Cyclicity User Manual

Emily Schlafly and Yuliy Baryshnikov

Spring 2016

Contents

1 Quickstart

These are the basic functions you will work with. Start by downloading the sample data to the *data* folder in the *cyclicity* folder. In Matlab, navigate to the *cyclicity* folder and enter the following:

```
addpath([pwd '/code/']);

cd data/
processRois('Pilot*.mat');
output = analyzeCyclicity('processed_Pilot*.mat');

cyclicity_figs(output);
er = evalRatio(output);
pl = permLocations(output,5);
ph = phaseHist(output,[],[],'rois.mat');
trips = makeTriples(output,.8);
```

What follows is an explanation of what each of the functions do and how to use them. Additionally, you can use the `help` command in Matlab to see function usage.

2 Introduction

The code in this package takes a set of time series data and returns a proposed cyclic ordering of the input processes along with some supplementary information that may be useful in analyzing the quality of the algorithm's results or in extracting more detailed information. The purpose of this manual is to briefly introduce the Matlab code of the cyclicity algorithm.

2.1 The Theory

Briefly, the way this works is by assuming that all of the processes follow the same underlying pattern, but offset in time so that the processes have a natural ordering. We can recover this ordering by measuring the algebraic area between each pair of trajectories using Green's theorem and using these areas to construct a matrix from which we get the cyclic ordering. The details are as given in the subsequent sections.

2.1.1 Cyclic vs. Periodic

First, we note that the processes under consideration are cyclic, but not necessarily periodic in the sense that the activity is repeated but not in a regular way. A function $f(t)$ is no t periodic if there exists no T such that $f(t+T) = f(t)$ for all t. Now, let $\phi(\tau)$ be some periodic function and let $t = R(\tau)$, i.e. R is a reparameterization of τ. The function $\phi(t)$ is cyclic but not periodic as long as there exists no $\hat{T}$ such that $R(\tau+\hat{T}) = t+nT$ for all τ and some integer n.

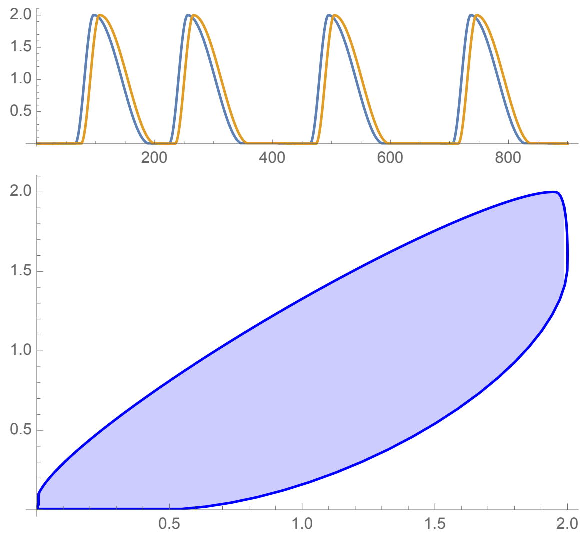

Figure 1: The plot on the top shows the traces of two cyclic but not periodic COOM processes modeled by the function $\gamma_k = \phi(R(\tau) - \alpha_k)$. The plot on the bottom shows the area between the two processes as determined by the integral $\int_I \gamma_l d\gamma_k$. This area is assigned a positive or negative value when the boundary is traced out in a counter-clockwise or clockwise direction, respectively.

2.1.2 COOM

We start by assuming that all of the processes display the same underlying behavior described by the function $\phi(t)$. The individual processes, $\gamma_k(t)$, may be amplified, shifted and noisy so that they may be expressed as

$$\gamma_k(t) = a_k\phi(t - \alpha_k) + N(t)$$

where $N(t)$ is the noise at time t. This is what we call a chain of offsets model because this is the type of behavior one might expect to see if the processes being analyzed behave in such a way that activity in one initiates activity in another, which in turn activates another, and so on. Specifically, the offset is given by the parameter α_k.

2.1.3 The Lead Matrix

To measure the lead-follow relationship between processes we calculate the algebraic area between the traces which can be done using integrals of the form

$$A_{kl} = \oint_I \gamma_l d\gamma_k = \frac{1}{2} \oint_I \gamma_l d\gamma_k - \gamma_k d\gamma_l \tag{1}$$

where I is the time interval. This measures the area shown such that the area is positive if the trajectory is traced out in a counterclockwise direction, and negative if traced in a clockwise direction. This area is calculated for each pair of traces and stored in a matrix A, which we call the **lead matrix**.

