Voxel-wise morphometry using RMINC
Version 1.0

Jason Lerch

May 13, 2010

1 Introduction

This document provides a worked example of voxel-wise morphometry using the RMINC library to R. The applications include Voxel Based Morphometry (VBM), Deformation Based Morphometry, or any method in which the goal is to produce a statistical test at every voxel of a series of files. The assumption is that the data of interest at every voxel is a scalar (i.e. a single value rather than a vector).

This document does not talk about preparing the data. For an overview of the steps necessary to perform VBM, for example, see `http://wiki.bic.mni.mcgill.ca/index.php/VoxelBasedMorphometry`

1.1 About R

The steps described herein use the R statistical environment (`http://r-project.org`). It is possible to simply follow this tutorial without having used R before, though everything will make a lot more sense with some basic exposure to that excellent (and free) program, as I won't go into any details about particular R commands. The web-site listed above has fairly extensive documentation, and there are some good books on R available as well.

1.2 About RMINC

RMINC is an R library designed to work with MINC2 files. It provides functionality to read and write MINC2 volumes, and to apply arbitrary R functions at every voxel of a set of files. There are also three specialized loops written to perform t-tests, Wilcoxon Rank-Sum tests, and linear models at every voxel.

1.3 About MINC

MINC is an excellent library, file format, and set of applications to work with medical imaging data. Note that RMINC needs to use MINC2 volumes and the MINC2 library and will not work with MINC1 files. Conversion to MINC2 from

MINC1 is easily accomplished using mincconvert. The conversion is thankfully lossless.

2 Input data

Once the files have been processed, the easiest way to proceed is by settting up a text file containing all the necessary information about each scan. This file should be comma or space separated, have on row per scan, with each column containing info about each scan. One of the columns should contain the filename pointing to the MINC volumes to be processed. An example might look like the following:

```
file,gender,scale
filename1.mnc,Male,0.98
filename2.mnc,Female,0.91
filename3.mnc,Female,0.92
```

Notice how the first row contains a header. This is optional, but makes later access to the data easier and is therefore recommended.

The next step is to actually load this filename into R. The steps are given below:

```
> library(RMINC)
> library(xtable)
> glim.file <- "/projects/mice/jlerch/male-female/all/analysis/male-female-scales.glim"
> gf <- as.data.frame(read.csv(glim.file, header=T))
```

The two library commands load the RMINC and xtable libraries into R. xtable is only necessary for display purposes in this manual (which, by the way, is being written using Sweave, a tool for combining R with latex). The variable `glim.file` is then assigned the location of the input data file, which in turn is read into the `gf` variable.

```
> xtable(gf[c(1, 2, 3, 22, 23, 24), c(2, 4, 5, 6)],
+     caption = "GLIM File")
```

The little code fragment and table above just shows what a subset of the gf variable looks like. To get a feel for the data and some of the RMINC functions, the first file is then read and a histogram produced (for the curious, the files used in this example actually consist of log Jacobians of deformation fields, not voxel density maps usually used in VBM).

```
> first.file <- mincGetVolume(gf$file[1])
> first.file
> hist(first.file)
```

Three things happened here. The first was a call to `minc.get.volume`, which returns a 1D array of size dimension1 * dimension2 * dimension3 (notice that the dimension sizes are printed out when this command is run). RMINC tends not to care about the 3D nature of these volumes. The main reason is that for the majority of statistical purposes neighbourhood information does not matter, as long as all input files share the same sampling (which is assumed). Moreover, if you really want to perform image processing operations that need this type of neighbourhood information - and thus cannot be vectorized by R - then you probably should not be using R in the first place.

The second command prints information about the volume to the terminal. The third command plots a histogram of the entire volume. Fairly self-explanatory.

3 Voxel-wise analyses

RMINC has the ability to execute arbitrary R functions at every voxel. This is done by looping over all voxels, creating a vector `x` of the same length as the number of volumes, the real value of each subject at that voxel being assigned to the vector. To illustrate with an example, to get a map of the mean values of all subjects, one would execute the following:

```
> mean.map <- mincApply(gf$file, quote(mean(x)))
> mincWriteVolume(mean.map, "/tmp/mean.mnc")
```

This is what happened: the R function `mean` was applied at every voxel to of all files, producing a 1D array called `mean.map`. The command `mincApply` needs two arguments: a list of filenames, and a string to be evaluated at every voxel. The string should contain a valid R expression, and should do something to the variable `x`. It should also produce a single scalar as output. A third, and optional, argument is a string containing the filename of a mask volume, which will result in the function only being evaluated where the mask is not 0.

After the mean was computed, it was written to a MINC2 file. `mincWriteVolume` needs two argumets: the data to be written to volume and the filename to write to.

