

# Imaging Packages in R

## Some packages we will use

All packages we will discuss are loaded on the RStudio Server:

- `oro.nifti` - reading/writing NIfTI images
  - made the `nifti` object/data class: like an array - but with header information
  - the main data class we will use
- `neurobase` - extends `oro.nifti` and provides helpful imaging functions

Let's load them:

```
library(oro.nifti)
library(neurobase)
```

## Reading in NIfTI images: assignment

We will use the `readnii` function (from `neurobase`) to read in a `nifti` object (this is an R object).

Here we read in the “training01\_01\_t1.nii.gz” file, and assign it to an object called `t1`:

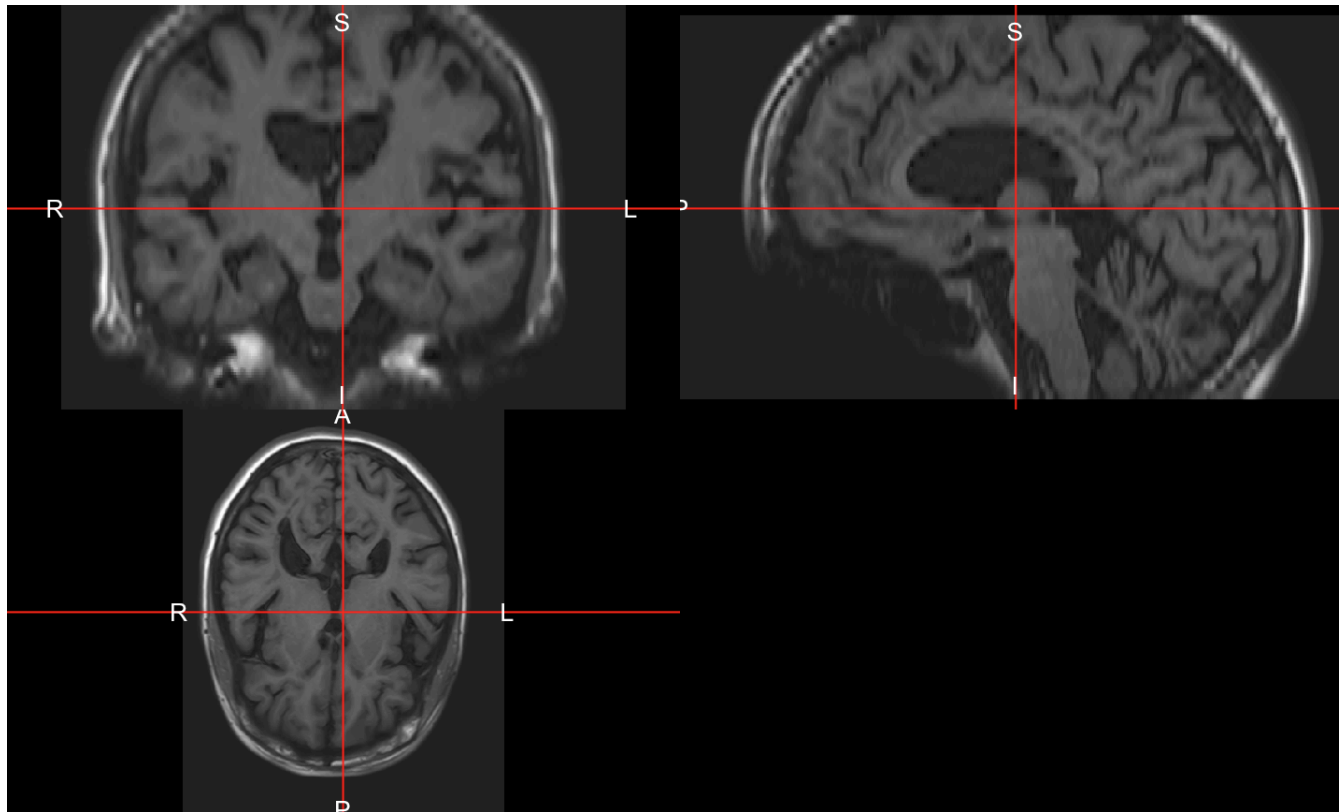
```
t1 = readnii("training01_01_t1.nii.gz")
```

Now, an object `t1` is in memory/the workspace.

```
class(t1)
```

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

# nifti images



## nifti images

By default, if you simply pass the object, it is printed, we can also do `print(t1):`

```
t1
```

```
NIIfTI-1 format
```

```
Type           : nifti
Data Type       : 4 (INT16)
Bits per Pixel  : 16
Slice Code      : 0 (Unknown)
Intent Code     : 0 (None)
Qform Code      : 2 (Aligned_Anat)
Sform Code      : 1 (Scanner_Anat)
Dimension       : 408 x 512 x 152
Pixel Dimension : 0.43 x 0.43 x 0.82
Voxel Units     : mm
Time Units      : Unknown
```