2.1.4 Sorting

Assume that the function ϕ is noiseless and composed of a single harmonic (or is approximated by a single harmonic such as the first term of the fourier expansion). Then calculating the integral in Eq. 1 using

$$\gamma_k(t) = a_k \phi(t - \alpha_k) = a_k \sin(t - \alpha_k),$$

we see that the areas in the lead matrix are given by

$$A_{kl} = a_k a_l I \sin(\alpha_k - \alpha_l) = \frac{a_k a_l I}{2i} \left(e^{i(\alpha_k - \alpha_l)} - e^{-i(\alpha_k - \alpha_l)} \right),$$

which says that the matrix A is a rank-2 matrix with basis elements

$$A^+_{kl} = e^{i(\alpha_k - \alpha_l)} \qquad A^-_{kl} = e^{-i(\alpha_k - \alpha_l)}.$$

Note that this integral is invariant with respect to the time reparameterization, which is what allows us to work with non-periodic processes. The eigenvectors of A are found by solving

$$A_{kl} v^l = \lambda v^k.$$

Using the (+) and (-) basis elements, the eigenvectors are found to be complex conjugates of each other with

$$v_1 = \left(a_1 e^{i\alpha_1}, a_2 e^{i\alpha_2}, \cdots, a_n e^{i\alpha_n} \right) \qquad v_2 = \overline{v_1}$$

where n is the number of processes. Hence, sorting the first eigenvector according to phase angle is the same as sorting by offset, which is how we arrive at our ordered result.

3 Toy Data Examples

The main purpose of this architecture is to analyze the cyclicity of time series data from one or more data sets. We start this tutorial by generating a toy data set using the `create_toy` function, which will be explained in more detail later. For now, we can use the following commands to get started:

Listing 1: Listing whatever

```
[toy_data,ss] = create_toy('periodic','rois',30,'shift',10);
cyclicity = analyze_cyclicity(toy_data);

% Generate plots
plot(toy_data.');
figure;
cyclicity_figs(cyclicity);

% Check permutation result
display([ss'; cyclicity.eperms{1}]);
```

Your plot of `toy_data` should look something like Fig. 2. Note that the exact shifts of the trajectories is random, so your plot will not be identical. The figure generated by `cyclicity_figs` should look like Fig. 3 and the permutation result should match identically.

```
ans =

  Columns 1 through 12

     7    27    25     3    22    14     6    11    17     5    18    21
     7    27    25     3    22    14     6    11    17     5    18    21

  Columns 13 through 24

    24     2     1    16     4    19    23    10    12    28     8    20
    24     2     1    16     4    19    23    10    12    28     8    20

  Columns 25 through 30

    29    15    30     9    13    26
    29    15    30     9    13    26
```

The `analyze_cyclicity` function returns a structure array variable with fields `phases`, `eperms`, `evals`, `slm`, and `subject`, each of which is a cell array. In the example above, only a single data set was analyzed so each cell contains only one element, but later we will see how to analyze multiple data sets in which case the cells will store one element for each data set analyzed.

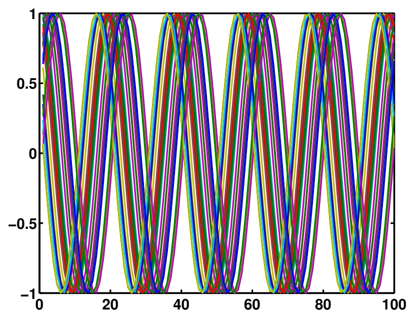

Figure 2: Sample output of `toy_data` variable from Listing 1.

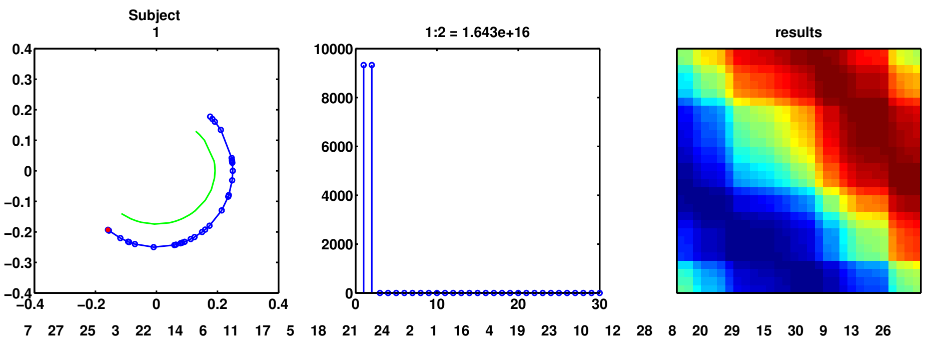

Figure 3: Results from Listing 1.

The contents of each variable can be seen in Fig. 3. The variable `phases` stores the (complex valued) eigenvector associated with the largest eigenvalue of the lead matrix as discussed in Sec. 2.1.4. Note that the elements are not sorted. `eperms` stores the sort order of `phases` and this corresponds to the cyclic ordering of the processes. `slm` stores the sorted lead matrix and `subject` stores the subject number or id.

3.1 Example 1

One problem that immediately arises is that so far we have no way of determining if there are multiple signals in the data set. Consider the set of trajectories created in Listing 2. The first nine traces in the variable `toy_data` will have one underlying function and the last nine, a different function. Note, however, that since both are `'cyclic'` type functions, they are very similar. To see the recreate the results in the tutorial, load the file ‘ex1.mat’ in the *examples* folder (`load('ex1.mat');`).

