General R (and Writing Images)

Reading in the image

Again we will read in the `training01_01_mprage.nii.gz` file, and assign it to an object called `t1`:

```
library(neurobase)
t1 = readnii("training01_01_mprage.nii.gz")
```

Data Classes

- Numeric - numbers (e.g. 1, 3.673)
- Character - strings or words (`"hey"`, `"I'm a string"`) in either single or double quotes
- Logicals - `TRUE` or `FALSE` - all capital letters and are **not** in quotes.

Data Types

- `vector` - 1-dimensional object of one class (all numeric or all character)
- `matrix` - 2-dimensional object of one class
- `data.frame` - 2-dimensional object, can be multiple classes (like Excel spreadsheet)
- `array` - object of dimensions > 2 of one class. The data in a `nifti` object is one of these (usually 3-D)
- `nifti` - an array with header information
- `list` - a general holder of things (discuss when necessary)

Vectors

- Create a vector of numeric values and assign to variable `v`

```
v = c(1, 4, 3, 7, 8)
```

- Subsetting (first index is 1, not zero):

```
print(v[4])
```

```
[1] 7
```

```
print(v[1:3])
```

```
[1] 1 4 3
```

```
print(v[c(1,3,5)])
```

```
[1] 1 3 8
```

`:` creates a sequence of numbers

Matrices

- Create a 3 x 4 numeric matrix and assign to variable `m`

```
m = matrix(1:12, nrow = 3)
```

- Subsetting - `[row,column]` format, if `row` or `column` missing then all values:

```
print(m[,4])
```

```
[1] 10 11 12
```

```
print(m[2,])
```

```
[1]  2  5  8 11
```

```
print(m[1,3])
```

```
[1] 7
```

Subsetting with logicals

You can either do subsetting with indices or logical vectors:

```
v[ v > 5 ]
```

```
[1] 7 8
```

the `which` command takes a logical and gets the indices:

```
which(v > 5)
```

```
[1] 4 5
```

```
v[ which(v > 5) ]
```

```
[1] 7 8
```

Working with `nifti` objects

The subsetting here is similar to that of arrays, so we will use the `t1`. Since it's 3-dimensions the subsetting goes to the 3rd dimension

```r
t1[5, 4, 3]
```

```
[1] 0
```

```r
t1[5, 4, ] # returns a vector of numbers (1-d)
t1[, 4, ] # returns a 2-d matrix
t1[1, , ] # returns a 2-d matrix
```

Working with `nifti` objects

Again, we can use a logical operation. Let's get the values of the `t1` greater than 400 (`head` only prints the first 6 values):

```
head(t1[ t1 > 400 ])
```

```
[1] 421 617 617 479 456 404
```

Working with `nifti` objects

The operation results in a `nifti` object, and if we look at the values, they are logical:

```r
class(t1 > 400)
```

```
[1] "nifti"
attr(,"package")
[1] "oro.nifti"
```

```r
head(t1 > 400)
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

`which` with `nifti` objects

The `which` function works to get indices, but you can pass the `arr.ind = TRUE` argument to get "array" indices:

```r
head(which(t1 > 400, arr.ind = TRUE))
```

```
     dim1 dim2 dim3
[1,]   98  129    1
[2,]   99  129    1
[3,]  100  129    1
[4,]  163  129    1
[5,]  164  129    1
[6,]  190  129    1
```

Working with `nifti` objects

Again, we can use a logical operation. Let's get the values of the `t1` greater than 400 (`head` only prints the first 6 values):

```
head(t1[ t1 > 400 ])
```

```
[1] 421 617 617 479 456 404
```

Working with `nifti` objects: reassignment

Subsetting can work on the left hand side of assignment too:

```
t1_copy = t1
t1_copy[ t1_copy > 400] = 400 # changed these values!
max(t1_copy) # should be 400
```

```
[1] 400
```

```
max(t1)
```

```
[1] 1505
```

Note, although `t1_copy` was copied from `t1`, they are not linked - if you change values in `t1_copy`, values in `t1` are unchanged.

Writing Images out

We now can write out this modified `t1_copy` image:

```
writenii(nim = t1_copy,
         filename = "training01_mprage_under400.nii.gz")
file.exists("training01_mprage_under400.nii.gz")
```

```
[1] TRUE
```

Vectorizing a `nifti`

To convert a `nifti` to a `vector`, you can simply use the `c()` function:

```
vals = c(t1)
class(vals)
```

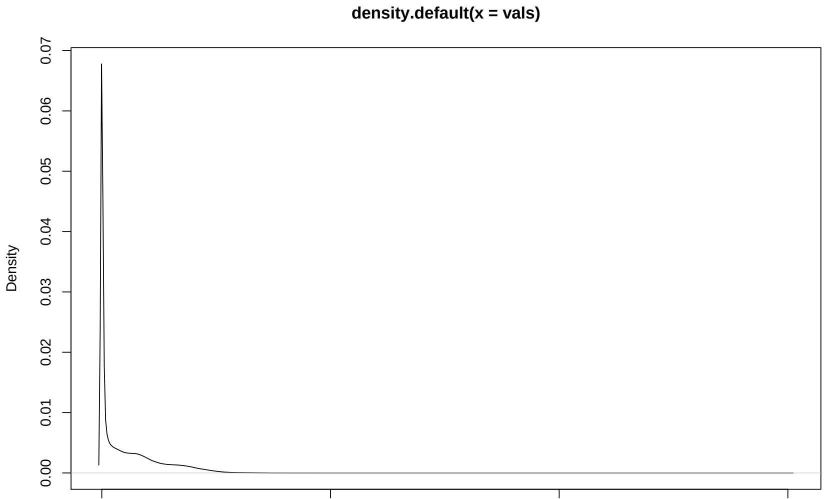
```
[1] "numeric"
```

Note an `array` can be reconstructed by using `array(vals, dim = dim(t1))` and will be in the correct order as the way R creates vectors and arrays.

Histogram

From these values we can do all the standard plotting/manipulations of data. For example, let's do a marginal density of the values:

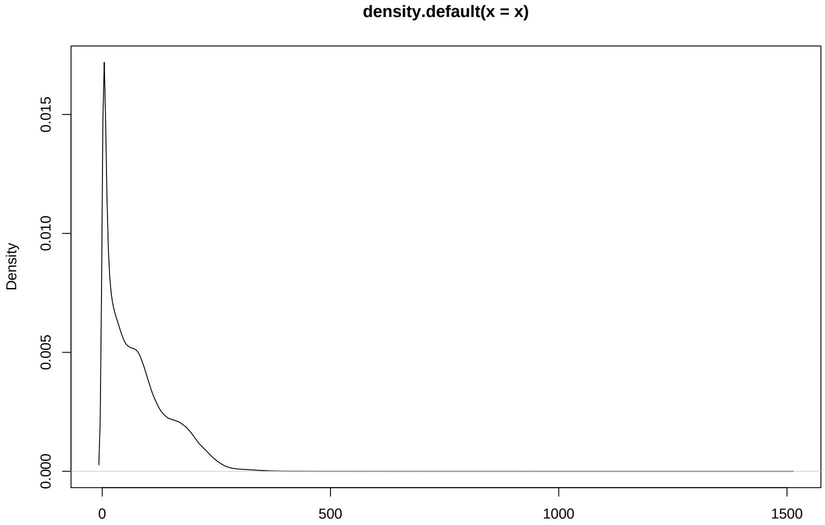
```
plot(density(vals))
```



Histogram

You can also pass in a mask to most standard functions:

```
plot(density(t1, mask = t1 > 0))
```



File helpers

Use `paste` if you want to put strings together with spaces:

Use `paste0` if you want no spaces by default:

`file.path(directory, filename)` will paste `directory` and `filename` with file separators (e.g. /)

```r
paste("img", ".nii.gz")
```

```
[1] "img .nii.gz"
```

```r
paste0("img", ".nii.gz")
```

```
[1] "img.nii.gz"
```

```r
file.path("output_directory", paste0("img", ".nii.gz"))
```

```
[1] "output_directory/img.nii.gz"
```

Operations with `nifti` objects

- Comparison: `>`, `>=`, `<`, `<=`, `==` (equals), `!=` (not equal)
- Logical: `!` - not, `&` - and, | - or (a “pipe”)
- Arithmetic: `+`, `-`, `*`, `/`, `^` - exponents
- Standard math functions: `log`, `abs`, `sqrt`

These work with an image and a number (`img + 2`) or two images of the same dimensions `img1 + img2`.

Main Packages we will use

- `oro.nifti` - reading/writing NIfTI images
- `neurobase` - extends `oro.nifti` and provides helpful imaging functions
- `fslr` - wraps FSL commands to use in R
  - registration, image manipulation
- `ANTsR` - wrapper for Advanced normalization tools (ANTs) code
  - registration, inhomogeneity correction, lots of tools
- `extrantsr` - allows `ANTsR` to work with objects from `oro.nifti`

Data Packages we will use

- `ms.lesion` - contains training/testing data of patients with multiple sclerosis (MS)
  - from the MS lesion challenge 2016 (`http://iacl.ece.jhu.edu/index.php/MSChallenge`)
- `kirby21.t1` - scan-rescan data for 3 subjects from Landman et al. (2011)
  - `https://www.nitrc.org/projects/multimodal`

Conclusions

- We have (briefly) covered some R data classes and types to get you started
- We will be using `nifti` objects
  - They are special 3-dimensional arrays
  - Contain numbers or logicals
- We have briefly covered subsetting and image manipulation
  - more on that later

Lists

- Initialize an empty list and add two elements to it

```
l = list()
l[[1]] = v
l[[2]] = m
print(l)
```

```
[[1]]
[1] 1 4 3 7 8

[[2]]
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- Subsetting:

```
print(l[[1]])
```