Neuropointillist Introduction – MDC ABCD Workshop July 2021

Purpose
This is a laptop-based introduction to neuropointillist. To run realistic and interesting models, you will probably need access to a cluster.

Conventions
You will be typing commands in a terminal window in a UNIX environment of some type (either MacOS, Linux, or the Linux Subsystem for Windows). Commands and file names are indicated in `courier font.` Steps that are not just informational, where you have to do something, are highlighted in yellow.

Before We Begin
Hopefully you have been able to follow the installation instructions on http://ibic.github.io/neuropointillist/installation.html to install the package and prerequisites on your laptop.

Make sure you have the most up to date version of the files.

1. If you have used `git` to install the package, update them using git:

   ```
   cd neuropointillist
   git pull origin master
   <type git username>
   <type git password>
   ```

2. If you downloaded the package as a zip file and unpacked it, do that again.
   ```
   wget https://github.com/IBIC/neuropointillist/archive/master.zip
   unzip master.zip
   mv neuropointillist neuropointillist.old
   mv neuropointillist-master neuropointillist
   ```

   If you do not have the command wget, you can simply go to the URL in your browser and save the file to someplace you can find it!

   Make sure that the neuropointillist directory is in your path. If you are not sure, from a terminal window type:

   ```
   export PATH=$PATH:~/neuropointillist
   ```

Walkthrough

Cd into the neuropointillist directory (top level). In the `neuropointillist` directory, there are three executables:

```
npoint
npointmerge
npointrun
```

`npoint` is the main executable that runs the R functions that read in a bunch of 4d fMRI or 3d statistics files and a model file and either splits them up into jobs to be run on a cluster, or runs them in parallel.

`npointrun` is used to run individual jobs.

`npointmerge` is used to merge all the results of split up jobs into output files.

`cd` into the directory `example.rawfmri.`
This directory has an example of simulated fMRI data and a model to show you how to set up an analysis of raw fMRI data.

Edit the file `readargs.R` with your favorite text editor.

This file is a convenience file that, if it exists in a particular directory, specifies the arguments for `npoint`. You could just as easily create a bash script to call `npoint` with the correct arguments, but I thought this approach helped keep all the files straight.

The arguments specify the following:
`-m mask_4mm.nii.gz`: This is a NiFTI file that contains a "1" in every voxel that should be analyzed and a "0" in every voxel you would like to ignore. Because we only analyze voxels that are in the brain, in grey matter, or in a particular structure, the mask greatly cuts down on computation time. To shorten the time to run the example, replace this file with `oneslice_4mm.nii.gz`.

```
-set1 setfilenames1.txt
-set2 setfilenames2.txt
```

The "setfilenames"options specify lists of files for each timepoint. They can be absolute paths or relative from the directory that you are in. You can specify up to five lists of files using command line flags.

```
-setlabels1 setlabels.csv
-setlabels2 setlabels.csv
```

The "setlabels" options specify any important information (subject numbers, occasion of measurement, covariates such as age or gender) for the corresponding setfile.

There is no rule that setfiles need to consist of data from only one timepoint. It is only a convenience to help people organize data from longitudinal studies. You can put all the

information that you need to model in one setfilenames file and one corresponding setlabels file.

`npoint` does the following:

1. It reads the mask and every one of the files listed in the setfiles.
2. It identifies the voxels in the mask.
3. If the files are 3D statistical output from a first level analysis, or any other 3D file (dirty secret – you can put ANYTHING IN HERE), it collapses the 3D file into a 1D vector where the columns are different voxels in the mask.
4. If the files are 4D fMRI files, it collapses each 4D file into a 2D matrix where the columns are different voxels in the mask and the rows are TRs.
5. It reads the comma-separated-value setfiles. The names of the columns are the
6. It makes sure that the number of rows in each setfiles corresponds to the number of rows generated by steps 3 or 4. In other words, if you are using 4D data, you need to provide a row for each TR. Each row might contain the convolved values of each explanatory variable.
7. It creates the data structures "voxeldat" with the collapsed MR data, and the "designmat" with the data that is used to model the MR data.

`-model fmrimodel.R`

This flag specifies the code that you provide. This code will be run on each voxel from the voxeldat and "attaches" to the variables in the designmat so that you can use those variables in ANY WAY and save ANY THINGS out to files.

`-output sgedata/sim`

This flag specifies a prefix for all the output files.

`-debug debug.Rdata`

It is helpful when developing your model to test it. This debug flag specifies to write out the data structures into an Rdata file that you can read into R, to more closely figure out what is going on in particular voxels.

`-sgeN 10`

This flag specifies how many pieces to split up the processing.

Edit `fmrimodel.R`

I won't talk a lot more about the model here – the important things to note are:

1. You define a `processVoxel` command in R that takes a voxel number (v) from the collapsed voxeldat structure.
2. Because you don't want the code to die if there is an error at one voxel (when there are so many!) you need to trap errors carefully and make sure to return some value that you can interpret as an error (and go back to the debug file as necessary)

3. You can return any number of values. The names that you give them are used to construct the output filenames.

Run `npoint`

This will read the arguments described above and generate a directory called `sgedata` with the output files. It will not actually run the model.

`cd` into `sgedata`

Here you can see a variety of files. Each chunk has a corresponding NiFTI file mask (with the .nii.gz extension) and its representation for R (with an .rds extension). You can also see the `debug.Rdata` file.

The files that are used to actually run the model begin with "runme". In this example, we ignore `runme.sge`, because you probably do not have access to a cluster that runs the Sun/Open Gridengine software. In the advanced workshop I will show you how to run the software on the Slurm Cluster.

Try running `runme.local`:

```
./runme.local
```

You need to specify the "./" in front of runme.local, because it is a command that is not in your PATH environment variable. (This is a UNIX thing).

This command will call the `npointrun` command to run each individual job, one by one (sequentially). When they are all finished, it will merge the outputs together using `npointmerge`.