In this tutorial, we will walk you through the steps to visualize and analyze data made available in the BioVis 2014 contest. The data that we will examine includes the following (that you will need to download from the contest site):

- brain cortical surface meshes
- brain region labels (for the 167 brain regions of interest)
- example subject data: 5 subjects, 4 networks (matrices) from each
- 18 contest subjects networks (matrices)
- 30 test networks (matrices)

The primary goal of the contest is to successfully match the 30 test networks to the 18 contest subject networks. Each of the 18 will be represented at least once in the 30.

The primary goal of this tutorial is to introduce young researchers to some basic ideas of brain network analysis using the Python scripting language and Python-based packages.

1 Python Setup

To get started, download the free Anaconda Python distribution for your particular operating system (OS) at: http://continuum.io/downloads. After you have downloaded and installed Anaconda, you can access Python in one of two ways, depending on your OS. On OSX and Linux, you should open a new *Terminal* window and verify that it is using the Anaconda Python by simply typing *$ which python*. You should see "anaconda" in the resulting path. On Windows, in your Programs folder, you will find an *Anaconda* folder and in there an *Anaconda Command Prompt*. That will open up a command prompt window to the installed folder. Regardless of your OS, you will need to navigate (using the *cd* command) to the directory containing the relevant scripts and data for the rest of the tutorial.

From your Terminal/Command Prompt window, install another Python package that we will use in the tutorial:

```
$ easy_install bctpy
```

The *easy_install* command is well-known in the Python community for installing packages from a known repository: pypi.python.org/pypi). The *bctpy* package is an implementation of (part of) the Brain Connectivity Toolbox (BCT), a set of MATLAB scripts that are widely used to analyze brain networks (www.brain-connectivity-toolbox.net).

Before starting the tutorial, we recommend you do the following:

- create a new working directory (folder) somewhere, e.g. *biovis14*
- in *biovis14*, create 4 subdirectories: *anatomy*, *example_networks*, *contest_networks*, *test_networks*
- download and uncompress (unzip) the contest data into the relevant directories

Next, download the scripts and additional data files necessary for this tutorial from: https://github.com/rheiland/biovis2014 and copy them to the relevant subdirectories.

The *roi_legend.txt* from the contest site contains the names and numeric identifiers of the 167 brain regions, some of which are listed here:

```
ctx_lh_G_and_S_frontomargin     11101
ctx_lh_G_and_S_occipital_inf    11102
ctx_lh_G_and_S_paracentral      11103
...
ctx_rh_G_and_S_frontomargin     12101
ctx_rh_G_and_S_occipital_inf    12102
ctx_rh_G_and_S_paracentral      12103
...
Left-Cerebellum-Cortex  8
Left-Thalamus-Proper    10
...
Brain-Stem      16
```

It will be very useful to color these regions so we can better visualize them on the cortex surface. Rather than assign arbitrary colors, we want to provide a meaningful color scheme, one that researchers are accustom to seeing. This information can be found online, mapping the region identifier to the name to a red-green-blue (RGB) color. (There is also a 4th "alpha" component that is unused). We make this available as a text file, *roi_cmap.txt*, in the github repository.

```
11100  ctx_lh_Unknown                   0   0   0    0
11101  ctx_lh_G_and_S_frontomargin     23 220  60    0
11102  ctx_lh_G_and_S_occipital_inf    23  60 180    0
...
```

Finally, if you are on OSX or Linux, you may need to remove two shell environment variables related to how Python finds modules and how shared libraries are found, e.g.:

```
$ unset PYTHONPATH
$ unset DYLD_LIBRARY_PATH
```

