RMINC

***The most up to date information is available on the RMINC on Github page. It should be the FIRST place you explore for information. The README (scroll down on the homepage) contains numerous examples/tutorials. Including how to do voxel-wise statistics, visualizing with RMINC, etc.***

RMINC

RMINC is a library for the R statistical environment designed to read and write MINC volumes as well as perform optimized statistical routines at every voxel of a set of files.

- Development
- Documentation
  - Some of the latest tutorials from GitHub
  - Analyzing volume differences in neuroanatomy with a single set of labels
  - Analyzing differences in neuroanatomy with multiple sets of labels
  - pMincApply
- Installation
  - Installing RMINC from a release:
  - Installing RMINC using the current state of the package:

Development

The main development home for RMINC can be found here: RMINC on Github - that page includes access to the source code, downloads, bug reporting, and facilities for asking questions (and getting them answered, of course).

Documentation

The main RMINC documentation comes in the form of a mini-book VBMstats.pdf (Note: this document was last updated on April 16, 2008).

Some of the latest tutorials from GitHub

RMINC Visualization: https://rawgit.com/Mouse-Imaging-Centre/RMINC/master/inst/documentation/visualizationTutorial.html
Exploring your data using the shiny app (launch_shinyRMINC):

```
# instructions on how to use the shiny app:
?launch_shinyRMINC
```
Analyzing volume differences in neuroanatomy with a single set of labels
To analyze individual structures in the brain, you will need two things: a set of registered images and an atlas that segments the final average of your registration pipeline into classified labels. This is known as a classified atlas. (Information about how you can align a segmented set of labels to your data set can be found here: Atlas to Atlas Segmentation) The main idea is the following: Through the registration pipeline we have information about the change in volume of the voxels for each of the individual mouse brains. This is captured in the Jacobian determinant files. Using the classified atlas, we can integrate the information captured in the Jacobian determinants to find out the volume of the structures for each of the individual input files.

The Jacobian determinant fields are fairly noisy if you use them unprocessed. For this reason we blur them before we do analysis on them. The amount of blurring you want to use depends on what kind of changes you want to capture. When looking at neuroanatomy it is important to use only a small amount of blurring, because you want to avoid smoothing out information around borders of structures. The mouse brains we generally look at have a resolution of 56 micron, and we do structural analysis on Jacobian determinants that have been blurred using a 100 micron kernel.

A second thing to keep in mind is that the registration pipeline produces two types of Jacobian determinants. One that reflects relative changes (overall scaling or brain size differences have been taking out) and absolute changes which still contain the brain size differences. When analyzing neuroanatomical structures you want to use the determinants that reflect the absolute changes. Currently the naming convention for those files is as follows:

FILENAME_BASE-log-determinant-scaled-fwhm_KERNEL_.mnc

Here is an example of how to do the analysis in R

1. When you are analyzing structures, make sure that the label file you use accurately segments out your final non-linear model, otherwise your analysis will be meaningless.
2. Use the Jacobian determinants that capture the absolute differences (in our case the "log-determinant-scaled")
3. Use little blurring, for instance for files with a 56 micron resolution use 100 micronblur (in our case the "log-determinant-scaled-fwhm0.1")
> library(RMINC)

# There is a variable that was loaded into R containing the genotype and the location of the Jacobian
determinants called filenames, it contains the following:

> filenames

<table>
<thead>
<tr>
<th>scaled_jacobians</th>
<th>genotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>01_wt-log-determinant-scaled-fwhm0.1.mnc</td>
<td>wt</td>
</tr>
<tr>
<td>02_wt-log-determinant-scaled-fwhm0.1.mnc</td>
<td>wt</td>
</tr>
<tr>
<td>03_wt-log-determinant-scaled-fwhm0.1.mnc</td>
<td>wt</td>
</tr>
<tr>
<td>04_wt-log-determinant-scaled-fwhm0.1.mnc</td>
<td>wt</td>
</tr>
<tr>
<td>05_mut-log-determinant-scaled-fwhm0.1.mnc</td>
<td>mut</td>
</tr>
<tr>
<td>06_mut-log-determinant-scaled-fwhm0.1.mnc</td>
<td>mut</td>
</tr>
<tr>
<td>07_mut-log-determinant-scaled-fwhm0.1.mnc</td>
<td>mut</td>
</tr>
<tr>
<td>08_mut-log-determinant-scaled-fwhm0.1.mnc</td>
<td>mut</td>
</tr>
</tbody>
</table>

# The first thing to do now is to get the volume information of all the structures, the atlas with the labels
is called resampled_atlas.mnc

> volumes <- anatGetAll(filenames$scaled_jacobians, "resampled_atlas.mnc")

# This atlas contains information about both left and right structures, but in this case we will combine the
information of left and right structures and only look at combined volumes:

> volumes_combined <- anatCombineStructures(volumes)

# Run a linear model to get information about f-statistics and t-statistics on the structures. The first
argument is the formula to be used, in this case we want to look at differences between the genotype, the
second argument is the data which is stored in the variable filenames, and lastly the actual volume information

> anatLm(~ genotype, filenames, volumes_combined)

# The output will be a list of all the structures with a number of columns for the values of the f-statistics
and t-statistics. Next to find out which of these changes survive multiple comparison corrections:

> anatFDR(anatLm(~ genotype, filenames, volumes_combined))