Listing 2: Data with two signals

```
% Generate trajectories
[toy_data1, ss1] = create_toy('cyclic');
[toy_data2, ss2] = create_toy('cyclic');
toy_data = [toy_data1; toy_data2];

% Analyze
cyclicity_figs(analyze_cyclicity(toy_data));
```

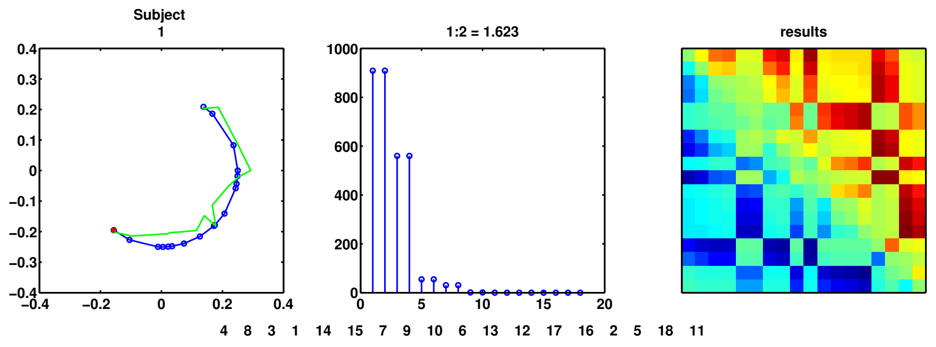

Figure 4: Results of cyclicity analysis on *Example 1* data.

The results should look like Fig. 4. What this shows is a data set with two different (but similar) underlying functions. Looking at the plot of the eigenvalues we see two main eigenvalue pairs and some fuzzy behavior in the lead matrix. We would like to determine which regions are related to one another. One obvious way to do this is to calculate the Fourier expansion of each trace and group the regions by similarity. Since this data is very clean and constructed from a single (time reparameterized) sine wave, looking at the harmonic expansion works fairly well to distinguish groups of activity. In noisy data sets, this will likely be less clear, but may still provide some insight. It is one possible route of further exploration.

3.2 Example 2

Another consideration that so far has shown up in theory more than in practice is in the ordering of the regions. When the algorithm sorts the regions by phase angle, there is a potential for problems if the shifts encompass more than one period of the underlying function. The code in Listing 3 shows how to set up an example that will demonstrate this.

Listing 3: Data with large shifts

```
% Generate data
[toy_data,ss] = create_toy('c','shift',300,'rois',30,'time',1000);
cyclicity = analyze_cyclicity(toy_data);

% Plot results
figure;
plot(toy_data');

figure;
subplot(121)
plot(cyclicity.phases{1}(ss))
subplot(122)
plot(cyclicity.phases{1}(cyclicity.eperms{1}))
```

3.3 The `create_toy` Function

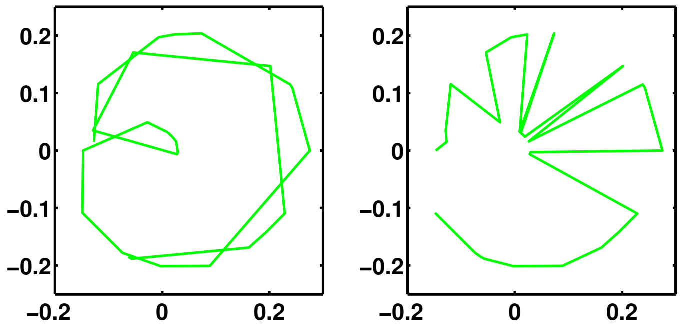

Figure 5: Results when `shift` is larger than a period of the underlying function. The plot on the left shows the phases ordered as they should be according to the amount that each trajectory is shifted. Note how the plot loops around multiple times. The plot on the right shows the phases ordered by the cyclicity algorithm. The algorithm cannot distinguish between the angles $\pi/4$ and $9\pi/4$ so it will not loop around as it should, but rather return a sort that passes over the entire period only once.

```
function [toy_data, ss] = create_toy(data_type, varargin)
Makes sample data. Choose what type of function to look at from
'periodic', 'cyclic', 'gaussian', 'data', 'brownian'.
Note that 'brownian' type trajectories are random unrelated
brownian motions.
Default: 'periodic'.
Optional arguments:
    rois:       number of trajectories. default = 9
    freq:       frequency or number of periods for 'periodic'
                or 'cyclic' type.
                default = 5
    time:       number of time points. default = 100
    noise:      stdev of noise with respect to stdev
                of signal.
                (e.g. noise=0.25 means signal to noise ratio
                ((sigma_signal/sigma_noise)^2) is (1/0.25)^2.
                default = 0
    shift:      maximum time steps between between signals.
                default = 5
    peaks:      number of peaks for 'gaussian' type.
                default = 7
Output:
    toy_data:   specified data set
    ss:         expected sort order
```

The purpose of this function is to create a toy time series data set with desired attributes. The input and output are shown in the usage of the function. Most of the options are pretty clear as to what they do - `time` changes the number of columns of the resulting matrix; `rois` changes the number of rows; `noise` adds gaussian noise so that the ratio of the standard deviation of the noise to the standard deviation of the signal is equal to `noise`. The data types possible are `'periodic'`, `'cyclic'`, `'gaussian'`, and `'data'`. The `'periodic'` and `'gaussian'` types are simply sine waves and sums of gaussians with randomly generated mean values, respectively. The `'cyclic'` type is a sine wave but with the time parameter distorted (reparameterized) to mimic the ideas discussed in Sec. 2.1.4.