2 Anatomy

We begin with something that should be easy to recognize - the human brain. The contest data (anatomy) includes various representations of the surface (cortex) of the brain. These datasets are provided in an OBJ file format. Basically, the OBJ format provides a list of vertices (as x,y,z) and a list of faces (that reference the vertices). Fortunately, one of the packages provided in the Anaconda Python is the VTK library and VTK has an OBJ file reader. So, using just a few lines of code, we can display any of the brain anatomy OBJ files. After the rendering window appears, the user can interactively rotate, zoom, and pan the geometry using the mouse buttons while the cursor is inside the window. All of the Python scripts (.py files) used in the section should be placed in and run from the *anatomy* directory.

```
$ python showOBJ.py single_subject_lh_pial
$ python showOBJ.py average_lh_inflated
```

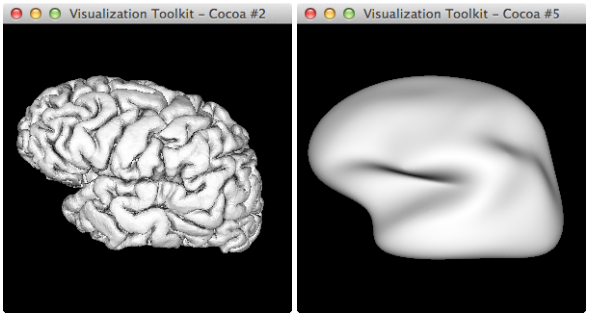

Figure 1: Single subject and averaged inflated cortex

However, as we discussed above, it would be preferable to color the 167 brain regions of interest. To accomplish this, we provide a script that will convert an OBJ file format into a PLY format which allows for coloring the cortex geometry. It relies on the *roi_cmap.txt* data file from the github repository.

```
$ python obj2ply.py single_subject_lh_pial
and
$ python obj2ply.py single_subject_rh_pial
```

This script will generate PLY (.ply) formatted files which VTK can also easily read and display.

```
$ python showBrain.py
```

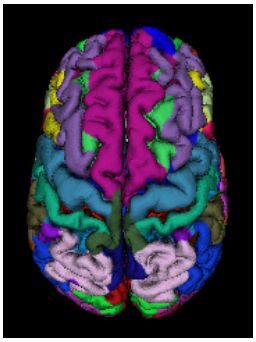

Figure 2: Both hemispheres of the cortex, colored by region

In order to see more of the cortex, we prefer to display the two halves separately, both medial (inside) and lateral (outside) views as in Figure 3 (geometry with no region identifier is white).

```
$ python renderPLY4.py average_rh_inflated
```

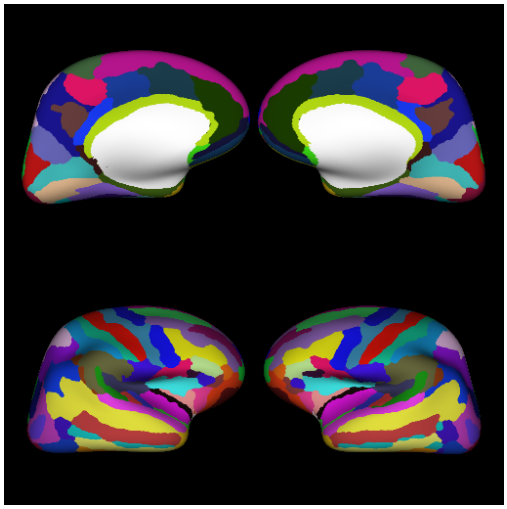

Figure 3: Averaged, inflated hemispheres colored by region

3 Networks, Matrices, and Graphs - Oh My!

The contest provides data that represents a correlation between 167 brain regions. These correlations are intended to represent the brain's "resting state" functional connectivity networks and are represented as a 167x167 connection matrix. Since the primary goal of this contest is to compare pairs of connection matrices, we ask the obvious question: how *does* one compare matrices? As is often the case in mathematics, one starts with a simpler question, e.g. how does one compare two numbers? One obvious answer is to calculate the difference between them; better yet, calculate the absolute value of their difference. Similarly, to see how close two points are in $3D$ space, one would calculate the distance between them: $d = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}$. For two matrices, A and B, one technique for calculating their "closeness" is to use something called the Frobenius norm. While this is not necessarily the best technique for this contest, it is a pedagogical starting point. It is defined as follows:

$$\|A-B\|_F = \sqrt{\sum_{i,j=1}^{n} |a_{ij}-b_{ij}|^2} \quad (1)$$

3.1 Example Networks

In the following script, we perform a Frobenius norm calculation between all pairs of matrices for the example subjects.

```
in .../example_networks:
$ python fnorms_EE.py
Enter a threshold value [-1,1]: 0
```

