Introduction and set up

R is a programming language designed specifically for statistical computing and graphics. It is available as a free software at **http://www.r-project.org/**

- ➔ **If working on a personal computer**, your first task consists in downloading the appropriate package. Go to the CRAN section on the left hand side of the R webpage and select a mirror from which to download (e.g. University of Bristol), then select your platform (Windows, Mac or Linux) and choose the base version. The latest release of R will be available for download (currently 2.9.2). Note that R is constantly updated and it is therefore recommended to frequently check for newer versions.
- ➔ Follow install instructions and launch R.
- ➔ **If working from the computer lab you do not need to install R,** just launch the program.

When opening R, a console window will appear (see screen shot at the end of this document). This represents the R environment and everything you type in this console will be interpreted by R **as long as it follows the R language convention**. Here, we will introduce a few commands to manipulate data and generate graphics but this is just a very limited aspect of the potential of R. A complete description of the language is available from the R-project website under the Manual section. In particular the document called "An introduction to R" provides a very good starting point. A reference card summarizing the main commands you are most likely to use is also provided in this tutorial's directory. You can also consult the reference manual at any time in R by typing: ?help or ? followed by a specific command name (e.g.?plot).

Tutorial number 1: plotting solar radiation data

Although you could type all R commands directly in the console this is not recommended since it does not enable you to save your work. The consol should only be used to test some commands or do quick calculations. In this tutorial (and for all your future projects) you are advised to open a script and save it as you progress.

- ➔ Open a new script (file/new script). A blank window will appear. This is the script in which you should type all following commands. To save your work just click on the script window and select file/save. To ask R to interpret the command you typed just "source" the script (go back to the console window, then go to file/source R code and select your saved script).
- ➔ We recommend starting every script by cleaning up the R environment. This will avoid conflict between variable defined from a previous project and still stored in the R memory. This is done using the following syntax:

```
rm(list=ls())
```

This is a good example showing that commands are not always self explanatory. To make your script more readable it is a good habit to comment what the script is doing as you go. This can be done using the special character # which tells R not to interpret the rest of the line. In this particular case a good way to start you script would be:

```
# R tutorial number 1, 6 Oct 2009
# start with a clean work space
rm(list=ls())
```

Having cleaned the environment we can now load new data. The file "Radiation_2009181.txt" located in the tutorial directory contains radiation data collected from the WXT510 on the 30th of June 2009 (DOY 181). It is an ascii file and contains 6 columns: Decimal time[1], hour, incoming solar, outgoing solar, incoming longwave and outgoing longwave radiation.

➔ Open the file with a text editor and check the data. You will notice that the first two rows are reserved for header information (variable names and units). The best way to load this type of data into R is to store it in a table or array. R is excellent at reading in data with very little information on the file format but in this case we need to specify that the first two lines are to be skipped. You will therefore add the following commands to your script:

```
# read data from radiation file and store into a table
# note that the file has 2 lines of header which we skip
TABLE <- read.table("C:/R_tutorial/Radiation_2009181.txt",skip=2)
```

You should of course adapt the path to your directory structure. Remember that if you have any doubt on the command syntax just type ?read.table in the console window.

This will have stored all the data into the array called TABLE. Before to go on with the tutorial it is a good step to check if everything is correct so far.

➔ Save your script (file/save) and then source it into the R environment (click on the console, then file/source R code and browse you script). If all the commands are correctly understood by R they will appear in red in the console, if not the execution will stop with blue lines indicating error. To check that the radiation data were correctly read in you can ask R to print the values in TABLE. To do so just type TABLE in the console window. R should print 96 lines of data, each column corresponding to one of the variables in the file "Radiation_2009181.txt".

➔ To clarify your script and since we are not working with all the variables in this array you can copy only the needed column in separate vectors:

```
# store only the columns we need
DECTIME<-TABLE[,1]
KDOWN<-TABLE[,3]
KUP<-TABLE[,4]
```

This will copy column number 1 into a vector called DECTIME, column number 3 into the vector KDOWN and column number 4 into the vector KUP. You can then study your data and ask R to print things like the mean, max or minimum values:

```
# get minimum, mean and maximum values
print(min(KDOWN))
print(mean(KDOWN))
print(max(KDOWN))
```

What do you think about the minimum value of the incoming solar radiation? Do you have any explanation for such value? How about the maximum (compare for instance to the solar constant)?

[1] Decimal time - this is just the time as a decimal, so 12:30 on day 181 is for instance represented as 181+12.5/24 = 181.5208

➔ Although a quick print of simple statistics can help you spot potential errors, the best way to analyze it is by plotting its evolution. We will now plot both the incoming solar radiation as a function of the decimal time.

```
# plot KDOWN as a function of DECTIME
plot(DECTIME,KDOWN,type="b",
xlab="Time (d)",ylab="Solar radiation (W m-2)")
```

The plot command possesses a wide range of optional fields and just a few are here used. Type ?plot in the console window or check the reference card for a full description of all options. Note also from this example that you do not need to type all the command on the same line. For long commands it is preferable to spread over several lines. The type="b" attribute here tells R to plot the data with both lines and symbols, alternatives are type ="l" for lines only and type="p" (the default) for points only.

You can now save your script and source it again. Alternatively you can also copy the last three lines (highlight them and ctrl-C) and paste them into the console window. A third window called R graphics should now appear with the desired plot. Does this profile of incoming solar radiation seem reasonable to you? How can you explain the dip between 181.3 and 181.4?

➔ We will now add the outgoing solar radiation **to the same plot**. If you use the plot command again it will overwrite the existing one. The alternative is to use one of the so called "low-level commands" (see reference card) like points or lines:

```
# add KUP to the same plot
lines(DECTIME,KUP,type="b",pch=2,col="red")
```

Note that this time we specify an additional option pch=2 to force R to use a different symbol (see the reference card for a complete list of symbols). We also specify that the line should be plot in red.

