R introduction (mainly based on Fox's chapters I and II)

- R on the internet www.r-project.org
- R's broad support basis (see R foundation member list)
- R's constant update service
- Differences between S-plus and R and other statistical software packages
- Advantages:
  - Free, powerful and frequently updated
  - Very broad support basis
  - Open source: quality control and replicable results are possible
  - *De facto* standard in the academic statistical community. Novel methods are first implemented as prototype in R and later may disseminate into commercial packages.
  - There are many books on R and statistical data analysis with R that are published in many different languages.
    Some are even freely downloadable (see http://cran.r-project.org/other-docs.html)
- Disadvantages:
  - Steep learning curve because R is mostly command line driven.
  - Basic documentation mainly summarizes the individual functions and their interaction needs to be studies by looking at provided sample code
  - Except for the Rcmdr no menu interface
  - There are a few outdated and poorly developed packages (We won't touch those)

Regression Analysis and Spatial analysis libraries for the R environment

The functionality of a basic installation of R is already quite powerful. However, specialized libraries (perhaps with sample datasets) have been developed to extend this functionality. These libraries usually must be downloaded and attached to your R session (see more in the InstallRInclVista.pdf document).
Our main library for doing regression analysis are John Fox's

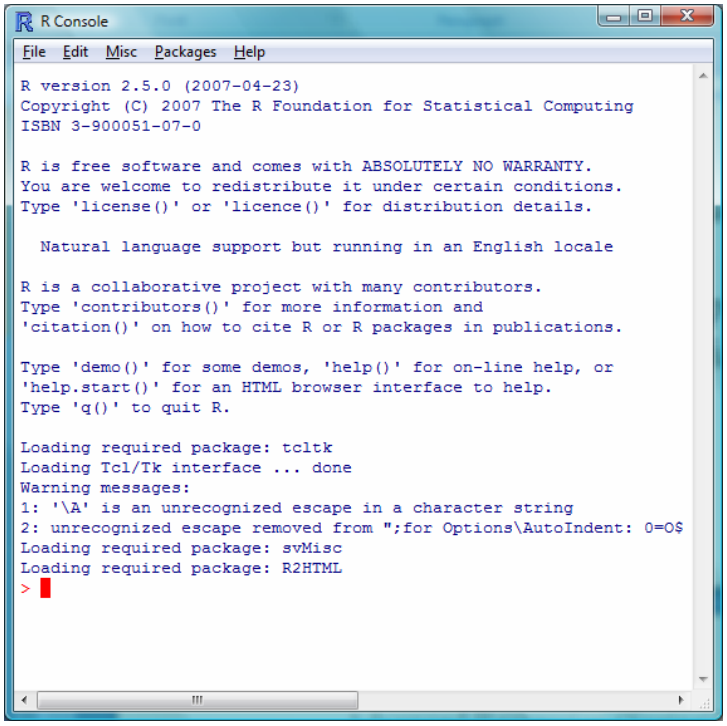
- car library, which is well documented in "An R and S-Plus Companion to Applied Regression"
- Most functions of the car library are also directly accessible through the Rcmdr interface of John Fox.
- John Fox's R companion website is http://socserv.mcmaster.ca/jfox/Books/Companion/index.html
- For some analyses we will use Venables & Ripley's MASS library

Subset of relevant R-packages that allow us to perform spatial regression analysis:

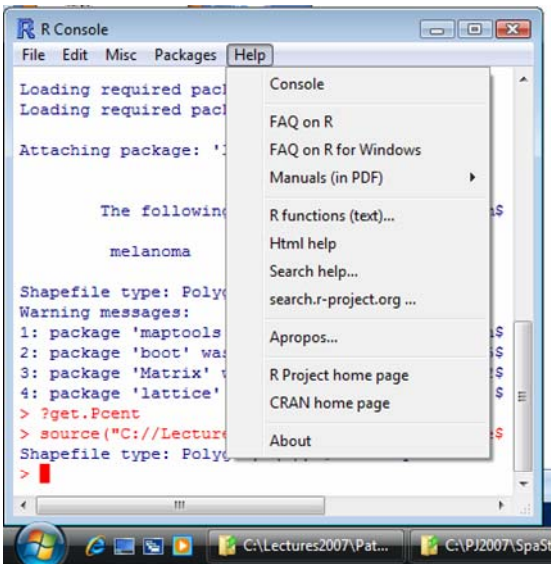
- Spatial Statistics:
  - spdep (autoregressive spatial regression and tests)
- Rudimentary mapping
  - maptools (mapping shape files)
- General spatial analysis overview:
  - http://cran.r-project.org/src/contrib/Views/Spatial.html
  - http://www.sal.uiuc.edu/tools/tools-sum/rgeo/rgeo-detail/geostatistics-packages-on-cran