The output from this script will list all example networks as *person_scan:* followed by a list of all other example networks, sorted (least to greatest) by their pairwise Frobenius norms. We are interested in seeing how well this metric captures the "closeness" of networks from the same person. Therefore, for each person's network, we only show its list out to the last network of the same person. If the Frobenius norm were a perfect measure of network closeness then we would see only the remaining three networks for each person. For person4 and person5, this is indeed true. For person1, it is almost true. For person2 and person3, it is worse. In fact, for network *3_3*, for example, the 3 "closest" networks are from person4.

```
1_1: 1_2 3_1 1_4 1_3
1_2: 1_1 3_1 1_4 1_3
1_3: 1_4 2_2 2_3 1_1 1_2
1_4: 1_3 1_1 1_2
2_1: 2_4 3_2 4_1 4_4 4_3 3_3 4_2 2_3 5_2 5_4 1_2 1_1
     5_3 2_2
2_2: 2_3 1_3 1_4 3_1 3_4 1_2 1_1 5_3 5_1 3_3 5_2 4_4
     5_4 4_3 4_2 2_4 2_1
2_3: 2_2 1_3 1_4 3_1 1_1 3_4 3_3 1_2 4_3 4_4 2_4 5_3
     4_2 5_1 5_2 2_1
2_4: 2_1 4_4 4_1 3_2 4_3 3_3 2_3 4_2 5_2 5_4 1_1 1_2 2_2
3_1: 3_4 1_1 1_2 5_3 1_4 1_3 5_1 3_3 2_3 2_2 5_2 5_4
     4_3 4_2 4_4 4_1 2_4 3_2
3_2: 4_1 2_1 4_4 3_3 2_4 4_3 4_2 5_4 5_2 1_2 1_1 5_3
     2_3 3_1 3_4
3_3: 4_4 4_3 4_2 3_4 3_1 1_2 1_1 2_3 4_1 3_2
3_4: 3_1 3_3 1_3 1_1 1_2 1_4 2_3 5_3 2_2 5_1 4_2 4_3
     5_2 4_4 5_4 4_1 2_4 3_2
4_1: 4_3 4_4 3_2 4_2
4_2: 4_3 4_4 4_1
4_3: 4_4 4_2 4_1
4_4: 4_3 4_2 4_1
5_1: 5_3 5_2 5_4
5_2: 5_4 5_3 5_1
5_3: 5_1 5_4 5_2
5_4: 5_2 5_3 5_1
```

3.2 Contest Subject Networks

We now turn our attention to another suite of datasets from the contest - the "contest subject networks". These are considered the *known* networks, against which we will try to match the *unknown* "test networks" (below). All of the Python scripts used in the section should be placed in and run from the *contest_networks* directory.

```
$ python showMatrix10.py
```

There are some things to notice about this matrix display. First, the script has a default rainbow (blue-to-red) color-map scheme that extends over the range of data values. (It is possible, of course, to

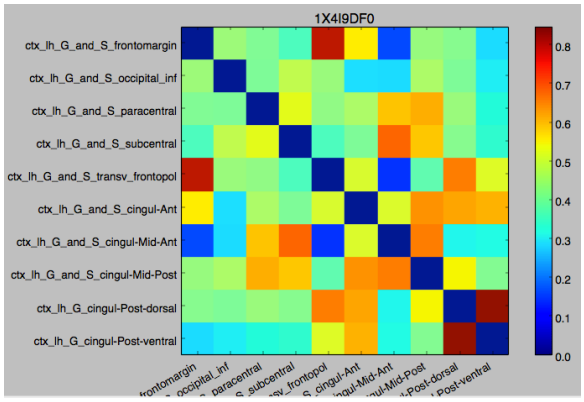

Figure 4: Zoomed in (10x10) region of a connection matrix

modify these defaults in the script). Second, the matrix is symmetric, i.e. it has the same values on "opposite" sides of the diagonal. This is a property of an *undirected* graph.