The `'data'` type is a smoothed trace from an fMRI data set - it was chosen randomly, not because it showed any desirable properties.

4 fMRI Examples

Now that we have a good idea of how the algorithm works and how to look at results, we can start to look at some real data. This section outlines the process for extracting subsets of data from the full fMRI scan data that Fatima and Sara give us. In general, we do not want to analyze each voxel separately, but rather specific regions. This section will show how to use the `processRois` function to extract regions of interest (ROIs). Then once the data is processed, we will see how to apply the `analyze_cyclicity` function to the processed files in their entirity or to specific ROIs or time intervals.

4.1 Extracting ROIs

Inside the *data* folder are four matrices which have been converted from nifti files to Matlab arrays, but have had no further processing done on them. The first thing we need to do is extract the regions that we are interested in analyzing. To do this we use the `processRois` function. Navigate to the *data* folder and run this command:

```
processRois('Pilot*.mat');
```

This should generate processed versions of each file in the current directory. Note that if you wish to analyze a different set of regions, you may do so by passing a table variable with columns 'regions' and 'coords' (or the name of a file containing the variable).

4.2 Cyclicity of Processed Files

Now that we have the processed data, we can analyze all four files at once using the following command:

```
output = analyze_cyclicity('processed_Pilot*.mat');
```

This should save a file named *cyclicity.mat* in the current directory as well as a struct variable named *output* to the workspace. To see the results type in the following:

```
cyclicity_figs(output);
```

This should open the four figures (one for each subject) as well as save them in a folder called *results_figs*.

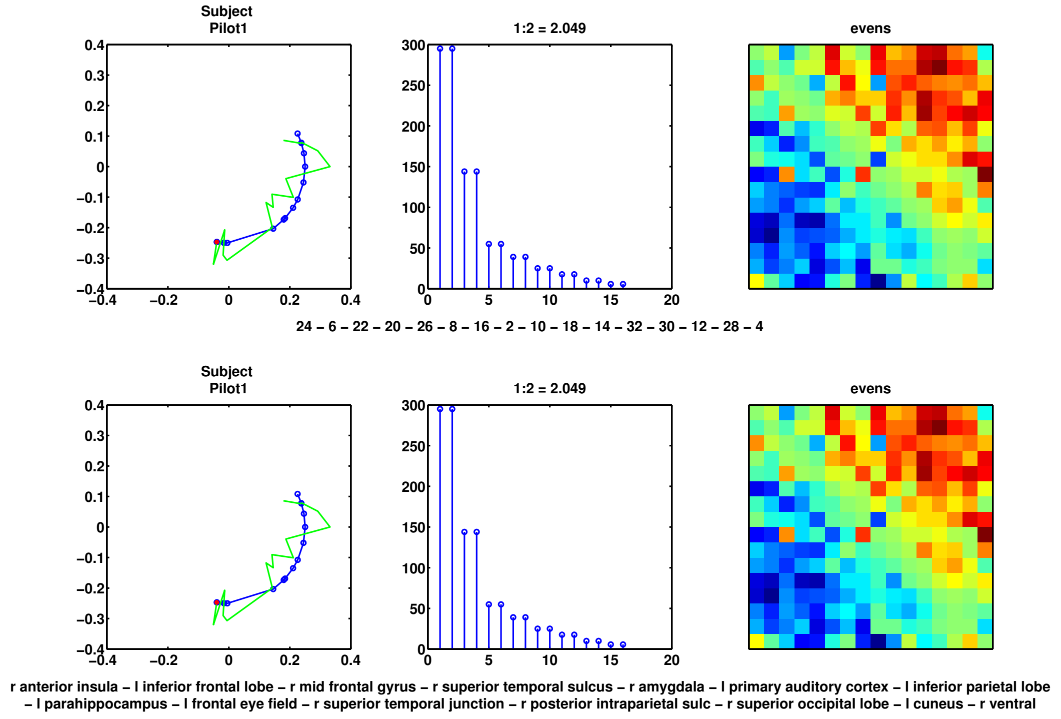

Figure 6: Cyclicity analysis of only even numbered regions with permutation given as region numbers (top) or region names (bottom).