# Operations with **nifti** objects

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

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

```
t1 + t1 + 2 # still a nifti
```

NIfTI-1 format

|                 |                      |
|-----------------|----------------------|
| Type            | : nifti              |
| Data Type       | : 4 (INT16)          |
| Bits per Pixel  | : 16                 |
| Slice Code      | : 0 (Unknown)        |
| Intent Code     | : 0 (None)           |
| Qform Code      | : 2 (Aligned_Anat)   |
| Sform Code      | : 1 (Scanner_Anat)   |
| Dimension       | : 408 x 512 x 152    |
| Pixel Dimension | : 0.43 x 0.43 x 0.82 |
| Voxel Units     | : mm                 |
| Time Units      | : Unknown            |

## Working with **nifti** objects

Again, we can use a logical operation. Let's create an image indicating values over \400\:

```
class(t1 > 400) # still a nifti
```

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

```
head(t1 > 400) # values are now logical vs. numeric
```

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

We will refer to images such as `t1 > 400` as a “mask”, simply binary images with logical values in them (or 0s and 1s)

## Subsetting with `nifti` objects: like arrays

The subsetting here is similar to that of arrays. Since `t1` is 3-dimensional the subsetting goes to the 3rd dimension:

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

```
[1] 0
```

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

```
t1[, 4, ] # returns a 2-d matrix
```

```
t1[1, , ] # returns a 2-d matrix
```

- You can subset with a logical array of the same dimensions!
- We can view values of the `t1` greater than 400 (`head` only prints the first 6 values):

```
head(t1[ t1 > 400 ]) # produces a vector of numbers
```

```
[1] 402 412 435 448 453 430
```



## which with nifti objects

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

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

|      | dim1 | dim2 | dim3 |
|------|------|------|------|
| [1,] | 180  | 258  | 1    |
| [2,] | 175  | 259  | 1    |
| [3,] | 176  | 259  | 1    |
| [4,] | 177  | 259  | 1    |
| [5,] | 178  | 259  | 1    |
| [6,] | 179  | 259  | 1    |

But can get the “vector” indices as well:

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

```
[1] 105036 105439 105440 105441 105442 105443
```

## 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 # replaced these values!  
max(t1_copy) # should be 400
```

```
[1] 400
```

```
max(t1)
```

```
[1] 1691
```

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_t1_under400.nii.gz")  
file.exists("training01_t1_under400.nii.gz")
```

```
[1] TRUE
```

We have seen that `file.exists` returns `TRUE` if a file exists

- useful in conjunction with `all`: `all(file.exists(VECTOR_OF_FILES))`

## Vectorizing a `nifti`

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

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

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

Essentially “strings out” the array. If you do `array(c(t1), dim = dim(t1))`, this will put things back “in order” of the `t1`.

Vectorizing is useful for making `data.frames` (covered later) when you want to do modeling at a voxel level.

```
df = data.frame(t1 = c(t1), mask = c(t1 > 400)); head(df)
```

```
   t1  mask
1    0 FALSE
2    0 FALSE
3    0 FALSE
4    0 FALSE
5    0 FALSE
6    0 FALSE
```

## File helpers - for constructing filenames

Use `paste` if you want to put strings together with spaces, `paste0` no spaces by default.

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

```
c(paste("img", ".nii.gz"), paste0("img", ".nii.gz"))
```

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

```
x = file.path("output_directory", paste0("img", ".nii.gz")); print(x)
```

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

`nii.stub` will strip off the nifti extension. If `bn = TRUE`, it removes the directory as well:

```
c(nii.stub(x), nii.stub(x, bn = TRUE))
```

```
[1] "output_directory/img" "img"
```

## 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, skull stripping
- `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 Package we will use

- `ms.lesion` - contains training/testing data of patients with multiple sclerosis (MS)
  - from an open MS MRI data set (Lesjak et al. 2017)

## 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
- `readnii` and `writenii` are used for reading/writing `nifti` objects to NIfTI files
- We have briefly covered subsetting and image manipulation
  - more on that later

## Website

[http://johnmuschelli.com/imaging\\_in\\_r](http://johnmuschelli.com/imaging_in_r)

Lesjak, Žiga, Alfiia Galimzianova, Aleš Koren, Matej Lukin, Franjo Pernuš, Boštjan Likar, and Žiga Špiclin. 2017. "A Novel Public MR Image Dataset of Multiple Sclerosis Patients with Lesion Segmentations Based on Multi-Rater Consensus." . Springer, 1–13.