In addition to using a color-coded matrix to display a connection network, we can also use a graph consisting of nodes and edges, as shown next. One could argue that each visualization technique has its benefits. After performing a modularity analysis technique (below), some of these benefits will become more obvious.

```
$ python showGraph10.py
Enter threshold: 0.5
```

This script displays the following graph, with nodes labeled numerically $0-9$. Note that by thresholding (at 0.5) the network, we will cause those matrix values $<= 0.5$ to be set to 0, thereby indicating there is no correlation between the two corresponding nodes; hence, no edge between them. Otherwise, the edges are color-coded white (low) to blue (high). We can see this representation reflects the matrix representation of Figure 4. Nodes 8 and 9 have a high correlation (dark red edge), as do nodes 0 and 4. (Warning: the script uses an iterative *spring_layout* algorithm to display the nodes of the graph, starting with random initial positions. Therefore your graph may look different than this, but will have the same structure).

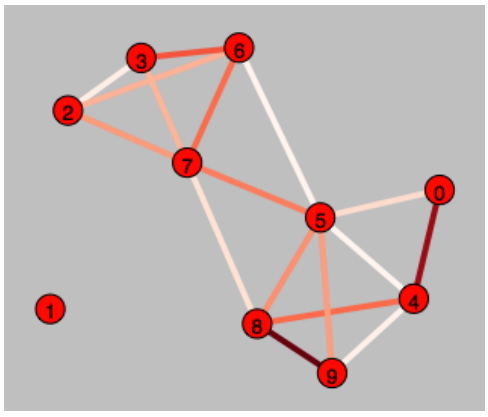

Figure 5: Graph corresponding to matrix in Figure 4

Next, we examine one of the full (167x167) connection matrices.

```
$ python showMatrix.py
```

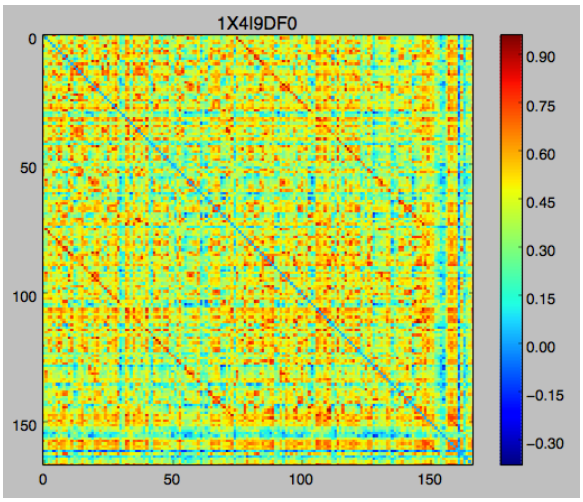

Figure 6: Full connection matrix without thresholding

Notice, once again, the matrix is symmetric about the diagonal and the colormap automatically adjusts to the range of the data. Typically, one would experiment with thresholding the lower bound of the range, e.g. a threshold of 0.0 or 0.5:

```
$ python showMatrixThresh.py
Enter threshold: 0.0
```