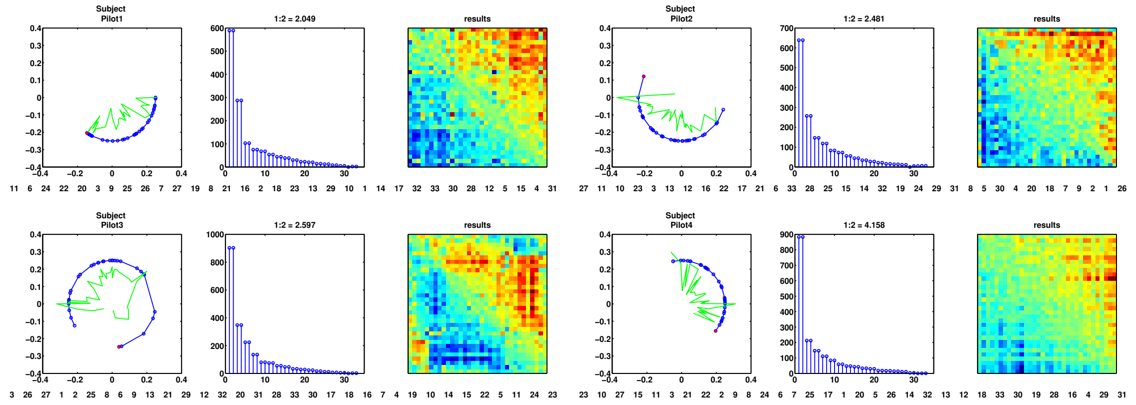

Figure 7: Results of cyclicity analysis on four pilot subjects.

4.2.1 Filter by Region

Note that if you want to look only at certain regions, or lines, of a data matrix you can pass the line numbers as the second argument to `analyze_cyclicity` (Fig. 6). For example, if you wanted to analyze only the even lines you could use the following command:

```
output = analyze_cyclicity('processed_Pilot*.mat',[2:2:33]);
cyclicity_figs(output,'row_labels',[2:2:33],'title','evens');
```

Or, to see the region names on the plots:

```
rois = importdata('rois.mat');
cyclicity_figs(output,'row_labels',...
    rois.regions([2:2:33]),'title','evens');
```

Note that for the commands above, the figures will be saved in a folder called *evens_figs*.

4.2.2 Filter by Time

In several studies, there are intervals of silence and intervals of music so here, we may want to isolate intervals with music and without. In the pilot subjects in the *data* folder, for example, Sara gives us the following supplemental information:

> For all participants, the music/silence breakdown is as follows:
>
> 1-34 = silence
> 35-67 = first song
> 68-100 = silence
> 101-133 = second song
> 134-166 = silence
> 167-199 = third song
> 200-232 = silence
> 233-265 = fourth song
> 266-300 = silence
>
> Note that all of the songs were jazz–no genre differences this time around. For Pilot 1, we didn't have the timing exactly right so we ended up short three images. They just got cut off the end; the timing is still the same.
>
> Also, Pilot 2 has a little bit too much motion in one direction, so I would normally not have bothered to give you that data. In this case, though, since you think you're just using it to set up the manual I included it.

So let's look at only the intervals of silence:

```
silence = [(1:34) (68:100) (134:166) (200:232) (266:297)];
outputSil = analyze_cyclicity('processed*.mat',[],silence);
cyclicity_figs(outputSil,'title','silence');
```

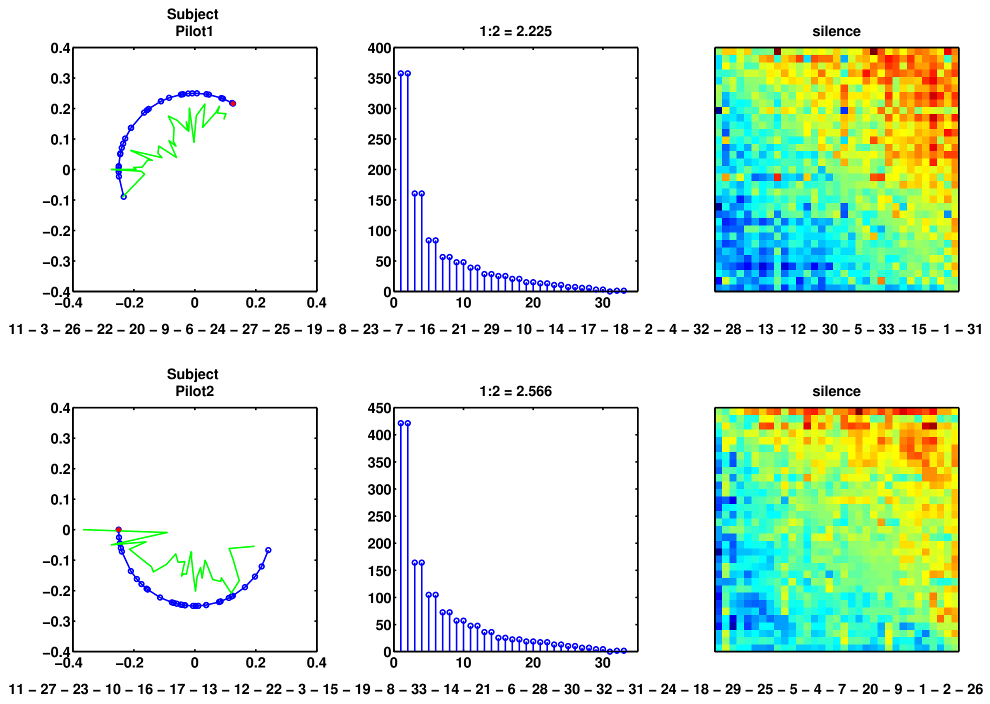

Figure 8: Results from first two subjects during intervals of silence only.