The R Environment

- Explain the R-GUI



- Concept of workspaces, history file and scripts. The associate extentions are
  - *.RData: Copy of the workspace with all its variables and custom user functions
  - *.Rhistory contains all command that have been issued during a session at the command prompt >
  - *.R is a file that contains scripts, which can be a set of basic R data analysis commands, individual functions, or an elaborate program
- Change the *working directory* were scripts, workspaces are store and searched by default.
- Receiving help:
  - > help(*FunctionName*) or
    > ?*FunctionName* or
    online html help documents
  - Thru the help menu:



Basic R mechanics

- The ">" prompt indicates that R is ready to receive commands
- Single commands can be run from the command line
  Collections of commands (or programs) can be stored in external *.R files and run as scripts with the command

> source("*C:\\SpaReg\\MyFile.R*")

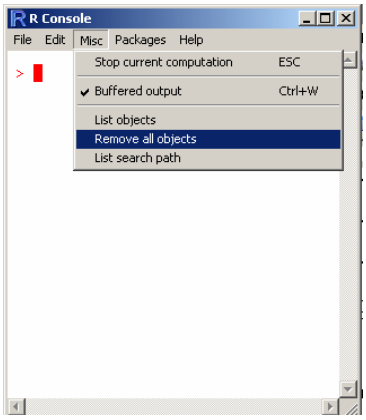
Notes:

- In R the backslash "\" is reserved as escape character, it thus must be substituted by the double backslash "\\" to separated component of a file path
- All characters and strings must be surrounded by quotation marks, e.g., "…"

- The arrow UP and DOWN keys navigate thru the history of previously issued R commands
- Previously issued commands can be edited only by using the arrow keys to position the prompt
- All R commands are implemented as functions with parentheses, even if they do not have specific parameters, e.g., the function ls(), which lists all variables and functions in a workspace
  > ls()
- Commands can be broken over several lines. The continuing prompt "+" will then be displayed automatically. Example:
  > ls(
  +)
- Assignment operator to a variable is <-
  > my.pi <- 3.14 (compare to > pi)
- Naming variables is alpha/numeric ("$" and "@" cannot be used)
  Tips: Name variables properly so you and an external reader of your R session can retrieve later what you were doing
  Use the dot to structure your variable names, e.g., *poly.X* and *poly.Y*
  R is case sensitive, e.g., *My.Var* and *my.var* are different (This has a high potential for typos)
  Warning: if you name a variable identically to an existing function in R that function will no longer be accessible:
  > exp <- 453.5
- Any function, data structure or variable that are defined during an R-session become objects in the workspace unless they are removed

from the workspace with the function
> rm(*My.Var*)

- Clean up by removing all objects from the R workspace:



- Key values:
  - Logical values are T and F (or TRUE and FALSE)
  - Unassigned variables have the value NULL
  - Missing numbers have the value NA
  - Infinity is Inf
- R only has a minimal editor for commands and function definitions. External editors such as TinR can be added (see more in the InstallRInclVista.pdf document)
- Also the data spread sheet is not very powerful. See function fix()
  > fix(*Columbus*).
- The best way of saving R-output and figures is to copy and past them into a graphically enhanced text editor such as Word
- All variables and functions in a workspace can be save for subsequent sessions (see menu File ► Save Workspace …)

Some R-functions:

- *Exercise:*
  > x <- seq(0.1 ,2,by=0.1)
  > x
  > y <- log(x)
  > plot(x,y)
  > help(plot)

- In plot menu select *History -> Recording* to keep more than one graphics in the plot memory.
  ```
  > plot(x,y,type="l")
  ```
- Sample functions: ls(), c(…), cbind(…), seq(…), summary(…), dim(…), length(…)
- *Exercise:*
  ```
  > z <- rnorm(length(x))
  > mat <- cbind(x,y,z)
  > dim(mat)
  > summary(mat)
  ```

Some Data Structures

List objects

- List objects allow to link data objects of *different types* and *different length* together into a list container:
  ```
  > my.cats <- c("Henry","Lily","Charlie")
  > age.cats <- c(5,15,4)
  > A <- matrix(rep(4,6),2,3)
  > my.list <- list(catname = my.cats, age = age.cats, mat = A)
  > my.list
  $catname
  [1] "Henry"   "Lily"    "Charlie"

  $age
  [1]  5 15  4

  $mat
       [,1] [,2] [,3]
  [1,]    4    4    4
  [2,]    4    4    4
  ```
- Individual objects of the list can be addressed either by
  ```
  > my.list$catname
  [1] "Henry"   "Lily"    "Charlie"
  > my.list[[1]]
  [1] "Henry"   "Lily"    "Charlie"
  ```
- Individual elements of an object in a list can be addressed by

```
> my.list$catname[1]
[1] "Henry"
```

- To delete an object in a list assign it the NULL value by issuing the command:

```
> my.list$mat <- NULL
```

- To get information about an object use the attributes function:

```
> attributes(my.list)
$names
[1] "catname" "age"
```

- Remove the object my.list

```
> rm(my.list,A)
```

- R-functions can only return one data object. Thus if a function is supposed to return several related results of different data-types (such as a numerical matrix and a vector of names), then these results are pooled together into a list object.