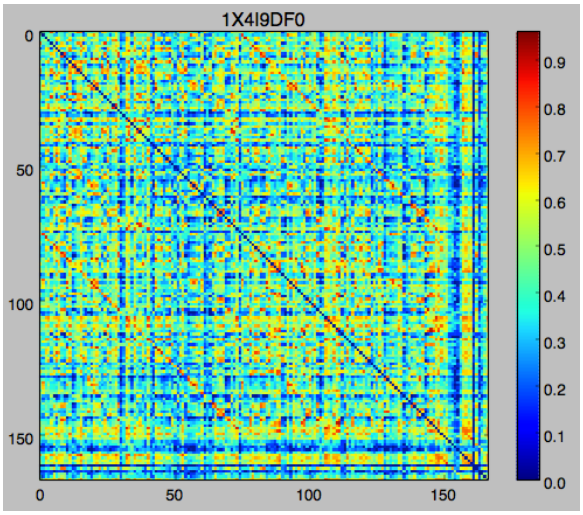

Figure 7: Full connection matrix with threshold = 0.0

Lastly, we apply one of the analysis functions from the BCT to the connection matrix. The *modularity* function determines an optimal structure within a network such that the number of within-group edges are maximized and between-group edges are minimized. In a social network/graph, one could imagine one group of friends (nodes) around person A and another group around person B.

```
$ python modules.py
Enter threshold: 0.0
```

In Figure 8, we see there are three modules (yellow-red blocks): one in the (roughly) upper-left quadrant and two in the lower-right

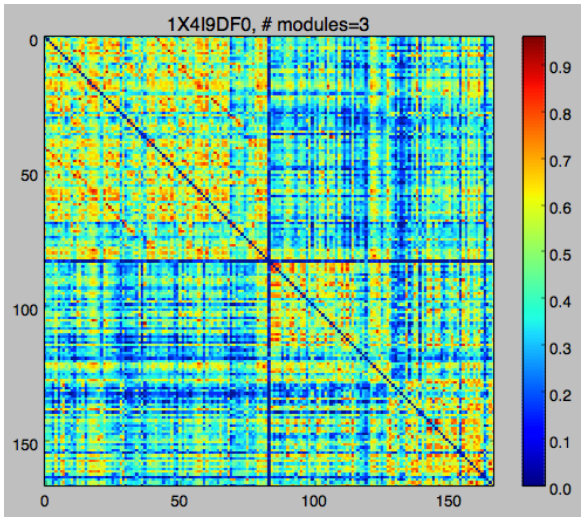

Figure 8: Modularity display of connection matrix (threshold=0.0)

quadrant. If you experiment with different thesholds, you will see that these modules will change.

Next, we want to see what these modules, or groups, look like when mapped onto the cortex (Figure 9).

```
$ python cortexModulesPLY.py average_lh_inflated 0.0
$ python cortexModulesPLY.py average_rh_inflated 0.0
$ python showPLY4.py 1X4I9DF0 average_lh_inflated
```

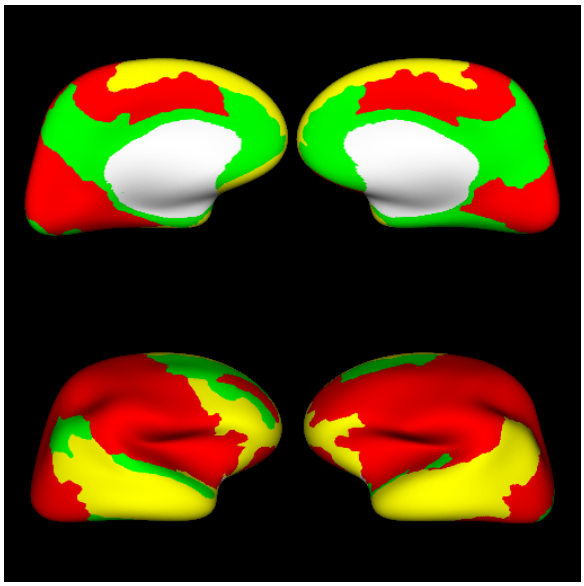

Figure 9: Modularity display on cortex regions

3.3 Test Networks

Now that we have covered some basic ideas of network analysis, we will attempt to perform the primary task of this contest: match 30 unknown (Test) networks to 18 known (Contest Subject) networks.

