APPM2720 Week 6 Lecture: More about reading data and working with images

This lecture gives some typical examples of reading in data and reformatting for analysis in R. To make this more relevant the focus is on image data. To work through these examples you will need to install the **fields**, **raster**, and **jpeg** R packages (in Rstudio *tools -> Install packages*)

Overview

As way to organize thoughts on this topic think about the image format as being most useful for representing 2-d surfaces and other data on a 2-d grid. The raster format is better for photos and other images where the main use is to overlay them on top of other information. Image format is the one for computing and statistics and raster format is a more attractive and faster plotting mode for viewing images. You can translate between the two forms but this is usually not necessary since they have different purposes.

About image formats in R

- An *image matrix* in R has two subscripts that index the pixels of the image. All images are assumed to be rectangular although sometimes the pixel sizes can be unequal in the X and Y directions. For the data set `lennon` has values that are assumed to be the shades of grey on a scale from 0 to 255. Where 0 is black and 255 is white.

In the image format:

- lennon[1,1] is the value in the bottom left
- lennon[256,1] the bottom right
- lennon[1, 256] the top left
- lennon[256,256] top right

```
library( fields)
data(lennon)
dim( lennon)
range( c( lennon))
```

The dimensions are 256X256 and the range is actually [0,192]. To find the values in particular values you can use subscripting: lennon[1:128, 1:128] would give you just a quarter of the image.

- An extension of an image matrix includes two arrays that indicate the scale of the rows and columns. This is not as useful for say a photo. For images where the pixels are grid of longitudes and latitudes this is invaluable especially when a map needs to be added on top of the image.

  The image format is a list with components **x**, **y** and **z**. For the lennon image the list would be

```
lennon2 <- list( x= 1:256, y=1:256,z=lennon)
```

here the z component is the 256X256 image matrix that was considered above. Using this as `image(lennon2)` will automatically scale the axis according to x and y.

The way to remember where everything goes is the pixel z[i,j] will be located at the point x[i], y[j] in the image plot. So the pixel lennon[23,45] will be at the point 23, 45 in the plot.

Reshaping data as matrices and arrays

Images are great examples for reading in an array of values and getting in the right format for visualizing. The file **lennonRaw.txt** is a dump of the lennon image values, column by column but written 10 values to a line.

Here are some steps to read it in.

```
tempData<- scan( "lennonRaw.txt")
test1<- matrix( tempData, ncol=256, nrow=256)
# or
test1<- array( tempData, c( 256, 256))
```

- Switch the roles of the rows and columns. The image will be flipped around the diagonal. This is the most common issue in reading in images.

```
test2<- aperm( test1, c(2,1))
```

See the R script **readImages.R** for other examples of modifying the image, e.g. rotating 90 degrees and flipping left to right or top to bottom.

Using the image and image.plot functions

To plot this image we use the image function. This function assumes a default color scale that is good for data analysis but not good for black and white photos. A better choice is a to 256 grey values ranging form black (0) to white (1.0). The code below gives a sequence on more elaborate modifications to the image plot. Note the last one gives a useful legned to indicate how the image values are mapped to the grey levels.

```
data(lennon)
image(lennon) # default colors
greyScale<-  grey( seq(0,1,,256))
image(lennon , col= greyScale)
# change the coordinates to be in "pixels"
image(1:256, 1:256,lennon , col= greyScale)
# add a color scale to indcate the grey levels
image.plot(1:256, 1:256,lennon, col= greyScale, xlab="", ylab="")
```

Here is a slightly different version that makes it mode like a photo. The zlim argument means map the grey scale to the range 0 to 255. Since the actual image only goes up to 192 we see that even the brightest parts are still sort of grey.

```
# make this more like a photo
par( mar=c(1,1,1,1)) # small margins
image(1:256, 1:256,lennon, col= greyScale, axes=FALSE,
 xlab="", ylab="", zlim =c(0,255) )
```

The advantage of bringing an image into R is that you still do data analysis and statistics on the numerical values and pixel locations.

```
# add a horiontal lines at pixel 100 and 200
image.plot(1:256, 1:256,lennon, col= greyScale, xlab="", ylab="")
abline( h=c(100,200), col="magenta")
```

plot a subset of the image

```
testImage<- lennon[1:200,1:100]
image(1:200, 1:100, testImage,
 col= greyScale)
```

Use the zlim to force this to use the same range of grey levels as the full image.

You can always turn pixel values to white by making them NAs
`testImage<- lennon testImage[ lennon< 10] <- NA`

or do other other arithmetic with the image matrix. For example to emphasize the brighter parts of the image you could plot the square root of the image values (e.g. `testImage <- lennon^(.5)`).

Reading in color images

The file **RlogoRaw.txt** is an 76X100 image that has three color channels (actually 3 images). To read this in

```
tempData<- scan( "RlogoRaw.txt")
test1<- array( tempData, c(76,100,3))
```

Each of `test1[,,1] , test1[,,2], test1[,,3]` is a separate 76X100 image matrix and represent the color channels red , green and blue (RGB).

The file **RlogoRaw.jpg** is the original jpeg image for the Rlog and can be read in R by `library(jpeg) test2<- readJPEG("Rlogo.jpg")` The objects **test1** and **test2** are the same! To plot these in R you will need to use the raster package to combine the color channels into the actual color at each pixel. (see next section)

To read in an image in the the png format: `library(png) test3<- readPNG("Rlogo.png")` Because the png format is different than the jpeg **test2** and **test3** are slightly different.

Working with raster format and the raster package

The raster package is designed for images that are akin to photos and less for manipuating them as matrices. Plotting raster images is easy.

```
tempData<- scan( "RlogoRaw.txt")
test1<- array( tempData/255, c(76,100,3))
testRaster<- as.raster( test1)
plot(testRaster)
```

plot in this case exploits the raster format and makes an image instead of the default scatterplot.

The image plots much faster but the "values" in testRaster are now characters indicating the color directly. So you can not do arithmetic on them.

In the raster format:

- testRaster[1,1] is the value in the top left
- testRaster[256,1] the bottom left
- testRaster[1, 256] the top left
- testRaster[256,256] bottom right

If you want to manipulate an image you should read it in as the array of 3 image matrices, modify these, then convert to the raster format to plot.

Working with RGB colors

The file `Rlogo.jpg` is a JPEG image of the R logo. As mentioned above when this is read into R the result is three different image matrices indicating the mix of red, green and blue colors at each pixel. The three different images are the red, blue and green channels that define the colors. Every color in R can be described by the levels of RBG. The levels are between [0,1] or [0, 255] depending on the function and defaults. Grey is defined by the RGB levels being equal (e.g. c(.5,.5,.5) is a medium grey. The **as.raster** function combines these numerical values into a color code that is a character string. The **rgb** function in R can also be used to do this. For example to convert RGB 255, 62, 150. we have

```
rgb( 255, 62,150, maxColorValue=255)
[1] "#FF3E96"
```

Technically "#FF3E96" is a character that indicates 3, two digit numbers in hexadecimal (base 16).

The important concept however, is the raster format represents the assembled color for each pixel using these color codes and this is why one can not manipulate the image when is it is the raster format.

Aside about color names.

Another way to describe colors is by their name see **colors()** for a listing of colors in R. You can use **col2rgb*** to convert and a color name to the rgb code and of course this is done internally by R whenever a color is specified in this way in a plot function. These kinds of names do not make sense in the raster format, however, because we are expecting the kind of arbitrary colors that would be present in a color photo.

```
col2rgb( "violetred1")
      [,1]
red    255
green   62
blue   150
```