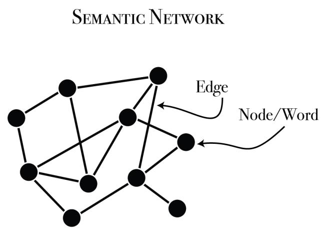
Calculating graph metrics

March 4th, 2020

Opening Files

First we need to make sure we have all the necessary files. Make sure you have MATLAB (at least version 2017a) running on your computer. After this we'll need to download the *Brain Connectivity Toolbox (BCT)* (https://sites.google.com/site/bctnet/) and have it saved in the same directory. Additionally we'll need the actual network files in the same location, which are the node labels and adjacency matrix for the semantic network and the co-occurrence network.



To summarize, we should see the following in our working directory:

- BCT folder
- Semantic Network (ex. *semanticNetworkAdjmat.txt*)
- Semantic Network words (ex. *semanticNetworkNodeLabels.txt*)
- Co-occurrence Network (ex. *coOccurrenceNetwork.txt*)
- Co-occurrence Network (ex. *coOccurrenceNodeLabels.txt*)
- Semantic analyses script *computingGraphMetricsSemanticAdj.m*
- Co-occurrence analyses script *computingGraphMetricsCooccurrenceAdj.m*

Running the code

Both analyses scripts are similar, so we will work through examples with the semantic analyses script.

1. Open MATLAB
2. On the left side you will find the Current Folder. This is the directory from which MATLAB will operate. If this is not the folder with all of the above files, use the "Browse for folder" button (opening folder icon) to the left of the file path at the top, then choose the correct folder.
3. Check to make sure all of the files listed above are now in your Current Folder. If not, return to step 2.
4. Double click to open the *computingGraphMetricsSemanticAdj.m* file.

At this point you can hit the Run button (big green triangle at the top) to run the entire script. This will execute everything in the script and create multiple figures which will output in new windows. The rest of this ReadMe goes into the details of the script sections and functions.

This script has different sections just like the script on Python that built your networks originally. Before the main part of the script begins, we prepare the workspace by clearing all variables that may be hanging around and add the *BCT* folder to your working paths so we can use all of the functions inside later.

```
clear
addpath(genpath('BCT'))
```

Section 1 - Read in the Data

We want to read in both the network and the word labels, so first we read the .txt files and create variables **wordsTable** and **adj**. However sometimes reading from .txt files can lead to some extra odd characters in the words so we see the first for-loop removes extra spaces, comments, and other characters that we do not want in our word list.

You'll find the co-occurrence analyses script has one extra step where we threshold the graph:

```
threshold = 1;        % Change this value to keep only stronger edges. How does
    your network change?
adj(adj>=threshold) = 1;
```

In this step we keep all edges that have values above the set threshold, **threshold**, so that we recover a binary (unweighted) graph.

At the end of this section the important variables we have created are:
nWords: number of words
wordList: 1×nWords cell with a word in each entry
adj: nwords × nWords matrix

You can always check what variables you have in the MATLAB Workspace usually located on the right of your screen (it can be moved anywhere).

Section 2 - Create Graph

Next we take our adjacency matrix and creates from it a graph using `G = graph(adj);`. The rest of this section is dedicated to plotting the graph.

Important variables created in this section:
G: graph object created from the adjacency matrix.

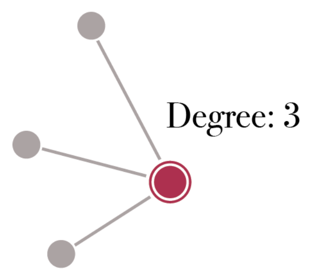
Section 3 - Compute relevant graph metrics

Next we actually compute graph metrics. These are quantitative reports of graph structure, each from a different perspective. Those we will focus on today are the degree, clustering coefficient, betweenness centrality, and modularity.

Node Degree

The degree of a node k is the number of edges emanating from this node. When we consider the adjacency matrix, recall each column corresponds to a node and any values with 1 in that column means there is an edge to another node. Then to get the degree of each node, we need the sum of that column.

```
degreeVector = sum(adj);
```



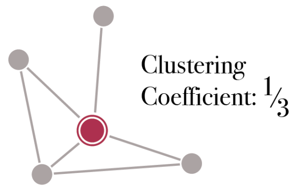
Now **degreeVector** contains the degree for each node in the graph. That is, the degree of node n is $degreeVector(n)$. Next we might want to know which nodes have the highest degree, to see which corresponding words are most connected. The following code will first find the maximum degree, then print those words corresponding to nodes with the maximum degree (since there might be more than one!).

```
% We can look at the maximum degree, minimum degree, etc...
maxDegree = max(degreeVector);
maxDegreeWords = wordList(degreeVector == maxDegree);
fprintf('The max degree is %i.\n and the word(s) with this degree are:\n',...
maxDegree)
for n = 1:length(maxDegreeWords)
fprintf('%s\n',maxDegreeWords{n})
end
```

Finally the script will plot the degree distribution using the *histogram* function.