Data-frames

- Data-frames can pool several vectors of *same length* but potentially of *different data-types* together.

  It is important that these vectors are of the same length.

```
> my.data <- data.frame(my.cats,age.cats)
> my.data
  my.cats age.cats
1   Henry      5
2    Lily       15
3   Charlie     4
```

- The character variable my.cats and the numeric variable age.cats are stored now in the data frame my.data
- The list function returns the objects that are presently in use

```
> ls()
> “my.cats”   “age.cats”   “my.data”
```

- To remove objects from an R session use the remove function

```
> rm(my.cats,age.cats)
> ls()
> “my.data”
```

- The variables my.cats and age.cats are still stored as variables in the data frame my.data
- To access the variables in the data frame several commands can be used
  > my.data$my.cats
  > my.data[1]
- Alternatively, a data frames can be attached to the *search path* so that the individual variables within a data-frame can be directly accessed
  > attach(my.data)
  > age.cats
  [1] 5 15 4
- To remove a dataframe again from the *search path* issue the command
  > detach(my.data)
  [The variables in my.data are no longer directly accessible]

Indexing elements in a vector or matrix:

- Matrices can store vectors of *same length* and *same data-type* in an rectangular arrangement
- To generate a matrix:
  - List of elements:
    > b <- c(10,20,30,40,15,25,35,45,1,2,3,4)
  - Place elements into 4x3 matrix:
    mat <- matrix(b,nrow=4,ncol=3)
- One element at location row and col mat[row,col]
  A sequence of values mat[1:2,] (here the first and second row)
  Exclusion of elements mat[-1,] (here the first row)
- One row mat[1,] or one column mat[,2]
- Also logical operations are permitted (here the first and second columns are displayed):
  > select <- c(T,T,F)
  > mat[,select]

Logical operation

- Vectorized logical operators:
  - == *[logical equal]*

  - & *[logical "and"]*
  - | *[logical "or"]*
- Compare vectors elementwise:

```
> x <- 1:3 # gives 1,2,3
> x.logical <- x==x
> x.logical
> xx <- x
> xx[1] <- 99
> xx == x
```

Working with the Concord1 dataset:

- Download the Concord1.sav data set and the script ScatterPlotMat.R from WebCT and store in a folder of your choice (the suggested destination folder is *C:\ SpaReg*)
- Reading an external file:

```
> library(foreign)
> Concord <- read.spss("C:\\ SpaReg\\Concord1.sav",
+ use.value.labels=TRUE, to.data.frame=TRUE)
> summary(Concord)    [notice the different data types]
> fix(Concord)
> Concord$WATER79    [notice the missing values]
> attach(Concord)
> WATER79
```

- Adding a variable to a data frame:

```
> Concord$LnWater81 <- log(WATER81)
> colnames(Concord)
```

- Removing a variable:

```
> Concord$LnWater81 <- NULL
```

Generate a scatter plot matrix:

- Discuss scatterplot script ScatterPlotMat.R

```
> pairs(cbind(INCOME,WATER81,WATER80),
+    panel=function(x,y){
```

```
+      points(x,y)
+      abline(lm(y~x), lty=2)
+      lines(lowess(x,y))
+      },
+   diag.panel=function(x){
+      par(new=T)
+      hist(x,main="", axes=F, nclass=12)
+      }
+   )
```

- Run the script:
  > source("*C:*\\ *SpaReg* \\ScatterPlotMat.R")

Perform a basic regression analysis:

> attach(Concord)
> Concord.lm <- lm(WATER81 ~ INCOME + PEOP81)
> summary(Concrod.lm) *[Inspect regression results]*
> hist(rstudent(Concord.lm), nclass=12) *[Any outliers?]*
> library(car) *[Needed for function qq.plot]*
[Identify outlier cases in interactive plot]
> qq.plot(Concord.lm,labels=row.names(Concord), simulate=T)
[Stop selection by right-click with mouse]

- Updating the model without outlier observations 85, 124, and 125
  > summary(update(Concord.lm,subset = -c(85,124 ,125)))
  [Compare estimated regression parameters]

Exercise:

- Download the file Columbus.dbf from WebCT.
- Load the dbf-file Columbus.dbf by using the function read.dbf in the library foreign
- Explore the dataset
- Run a regression model with CRIME as dependent variable and DISCBD as independent variable.