13 June 2016

Delaytools User Manual

Delaytools is a suite of python programs used to analyze time resolved imaging data to find time lagged correlations between the voxelwise time series and other time series.

Why do I want to know about time lagged correlations?

This comes out of work by our group (The Opto-Magnetic group at McLean Hospital - http://www.nirs-fmri.net) looking at the correlations between neuroimaging data (fMRI) and NIRS data recorded simultaneously, either in the brain or the periphery. We found that a large fraction of the "noise" we found at low frequency in fMRI data was due to real, random[*] fluctuations of blood oxygenation and volume (both of which affect the intensity of BOLD fMRI images) in the blood passing through the brain. More interestingly, because these characteristics of blood move with the blood itself, this gives you a way to determine blood arrival time at any location in the brain. This is interesting in and of itself, but also, this gives you a method for optimally modelling (and removing) in band physiological noise from fMRI data (see references below).

After working with this for several years we've also found that you don't need to used simultaneous NIRS to find this blood borne signal - you can get it from blood rich BOLD voxels for example in the superior sagittal sinus, or bootstrap it out of the global mean signal in the BOLD data. You can also track exogenously applied waveforms, such as hypercarbic and/or hyperoxic gas challenges to really boost your signal to noise. So there are lots of times when you might want to do this type of correlation analysis. This package provides the tools to make that easier.

A note on coding style:

This code has been in active development since June of 2012. This has two implications. The first is that it has been tuned and refined quite a bit over the years, with a lot of optimizations and bug fixes - most of the core routines have been tested fairly extensively to get rid of the stupidest bugs. I find new bugs all the time, but most of the showstoppers seem to be gone. The second result is that the coding style is all over the place. When I started writing this, I had just moved over from C, and it was basically a mental port of how I would write it in C, and was extremely unpythonic. Over the years, as I've gone back and added functions, I periodically get embarassed and upgrade things to a somewhat more modern coding style. I even put in some classes - that's what the cool kids do, right? But the pace of that effort has to be balanced with the fact that when I make major architectural changes, I tend to break things. So be patient with me, and keep in mind that you get what you pay for, and this cost you nothing!

---

[*] "random" in this context means "determined by something we don't have any information about" - maybe EtCO2 variation, or sympathetic nervous system activity - so not really random.

13 June 2016

Python version:

This code has been most heavily tested in python 2.7. I've been slowly converting over to making it python 3 compatible, since all the libraries I needed have now been ported. As far as I know, the code works fine in python 3.5 - I've switched over to that on my development machine, and have not hit any version related issues in a while now, and according to PyCharm's code inspection, there are no incompatible constructions. However that's no guarantee that there isn't a problem in some option I haven't bothered to test yet, so be vigilant, and please let me know if there is some issue with python 3 that I haven't caught (or any bugs, really).

Why are you releasing your code?

For a number of reasons. The first is that I want people to use it! I think if it were easier for people to do time delay analysis, they'd be more likely to do it. I don't have enough time or people in my group to do every experiment that I think would be interesting, so I'm hoping other people will, so I can read their papers and learn interesting things. The second is that it's the right way to do science – I can say lots of things, but if nobody can replicate my results, nobody will believe it (we've gotten that a lot, because some of the implications of what we've seen in resting state data can be a little uncomfortable). We've reached a stage in fMRI where getting from data to results involves a huge amount of processing, so part of confirming results involves being able to see how the data were processed. This has two implications. The first is that if you had to do everything from scratch, you'd never try to confirm anybody's results. The second is, in any complicated processing scheme, it's quite possible to make dumb mistakes, and I may have made some. As much as I'm queasy about somebody potentially finding a mistake in my code, I'd rather that they did so. If I'm wrong, I'd rather know it sooner than later, so having other people using my tools will help me find problems, and hopefully over time I can become more confident in my results[‡].

How do I cite this?

Good question – working on that.

What's included in this package?

I've included a number of tools to get you going – I'll add in a number of other utilities as I get them closer to the point that I can release them without people laughing at my code. For the time being, I'm including the following:

rapidtide2

The central program in this package is rapidtide2. This is the program that quantifies the time strength and time delay of pervasive signals in a BOLD fMRI dataset.

[‡] Just to be clear – I'm very confident that the broad conclusions we've drawn over the years are correct; we do use multiple tool chains for all of our analyses, and we've looked at an awful lot of data in a lot of ways. But that doesn't mean that all of our processing is perfect in every way, and I'm sure that there are better ways to do some of the things I do in my code.