A slightly more complex example involves using the built in `t.test` function to compute a t.test at every voxel.

```
> mask <- "/projects/mice/jlerch/male-female/all/analysis/mask.mnc"
> f <- function(x) {
+    t.test(x ~ gf$gender$statistic)
+ }
> t <- mincApply(gf$file, quote(f(x)), mask)
> minc.write.volume(t, "/tmp/t-test.mnc")
```

Here a function is created, which simply calls `t.test` comparing the voxel to the gender assignement from the input file. Only the statistic is output. Note that, since the third argument is present, this test is only evaluated inside the mask. The file is then output.

3.1 The need for speed

`mincApply` is tremendously powerful since it allows any R function to be used, though a little work might be necessary to coerce it to only output a single scalar. It can be, however, excruciatingly slow. The most common useful functions have thus been entirely rewritten in C. The difference is noticeable. For a t-test, for example, the above bit of code takes over half an hour to complete. The equivalent using the C functions (explained below), takes less than 7 seconds [1]

The function in question is `minc.model`. It can, at the moment, do three things: t-tests, Wilcoxon Rank-sum Tests (a.k.a. Mann-Whitney U tests), and linear models. They are explained in turn.

The t-test version of `minc.model` needs two arguments: a list of filenames, a vector of group assignments (presented in the same order as the filenames. The latter should evaluate to 1 or 0 when cast to a double. A mask can be specified as an optional argument. The same holds for the Wilcoxon Rank-sum test.

```
> t2 <- minc.model(gf$file, gf$gender == "Female", "t-test")
> w <- minc.model(gf$file, gf$gender == "Female", "wilcoxon")
> minc.write.volume("/tmp/t-test2.mnc", gf$file[1], t2)
> minc.write.volume("/tmp/w-test.mnc", gf$file[1], w)
```

Notice how the second argument is forced to evaluate to 0 or 1 (or TRUE or FALSE) by doing an explicit check against one of the groups.

Linear models are slighlty different. The second argument should be a matrix, which is most easily obtained by using the R function `model.matrix`. This is best illustrated by an example:

```
> l <- minc.model(gf$file, model.matrix(file ~ gender, gf), "lm")
> minc.write.volume("/tmp/lm.mnc", gf$file[1], l[,2])
```

The output above should actually be the same as t2. Unlike the t-test or wilcoxon version of minc.model, the lm version actually produces a matrix as an output, which has as many rows as there are voxels, and as many columns as there are terms in the model matrix. In the above case there are two terms: the intercept plus the gender term, which is why only the second one (the gender term) is written to file. The values output are t-statistics representing the marginal significance of each term.

A quick note about memory usage: both `minc.apply` and `minc.model` make use of a slice loop construct. In other words, an entire slice for each volume being evaluated is loaded into memory at a time, but no more. This is a lot friendlier than loading entire volumes into memory, but might still be too much for either vast numbers of volumes or extremely large volumes.

[1]These numbers using the same 40 volumes of size 141 by 274 by 210, performed on an AMD 2.6 Ghz 64-bit processor.

4 Plotting

R is a marvelous environment for plotting. RMINC makes this available by allowing for easy access to individual voxels. The relevant commands are illustrated below:

```
> voxel <- minc.get.voxel.from.files(gf$file, 69,167,105)
> plot(voxel ~ gender, gf)
```

The function `minc.get.voxel.from.files` takes four arguments: a list of filenames, and the voxel coordinates to get. Note that these coordinates are in the dimension order of the volumes - in other words, if they are ordered Z-Y-X, then the arguments should be ordered in Z-Y-X as well.

5 Thresholding

Thresholding is a great example of why performing these voxel wise analyses inside R is a good thing. Take, for example, the following code, which generates p-values from the earlier created t-statistics and then adjusts them based on the False Discovery Rate:

```
> p.values <- pt(t2, 38)
> min(p.values)
> p.fdr <- p.adjust(p.values, "fdr")
> min(p.fdr)
```

Two R functions are used - the first to generate p-values from the t-statistics (given 38 degrees of freedom), the second to adjust them for multiple comparisons using the FDR technique. The above is a bit misleading, however, since controlling for multiple comparisons is best done inside a mask (i.e. just the brain) rather than the whole volume. This too is easily done:

```
> mask <- "/projects/mice/jlerch/male-female/all/analysis/mask.mnc"
> mask.volume <- minc.get.volume(mask)
> p.fdr.mask <- vector(length=length(p.fdr), mode="numeric")
> p.fdr.mask[mask.volume > 0.5] <- p.adjust(p.values[mask.volume > 0.5], "fdr")
> minc.write.volume("/tmp/fdr-corrected-p-values.mnc", gf$file[1], p.fdr.mask)
```

The key difference to the above is just the array indexing used to restrict the thresholding operation.

6 Misc.

There are two other functions that might be of interest: `minc.get.hyperslab` and `minc.get.volume`. The latter was already illustrated in an early example, the former is similar in that it takes a filename as an argument, as well as a vector of length 3 of the starts, and a same-sized vector of counts. These should again be given in the same order as dimension ordering of the volume.