Clustering Coefficient

The degree is a very node-centric measure. In other words, it does not care about the node's neighborhood, only the node itself and it's own edges. The node could be connected like a star, or within a very dense web of neighboring connections. The clustering coefficient can help us get an idea of the node's neighborhood.



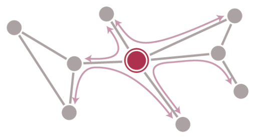
The clustering coefficient is defined by counting edges between a node's neighbors. If two of a node's neighbors are connected, then this forms a triangle. The clustering coefficient counts is the ratio of the number of such triangles to the number possible, the latter of which is determined by the node degree. In the example on the left, the red node has 4 neighbors, so it could be a part of 6 triangles. However, it is actually only a part of 2 triangles. Thus the clustering coefficient is 1/3.

In the code we calculate this with a function from the Brain Connectivity Toolbox:

```
% Next we can compute the clustering coefficient using the BCT...
clusteringVector = clustering_coef_bu(adj);
fprintf('The average clustering coefficient is %f\n',...
mean(clusteringVector))
```

Betweenness Centrality

While the clustering coefficient helps us understand the node's local environment, we still do not really have a sense of how the node fits within the entire graph. For example, if one was to traverse the graph, would we generally pass through this node? Or is it more in the outskirts of the graph?



The betweenness centrality looks at all shortest paths between pairs of nodes, then calculates the ratio of shortest paths that go through a particular node to all shortest paths.

In the code, we calculate the betweenness centrality using the BCT:

```
% ... and betweenness centrality...
betweennessVector = betweenness_bin(adj);
fprintf('The average betweenness centrality is %f\n',...
mean(betweennessVector))
```

Questions: Does a node with large degree necessarily have a high clustering coefficient or betweenness centrality? The script will generate plots to look at how these metrics relate to one another in your network.

The final part of this section reorders the words by decreasing degree, clustering, or betweenness. The script prints out the top ten, but you can use the created variables **wordList_bydegree**, **wordList_byclustering**, and **wordList_bybetweenness** to look at the entire ranked list.

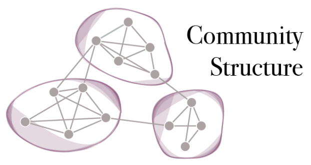
```
% Which nodes have the largest degree, clustering, and betweeness?
% Here we calculate the indices that give the sorted metric. So the first
% entry of inds_* will be the index of the largest entry of that vector.
[~,inds_degree] = sort(degreeVector,'descend');
[~,inds_clustering] = sort(clusteringVector,'descend');
[~,inds_betweenness] = sort(betweennessVector,'descend');

% To order the words by metric, we now use these indices
wordList_bydegree = wordList(inds_degree);
wordList_byclustering = wordList(inds_clustering);
wordList_bybetweenness = wordList(inds_betweenness);

% Print the top words of each
fprintf('\nThe top ten words with highest degree are:\n')
for i = 1:10; fprintf('%s ',wordList_bydegree{i}); end
fprintf('\nThe top ten words with highest clustering are:\n')
for i = 1:10; fprintf('%s ',wordList_byclustering{i}); end
fprintf('\nThe top ten words with highest betweenness are:\n')
for i = 1:10; fprintf('%s ',wordList_bybetweenness{i}); end
```

Section 4 - Community Structure

The final measure we will examine the "meso-scale graph structure", that is, the larger blocks that make up the whole graph. The large question we ask here is does the graph break into multiple tightly-connected subgraphs that seem to stand on their own?



This is called the community structure of the graph, and each densely connected subgraph a community. Before asking how the graph might be subdivided into communities, we should first ask if the graph even breaks up well into communities. This idea is summarized by the modularity, which quantifies how nicely we can divide up the entire network into communities.

In the code, we use the modularity function from the BCT which returns both the modularity (**modularity**) and a vector describing how to break the graph into communities (**communityVector**), so that node n belongs to community $communityVector(n)$.

```
[communityVector,modularity]=modularity_und(adj,1);
nCommunities = max(communityVector);
```

Finally we plot the network with nodes colored by their community. Please note the colors are chosen randomly each time, so if you don't like the colors in one run, simply re-run the code and you will get new colors.