And how about the intervals with music:

```
music = [(35:67) (101:133) (167:199) (233:265)];
outputMus = analyze_cyclicity('processed*.mat',[],music);
cyclicity_figs(outputMus,'title','music');
```

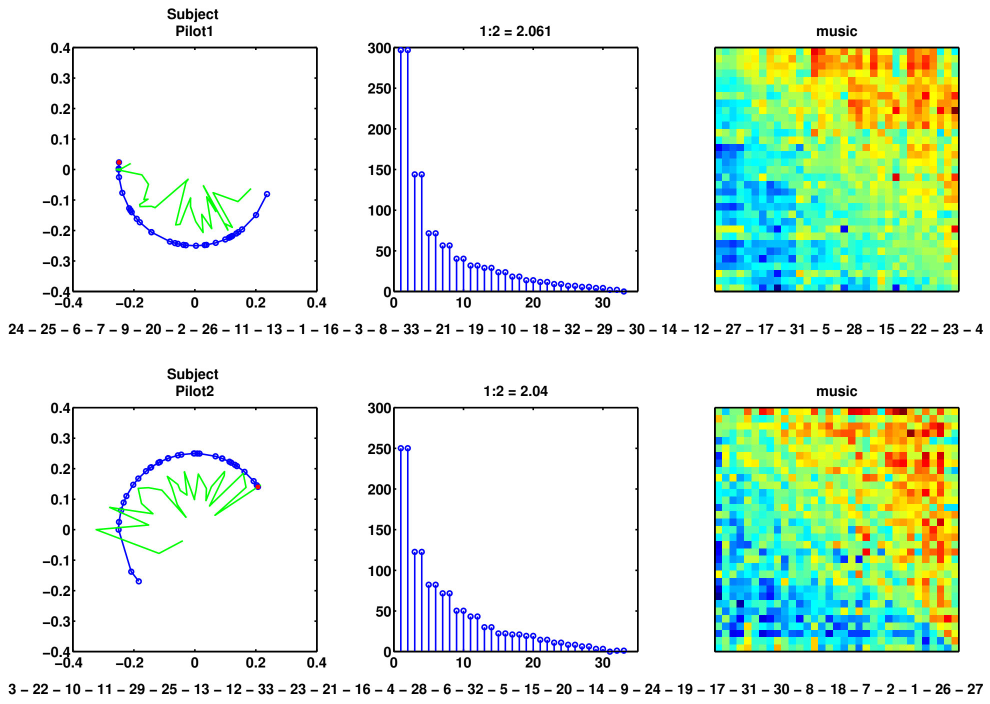

Figure 9:
Results from first two subjects during intervals with music.

5 Looking at Results

This section is meant to address the issue of how to compare and contrast the cyclicity results across subjects and how to interpret results on the whole. The first metric that we look at is the eigenvalue ratios which gives us an idea of the quality of analysis. The other three metrics that we look at are different ways of looking for consistencies in the data.

5.1 Eigenvalue Ratios

```
eval_ratio = evalRatio(dir_search, outid, exp_setup);
```

When considering the quality of the cyclicity results, we look for the ratio between the magnitude of the first eigenvalue pair and that of the second eigenvalue pair to be greater than two. Ratios at or around two are what we would find if the trajectories were completely unrelated brownian motions so we want to see that the data have a more clear relationship than random trajectories.

Let's start by looking at random data:

```
brownian = arrayfun(@(x) ...
  createToy('brownian','time',1000),   ...
  (1:100),'UniformOutput',0);
output = analyzeCyclicity(brownian);
ratios = evalRatio(output);
```

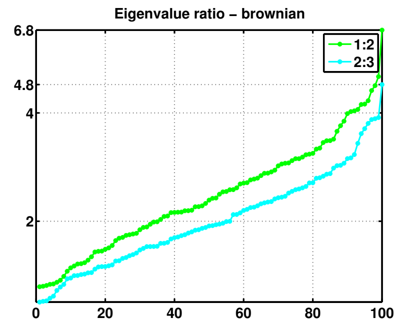

Figure 10: Eigenvalue ratios of brownian motion. Notice how the first and second ratios run parallel to each other and 90% of the ratios are less than 4. We need our data to have higher ratios than shown here, with more differentiation between the first (1:2) and second (2:3) ratios.

In order for our results to be meaningful, we need to see a plot that looks different from the one above. So now let's look at some results that we got from subjects that we analyzed from the Human Connectome Project.

```
ratios = evalRatio('cyclicityHCP.mat');
```

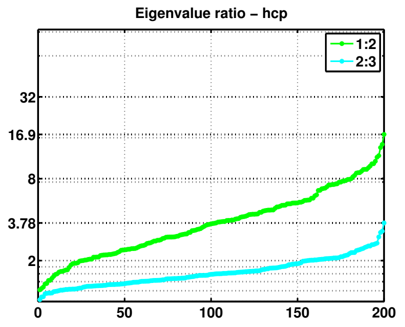

Figure 11: Eigenvalue ratios of HCP subjects. Approximately 40% of subjects have eigenvalue ratios over 4 and the second ratio stays much lower with more than 80% of the ratios less than 2 as compared to only about 50% less than 2 in the brownian case.