13 June 2016

Description:

At its core, rapidtide is simply performing a full crosscorrelation between a "probe" timecourse and every voxel in an fMRI dataset (by “full” I mean over all time lags, rather than only at zero lag, as in a Pearson correlation). As with many things, however, the devil is in the details, and so rapidtide provides a number of features which make it pretty good at this particular task. A few highlights:

1) There are lots of ways to do something even as simple as a cross-correlation in a nonoptimal way (not windowing, improper normalization, doing it in the time rather than frequency domain, etc.). I'm pretty sure what rapidtide does is, if not the best way, at least a very good and very fast way.
2) rapidtide has been optimized and profiled to speed it up quite a bit; it has an optional dependency on numba – if it’s installed, some of the most heavily used routines will speed up significantly due to judicious use of `@jit`.
3) The sample rate of your probe regressor and the fMRI data do not have to match - rapidtide resamples the probe regressor to an integral multiple of the fMRI data rate automatically
4) The probe and data can be temporally prefiltered to the LFO, respiratory, or cardiac frequency band with a command line switch, or you can specify any low, high, or bandpass range you want.
5) The data can be spatially smoothed at runtime (so you don't have to keep smoothed versions of big datasets around). This is quite fast, so no reason not to do it this way.
6) rapidtide can generate a probe regressor from the global mean of the data itself - no externally recorded timecourse is required. Optionally you can input both a mask of regions that you want to be included in the mean, and the voxels that you want excluded from the mean (there are situations when you might want to do one or the other or both).
7) Determining the significance threshold for filtered correlations where the optimal delay has been selected is nontrivial; using the conventional formulae for the significance of a correlation leads to wildly inflated p values. rapidtide estimates the spurious correlation threshold by calculating the distribution of null correlation values obtained with a shuffling procedure at the beginning of each run (the default is to use 10000 shuffled correlations), and uses this value to mask correlation maps it calculates.
8) rapidtide can do an iterative refinement of the probe regressor by aligning the voxel timecourses in time and regenerating the test regressor.
9) rapidtide fits the peak of the correlation function, so you can make fine grained distinctions between close lag times. The resolution of the time lag discrimination is set by the length of the timecourse, not the timestep – this is a feature of correlations, not rapidtide.
10) Once the time delay in each voxel has been found, rapidtide outputs a 4D file of delayed probe regressors for using as voxel specific confound regressors or to estimate the strength of the probe regressor in each voxel. This regression is performed by default, but these outputs let you do it yourself if you are so inclined.

11) I've put a lot of effort into making the outputs as informative as possible - lots of useful maps, histograms, timecourses, etc.
12) There are a lot of tuning parameters you can mess with if you feel the need. I've tried to make intelligent defaults so things will work well out of the box, but you have the ability to set most of the interesting parameters yourself.

Inputs:

At a minimum, rapidtide needs a Nifti file to work on (space by time), which is generally thought to be a BOLD fMRI data file. This can be Nifti1 or Nifti2; I can currently read (probably) but not write Cifti files, so if you want to use grayordinate files you need to convert them to nifti in workbench, run rapidtide, then convert back. As soon as nibabel finishes their Cifti support, I'll add that.

The file needs one time dimension and at least one spatial dimension. Internally, the array is flattened to a time by voxel array for simplicity.

Outputs:

Outputs are space or space by time Nifti files (depending on the file), and some text files containing textual information, histograms, or numbers. Output spatial dimensions and file type match the input dimensions and file type (Nifti1 in, Nifti1 out). Depending on the file type of map, there can be no time dimension, a time dimension that matches the input file, or something else, such as a time lag dimension for a correlation map.