Analyzing differences in neuroanatomy with multiple sets of labels

In addition to the procedure described above (which uses a single atlas and jacobian determinants for each individual brain) we can generate a unique set
of labels for each individual brain. This can be done using MAGeT. Once you have a set of labels for each brain, analysis can proceed as described above,
but with one major difference: the anatGetAll command must be called with a different set of arguments. Let’s look at this function more closely:
> anatGetAll
function (filenames, atlas = NULL, method = "jacobians", defs = "/projects/mice/jlerch/cortex-label/c57_brain_atlas_labels.csv",
  dropLabels = FALSE, side = "both")

The default values for method, defs, dropLabels and side are set for users at MICe who do 56 micron ex-vivo registrations. In these cases, the anatGetAll call is as shown above:

volumes <- anatGetAll(filenames$scaled_jacobians, "resampled_atlas.mnc")

For labels generated using MAGeT, you would need the following:

- filenames: Instead of the scaled_jacobian files, you would instead include the labels for each brain generated by MAGeT.
- atlas: This argument is NOT USED when each file has its own set of labels.
- method: method = "labels" must be specified
- defs: This is critical if the label definitions for the files you are looking at differ from the standard set of 62, described in Dorr, et. al. This will also need to be specified for users who wish use the set described in Dorr et.al, but are not using the machines at MICe.
- dropLabels and side can continue to use the defaults.

So, putting it all together, here is what your anatGetAll call and following analysis would look like:

#anatGetAll call, with slightly different arguments
volumes <- anatGetAll(filenames$labels, method="labels", defs="brain_label_mappings.csv")

#combining structures, anatLm, anatFDR proceed as above:
volumes_combined <- anatCombineStructures(volumes, defs="brain_label_mappings.csv")
anatLm(~ genotype, filenames, volumes_combined)
anatFDR(anatLm(~ genotype, filenames, volumes_combined))

**pMincApply**

pMincApply, just as mincApply, can be used to run any function on all voxels in your input files. The difference with mincApply, is that pMincApply can be parallelized (hence the p). You can use snowfall to run it locally using multiple cores on your machine, or sge to submit jobs to a batch system. Here is a short example of how to use pMincApply to run a function on your input files that is not implemented by any of the standard MINC functions.
# load the RMINC library
library(RMINC)

# load the mapping of your input files and in this case genotype
# the aim of this example is to test the variance of the Jacobian determinants between wild types and mutants
gf <- read.csv("filenames_and_genotype.csv")

# Figure out how your function works on your data. For example test
# it on a single voxel:
voxel <- mincGetVoxel(gf$jacobians, 0,0,0)

# Run the function you want to use
# the "car" library provides "leveneTest"
library(car)
leveneTest(voxel, gf$Strain, center=mean)

# the output looks like this:
Levene's Test for Homogeneity of Variance (center = mean)
   Df F value Pr(>F)
group  2  1.5514 0.2188
   74

# in this example we will extract the F value, and the
# p value which can be done as so:
leveneTest(voxel, gf$Strain, center=mean)[1,2]
leveneTest(voxel, gf$Strain, center=mean)[1,3]
# also it's smart to encapsulate the return value by
# as.numeric to make sure it's returned as a number

# write a function that will be passed on to pMincApply. This
# function will directly refer to variables in the gf object
# this function will return the F-value and p value from the Levene's Test
# which comes from row 1, column 2 (as a numeric value) and column 3
leveneTestForRMINC <- function(x) {
  fout <- as.numeric(leveneTest(x, gf$Strain, center=mean)[1,2])
  pout <- as.numeric(leveneTest(x, gf$Strain, center=mean)[1,3])
  return(c(fout, pout))
}

# Use the "global" argument to specify all variables and functions you need
# Use the "packages" argument to specify all libraries
# The mask will restrict the calculations to that area
outLeveneTest <- pMincApply(gf$jacobians, quote(leveneTestForRMINC(x)), mask="mask.mnc", workers=8, global=c("gf","leveneTestForRMINC"), packages="car")

# write your stats out to file
mincWriteVolume(outLeveneTest, "f_values.mnc", 1)
mincWriteVolume(outLeveneTest, "p_values.mnc", 2)

# instead of running the pMincApply in parallel on your machine, you can also use sge:
outLeveneTest <- pMincApply(gf$jacobians, quote(leveneTestForRMINC(x)), mask="mask.mnc", workers=8, global=c("gf","leveneTestForRMINC"), packages="car", method="sge")

# Installation

Installing RMINC from a release:
• Download the tarball from the Github RMINC website https://github.com/mcvaneede/RMINC/tree/master/releases
• By default, the library will be installed in /usr/share. If you want to change this location, the R_LIBS variable needs to be set:

```bash
export R_LIBS=/build/directory
```

• Install the package (example is for the tarball version 0.5):

```bash
R CMD INSTALL RMINC_0.5.tar.gz --configure-args="--with-build-path=/install/directory/minc2"
```

### Installing RMINC using the current state of the package:

• Retrieve a copy of the bazaar repository of the RMINC library

```bash
git clone https://github.com/mcvaneede/RMINC.git RMINC
```

• The R_LIBS variable determines where the library is installed. By default it will be installed under /usr/share. If you want to install the library somewhere else, the R_LIBS environment variable should be set.

```bash
export R_LIBS=/build/directory
```

• Run autogen.sh

```bash
cd RMINC
./autogen.sh
cd..
```

• Install the package

```bash
R CMD INSTALL RMINC --configure-args="--with-build-path=/install/directory/minc2"
```

• If for some reason setting the R_LIBS environment variable does not work, you can also explicitly state where you want to install the library as follows:

```bash
R CMD INSTALL RMINC --library=/patch/to/library/tree --configure-args="--with-build-path=/install/directory/minc2"
```