In the legend, '1:2' ('2:3') refers to the magnitude of the first (second) pair of eigenvalues to that of the second (third) pair of eigenvalues. We look at this plot in order to determine if we are actually picking up on any real signal, rather than randomness. In the long time limit, the eigenvalue ratios of brownian motion lead matrices approach 2, but in general our data sets are only about 1000 time steps long, so we should get an idea for how a random data set would look in this time frame. Looking at the plot of the eigenvalue ratios for the brownian motion trajectories (Fig. 10), about 90% of the 1:2 ratios are below 4 and about 50% of the 2:3 ratios are below 2, and additionally, the 1:2 and 2:3 ratios run approximately parallel to each other.

In Fig. 11, on the other hand, we see the eigenvalue ratios for fMRI data from the Human Connectome Project (HCP). There are a few differences between Fig. 10 and Fig. 11 which lead us to the conclusion that this method is picking up on an interesting signal: 1.) the 1:2 ratios are, in general, much higher - only about 50% of these are less than 4 compared with about 90% in Fig. 10, 2.) about 75% of the 2:3 ratios are less than 2 (compared with approximately 50%), and 3.) the 1:2 and 2:3 ratios do not run parallel to each other as in the brownian motion analysis.

One approach that we have not pursued, but might be worth investigating is to weed out subjects with low eigenvalue ratios and then do the group analyses described below.

5.2 Order Distribution

```
perm_locs = permLocations(dir_search, numClusters, outid, ...
  region_names, exp_setup)
```

One analysis that we were able to get some interesting insight from was where in the permutation a given region appeared. At one point we narrowed our regions down to those that had the strongest signal across all subjects from the HCP data set and what we saw was that the left primary auditory cortex was consistently showing up very early in the permutation sequence. I never found a way to visualize this that I was completely happy with, so there are a few plots for this one.

The top left plot in Fig. 12 shows the standard deviation of the proportion of time spent in each location for each region. A higher standard deviation means that a region concentrates in only a few locations in the permutation so higher values are more interesting here. The plot in the lower right shows the proportion of subjects in which each region appears in each location. Using this plot in combination with the plot in the upper left corner highlights which regions are more localized and where they are found.

The other two plots give two different views of the same thing. They show the proportion of times each region shows up in a subset of locations. For example, the top right plot says that region 2 was in permutation location 27-33 more than 50% of the time and region 25 was in location 1-7 about 55% of the time. The number of subsets is given by the second input value and the locations are split as evenly as can be (favoring the beginning and ending subsets - the first and last groups have 7 locations, while the middle group has only 6).

Alternatively, use the following to show the region names and export the figure to an .eps file:

```
pl = permLocations('cyclicityHCP.mat',5,[],'rois.mat','eps');
```

5.3 Signal Champions

```
[regions,counts] = phaseHist(dir_search, top_n, outid, ...
  region_names, exp_setup)
```

The signal champions are the regions that consistently have a high absolute values across subjects. The `phaseHist` function makes a histogram of the regions which have phase magnitudes in the top `top_n` for each subject. The higher signal indicates a more clear

```
pl = permLocations('cyclicityHCP.mat',5);
```