Usage:

```
usage: rapidtide2 fmrifilename outputname
[-r LAGMIN,LAGMAX] [-s SIGMALIMIT] [-a] [--nowindow] [-G] [-f GAUSSSIGMA] [-O
oversampfac] [-t TRvalue] [-d] [-b] [-V] [-L] [-R] [-C] [-F
LOWERFREQ,UPPERFREQ[,LOWERSTOP,UPPERSTOP]] [-o OFFSETTIME] [-T] [-p] [-P] [-A
ORDER] [-B] [-h HISTLEN] [-i INTERPTYPE] [-I] [-Z DELAYTIME] [-N NREPS][--
refineweighting=REFINETYPE] [--refinepasses=NUMPASSES] [--
excludemask=MASKNAME] [--includemask=MASKNAME] [--lagminthresh=LAGMINTHRESH]
[--lagmaxthresh=LAGMAXTHRESH] [--ampthresh=AMPTHRESH][--
sigmathresh=SIGMATHRESH] [--refineoffset] [--pca] [--ica] [--refineupperlag]
[--refinelowerlag] [--tmask=MASKFILE][--limitoutput] [--
timerange=STARTPOINT,ENDPOINT]
[--numskip=SKIP] [--sliceorder=ORDER] [--regressorfreq=FREQ] [--
regressor=FILENAME] [--regressorstart=STARTTIME]

required arguments:
    fmrifilename    - the BOLD fmri file
    outputname    - the root name for the output files

preprocessing options:
    -t TRvalue       - override the TR in the fMRI file with the value
                       TRvalue
    -a               - disable antialiasing filter
    --nodetrend      - disable linear trend removal
    -I               - invert the sign of the regressor before processing
    -i               - use specified interpolation type (options are 'cubic',
                       'quadratic', and 'univariate (default)')
    -o               - apply an offset OFFSETTIME to the lag regressors
    -b               - use butterworth filter for band splitting instead of
```

```
                       trapezoidal FFT filter
    -F               - filter data and regressors from LOWERFREQ to
UPPERFREQ.
                       LOWERSTOP and UPPERSTOP can be specified, or will be
                       calculated automatically
    -V               - filter data and regressors to VLF band
    -L               - filter data and regressors to LFO band
    -R               - filter data and regressors to respiratory band
    -C               - filter data and regressors to cardiac band
    -N               - estimate significance threshold by running NREPS null
                       correlations (default is 10000, set to 0 to disable)
    --nowindow       - disable precorrelation windowing
    -f GAUSSSIGMA    - spatially filter fMRI data prior to analysis using
                       GAUSSSIGMA in mm
    -M               - generate a global mean regressor and use that as the
                       reference regressor
    -m               - mean scale regressors during global mean estimation
    --sliceorder     - use ORDER as slice acquisition order used (6 is
Siemens
                       interleave, default is 0 (do nothing))
    --numskip SKIP   - SKIP tr's were previously deleted during preprocessing
                        (default is 0)

correlation options:
    -O OVERSAMPFAC      - oversample the fMRI data by the following integral
                       factor (default is 2)
    --regressor      - Read probe regressor from file FILENAME (if none
                       specified, generate and use global regressor)
    --regressorfreq  - Probe regressor in file has sample frequency FREQ
                       (default is 1/tr)
    --regressorstart - First TR of fmri file occurs at time STARTTIME
                       in the regressor file (default is 0.0)
    -G               - use generalized cross-correlation with phase alignment
                       transform (GCC-PHAT) instead of correlation

correlation fitting options:
    -Z DELAYTIME     - don't fit the delay time - set it to DELAYTIME seconds
                       for all voxels
    -r LAGMIN,LAGMAX - limit fit to a range of lags from LAGMIN to LAGMAX
    -s SIGMALIMIT    - reject lag fits with linewidth wider than SIGMALIMIT

regressor refinement options:
    --refineweighting  - apply REFINETYPE weighting to each timecourse prior
                         to refinement (valid weightings are 'None',
                         'R', 'R2' (default)
    --refinepasses     - set the number of refinement passes to NUMPASSES
                         (default is 1)
    --includemask      - only use voxels in MASKNAME for global regressor
                         generation and regressor refinement
    --excludemask      - do not use voxels in MASKNAME for global regressor
                         generation and regressor refinement
    --lagminthresh     - for refinement, exclude voxels with delays less
                         than LAGMINTHRESH (default is 1.5s)
    --lagmaxthresh     - for refinement, exclude voxels with delays greater
                         than LAGMAXTHRESH (default is 1000s)
    --ampthresh        - for refinement, exclude voxels with correlation
                         coefficients less than AMPTHRESH (default is 0.3)
    --sigmathresh      - for refinement, exclude voxels with widths greater
```

```
                          than SIGMATHRESH (default is 50s)
    --refineoffset      - adjust offset time during refinement to bring peak
                          delay to zero
    --refineupperlag    - only use positive lags for regressor refinement
    --refinelowerlag    - only use negative lags for regressor refinement
    --pca               - use pca to derive refined regressor (default is
                          averaging)
    --ica               - use ica to derive refined regressor (default is
                          averaging)

output options:
    --limitoutput     - don't save some of the large and rarely used files
    -T                - save a table of lagtimes used
    -h HISTLEN        - change the histogram length to HISTLEN (default is
                        100)
    --timerange       - limit analysis to data between timepoints STARTPOINT
                        and ENDPOINT in the fmri file
    --noglm           - turn off GLM filtering to remove delayed regressor
                        from each voxel (disables output of rCBV)

miscellaneous options:
    -c                - data file is a converted CIFTI
    -S                - simulate a run - just report command line options
    -d                - display plots of interesting timecourses

experimental options (not fully tested, may not work):
    --tmask=MASKFILE - only correlate during epochs specified in
                        MASKFILE (NB: each line of MASKFILE contains the
                        time and duration of an epoch to include
    -p                - prewhiten and refit data
    -P                - save prewhitened data (turns prewhitening on)
    -A, --AR          - set AR model order to ORDER (default is 1)
    -B                - biphasic mode - match peak correlation ignoring sign
```