1,1,2), more = TRUE)`
  - `> print(spplot(zn[c("direct", "log")], cuts=4, col.regions=grey.colors(5, 0.90, 0.50, 2.2)), split = c(1,2,1,2))`

Plotting Point

- Compare the absolute level in ppm
- `> cuts=c(0,20,50,200,500,2000)`
- `> grys <- grey.colors(7, 0.90, 0.50, 2.2)`
- `> print(spplot(meuse[1:4], main = "ppm", cuts=cuts, cex=.5, col.regions=grys), split=c(1,1,2,1),more=T)`
- `> meuse$lead.st = as.vector(scale(meuse$lead))`
- `> meuse$zinc.st = as.vector(scale(meuse$zinc))`
- `> meuse$copper.st = as.vector(scale(meuse$copper))`
- `> meuse$cadmium.st = as.vector(scale(meuse$cadmium))`
- `> cuts=c(-1.2,0,1,2,3,5)`
- `> print(spplot(meuse, c("lead.st", "zinc.st", "cadmium.st", "copper.st"), main = "standardised", cex = .5, cuts = cuts, col.regions=grys), split=c(2,1,2,1))`

Plotting Countur

- Countur Lines
  - > data(meuse.grid)
  - > coordinates(meuse.grid) <- c("x", "y")
  - > meuse.grid <- as(meuse.grid, "SpatialPixelsDataFrame")
  - > cl = ContourLines2SLDF(contourLines(as.image.SpatialGridDataFrame(meuse.grid["dist"])))
  - > grys <-grey.colors(nlevels(cl$level), 0.90, 0.50, 2.2)
  - > print(spplot(cl, colorkey=list(height=0.8, width=0.6), col.regions=grys), split = c(1,1,2,2), more=TRUE)
  - > cuts = (0:10)/10
  - > grys <- grey.colors(length(cuts), 0.90, 0.50, 2.2)
  - > print(spplot(meuse.grid, "dist", colorkey=list(labels=list(at=cuts), at=cuts), col.regions=grys, pretty = TRUE), split = c(2,1,2,2), more = TRUE)
  - > meuse.grid$f = factor(meuse.grid$ffreq, labels = c("annual", "2-5 yrs", "> 5 yrs"))
  - > print(spplot(meuse.grid, "f", colorkey=list(height=0.4, width=0.6), col.regions=grey.colors(3, 0.90, 0.50, 2.2)), split = c(1,2,2,2), more=FALSE)

- **Adding Reference and Layout Elements to Plots**
  - `> river <- list("sp.polygons", meuse.sr)`
  - `> north <- list("SpatialPolygonsRescale", layout.north.arrow(), offset = c(178750,332500), scale = 400)`
  - `> scale <- list("SpatialPolygonsRescale", layout.scale.bar(), offset = c(180200,329800), scale = 1000, fill=c("transparent","black"))`
  - `> txt1 <- list("sp.text", c(180200, 329950), "0")`
  - `> txt2 <- list("sp.text", c(181200, 329950), "1 km")`
  - `> pts <- list("sp.points", meuse, pch = 3, col = "black")`
  - `> meuse.layout <- list(river, north, scale, txt1, txt2, pts)`
  - `> grys <- grey.colors(7, 0.90, 0.50, 2.2)`
  - `> print(spplot(zn["log"], sp.layout = meuse.layout, cuts=6, col.regions=grys))`

- Color Palettes and Class Intervals
  - `> library(RColorBrewer)`
  - `> library(classInt)`
  - `> pal <- grey.colors(4, 0.95, 0.55, 2.2)`
  - `> q5 <- classIntervals(meuse$zinc, n=5, style="quantile")`
  - `> q5Colours <- findColours(q5, pal)`
  - `> plot(meuse, col=q5Colours, pch=19)`
  - `> points(meuse, pch=1)`
  - `> box()`
  - `> title(main="Quantile")`
  - `> legend("topleft", fill=attr(q5Colours, "palette"), legend=names(attr(q5Colours, "table")), bty="n", cex=0.8, y.intersp=0.8)`