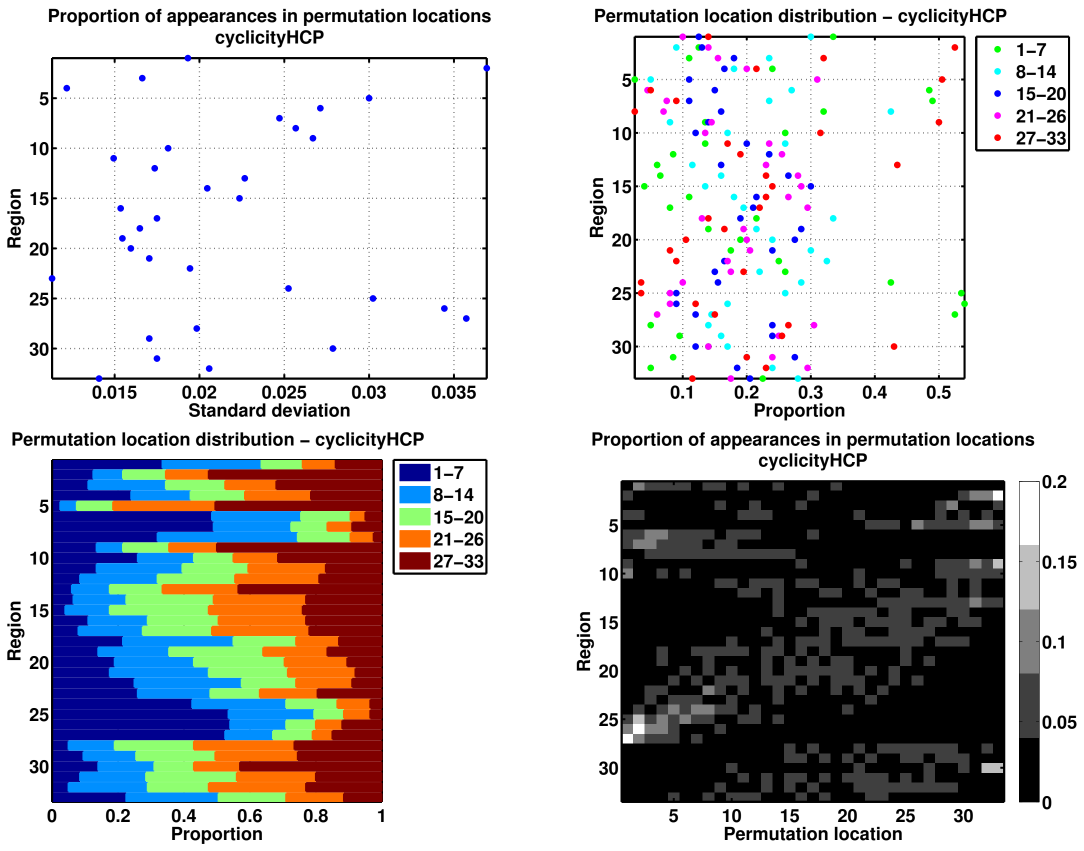

Figure 12: (top left) This plot shows the standard deviation for each region. Regions that stay clustered around one location will have a higher standard deviation than those that are more evenly distributed across locations. (top right) The benefit to this style is that it is very easy to see outliers and approximately where in the permutation they sit. The down side this divides the permutation locations into the number of regions indicated so the location of the cutoff will affect how this plot looks. (bottom left) This is a bar plot of the same values as in the top right plot - no new information, but easier to see the distribution for a certain region. (bottom right) This shows the proportion of times the region appeared in each permutation location. This plot and the top left in combination are probably the most useful.

```
[regions,counts] = ...
  phaseHist('cyclicityHCP.mat');
```

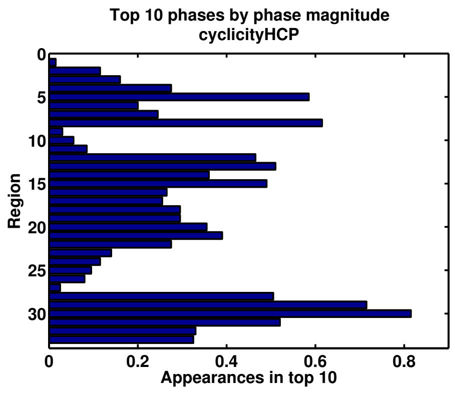

Figure 13: Histogram of regions with phase magnitudes in top 10 for each subject. Looking at this plot, we would conclude that the algorithm works well for regions 5, 8, 29, and 30 so from here, we might decide to focus on these regions.

placement in the permutation, so these should be regions for which the algorithm works well.

5.4 Triples

```
trips = makeTriples(dir_search, thresh, outid, region_names, ...
    exp_setup)
```

This function will return all triples which appear in at least `thresh` proportion of subjects, where `thresh` is in $[0, 1]$. It will also save a text file with the results. If `dir_search` is a cell or regular expression matching multiple files then it will also display a plot comparing which triples appear at or above the given threshold for each data set. Fig. 14 shows triples from cyclicity results from a group with normal hearing and a group with tinnitus (*NH_cyclicity.mat* and *TIN_cyclicity.mat*, respectively).

In addition to the plots, we used the saved results with some additional processing to analyze and compare the results. This is more or less where we left off.

```
trips = makeTriples('*_cyclicity.mat', .8, [], 'rois.mat');
% or equivalently
trips = makeTriples({'NH_cyclicity.mat';'TIN_cyclicity.mat'}, .8, ...
  [], 'rois.mat');
```

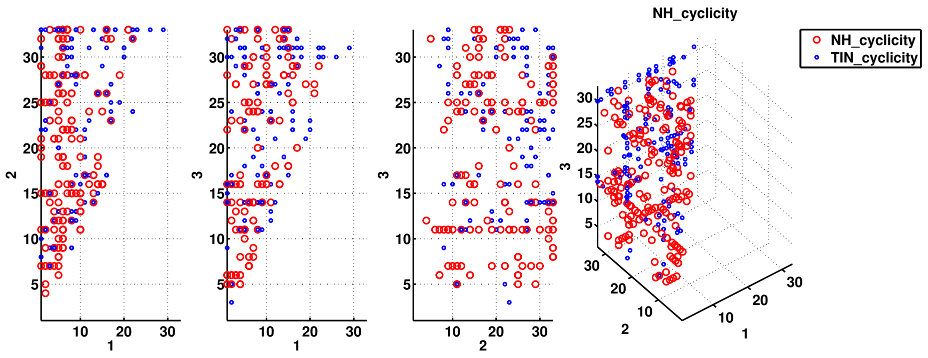

Figure 14: Visual representation of triples comparing common triples in different groups. The plots show a marker wherever there is a triple for each group.