Our basic approach for matching the 30 unknown to the 18 known matrices is:

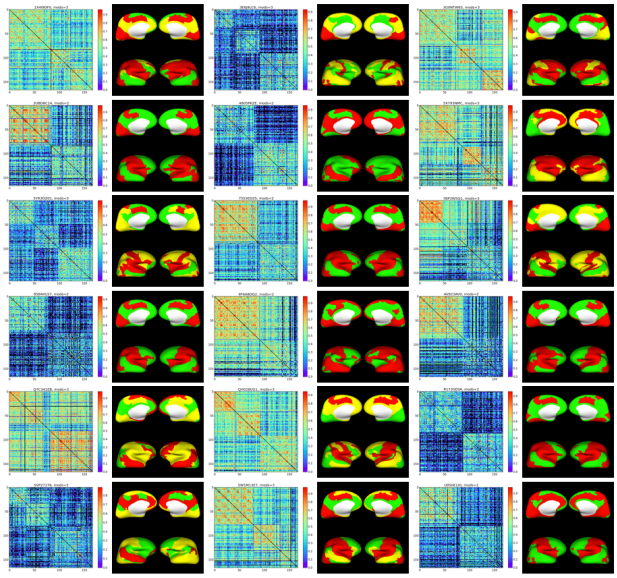

Figure 10: Modularity for all 18 contest subjects

```
For each unknown test network T
  Compute the modularity matrix of thresholded T
  For each known contest subject network C
    Compute the Frobenius norm of T-C
    Compute the modularity matrix of thresholded C
  Sort the Frobenius norms for T
Visually compare the modularity matrices/cortexes for
the lowest norms to determine "best fit".

in .../test_networks:
$ python fnorms_TC.py
Enter a threshold value [-1,1]: 0

T1: C7 C18 C5 C15 C10 C2 C8 C1 C16 C12 C4 C3 C14 C9 ...
T2: C2 C16 C15 C10 C7 C5 C18 C12 C1 C9 C8 C4 C14 C17 ...
T3: C15 C7 C2 C16 C10 C5 C18 C12 C1 C8 C4 C9 C14 C3 ...
T4: C3 C1 C8 C12 C14 C4 C17 C9 C18 C5 C7 C16 C10 C15 ...
T5: C11 C4 C3 C9 C1 C8 C17 C14 C13 C12 C18 C6 C5 C7 ...
T6: C5 C15 C16 C2 C10 C7 C18 C12 C4 C8 C9 C1 C3 C14 ...
T7: C12 C9 C4 C18 C8 C3 C17 C1 C11 C5 C7 C14 C10 C15 ...
...
T23: C4 C12 C9 C8 C3 C18 C1 C7 C17 C5 C15 C14 C10 C11 ...
T24: C12 C1 C3 C8 C4 C17 C14 C9 C18 C5 C7 C11 C6 C15 ...
T25: C17 C1 C8 C3 C14 C7 C5 C18 C15 C16 C2 C4 C10 C9 ...
T26: C14 C1 C17 C8 C3 C7 C5 C4 C18 C15 C10 C6 C2 C16 ...
T27: C18 C10 C7 C16 C2 C15 C5 C12 C1 C9 C4 C8 C17 C3 ...
T28: C2 C15 C16 C7 C10 C5 C18 C12 C9 C4 C1 C8 C3 C17 ...
T29: C16 C15 C10 C7 C5 C2 C18 C12 C8 C4 C9 C1 C3 C17 ...
T30: C2 C15 C16 C7 C10 C5 C18 C9 C12 C4 C8 C1 C3 C17 ...
```

4 Conclusion

We have presented a brief overview and hands-on tutorial for brain network analysis targeted, primarily, for an audience of inexperienced students and researchers. The tutorial uses freely available Python packages and scripts. Our hope is that students will feel empowered and be motivated to continue experimenting with other parameters and more data analysis techniques. While we have limited this tutorial to the discussion of thresholding and modularity, there are many more concepts to explore in network analysis.

Acknowledgements

The authors wish to thank Roan LaPlante for assistance with *bctpy*.