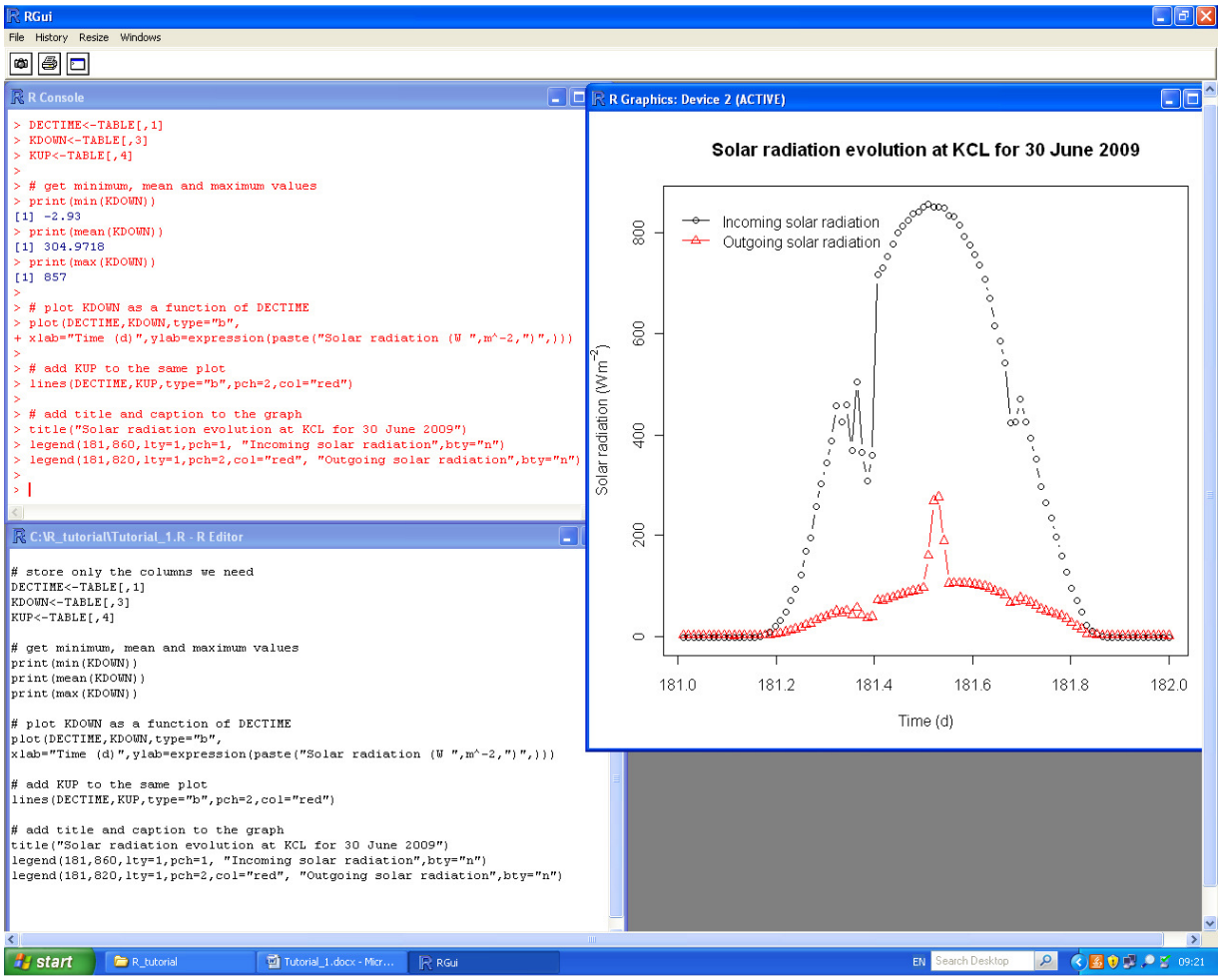
Copy & pasting the previous line into the window console should be enough to get the updated plot, but you can also save and source the script if you prefer. Does this profile of outgoing solar radiation seem reasonable to you? Would you have an explanation for the peak around mid-day?

➔ To finalize the plot, title and legends need to be added. This is done via the following low-level plotting commands:

```
# add title and caption to the graph
title("Solar radiation evolution at KCL for 30 June 2009")
legend(181,860,lty=1,pch=1, "Incoming solar radiation",bty="n")
legend(181,820,lty=1,pch=2,col="red", "Outgoing solar
radiation",bty="n")
```

The title command will simply add the text you specify to the top of the plot. The legend command will add the specified text at the x and y location provided (e.g. x=181 and y=860 for the 1st example).Note that these are in the units of the variables plotted on the graph. lty and pch options work in the same way as in plot and bty="n" prevents R from drawing a box around the legend.

Either from a copy & paste of the last few lines or by saving and sourcing the whole script you should obtain the same plot as below:



Note that the units in the y-axis label are not represented correctly. To force R to interpret mathematical notations the command “expression” is needed, and it can only be mixed with normal text using the paste command which concatenates strings. To get a correct y-label you should therefore replace the plot command by:

```
plot(DECTIME,KDOWN,type="b",xlab="Decimal time",
ylab=expression(paste("Solar radiation (W ",m^-2,")",)))
```

The script corresponding to this tutorial is provided in the same directory (Tutorial_1.R). If you encounter any problem please refer to this script to spot your errors. If you want to directly source this script remember to change the path to the data file in the plot command.

Any suggestions or revisions to this document please email: **thomas.loridan@kcl.ac.uk**