MATLAB for Network Analysis

APM 598

Professor Armbruster

16 October 2014

Peter Chotras

Brady Gilg

Jaime Lopez

Dan Magee

<u>What does the program do well?</u>

MATLAB is a 4th generation programming language meant for use in engineering and applied mathematics. While it is not designed for network analysis or visualization, it has many properties that make it a useful tool in such applications.

One of MATLAB's strengths is the fact that a large number of people use it. As a result of this, there are large amount of premade network analysis and visualization m-files available online. Once an m-file is found, it takes little effort to put the file to use (usually just dragging and dropping). This makes it very easy to aggregate a large amount of algorithms in MATLAB. Table 1 demonstrates this by comparing network analysis programs/libraries and a compilation of m-files found in thirty minutes of googling.

<u>Table 1:</u> Comparing a compilation of easily found m.files with free network analysis programs and libraries.

	MATLAB	**Net Workbench**	**BOOST**	**MAVisto**
Node/Degree Metrics	X	X	X	X
Basic Centrality Metrics	X	X	X	
PageRank	X	X		
Motif Identification				X
Random Graph Generation	X	X	X	X
Small-World Graph Generation	X	X		
Visualization	X	X	X	X
Diffusion/Flow on Networks	X		X	

If an m-file needs to be modified or created, this can be done easily in MATLAB. Code in MATLAB is simple to read and write; using the programs and creating new programs does not require a large amount of programming knowledge. Since MATLAB was designed for linear algebra, this is especially true for matrix-based calculations. The way matrices are handled is fairly intuitive and hassle-free. This makes MATLAB a convenient platform for network analysis since many network algorithms incorporate the adjacency matrix.

However, there are some potential disadvantages to using MATLAB. A major disadvantage is cost. If one does not have access to MATLAB through a university license, an individual license must be purchased for about $2000. In that case, MATLAB a terrible option compared to free programs. MATLAB may also be a poor choice if one has concerns about the efficiency or runtime of the program. Programs created in a less abstract language, like C++, will likely have better runtimes.

How easy is it to adjust it for specific needs?

There are a number of specific toolboxes developed by programmers to address needs of individuals attempting to analyze specific complex adaptive systems. The MathWorks, Inc., which created MATLAB, offers a number of toolboxes which users can download to use MATLAB for specific needs. One example is their toolbar designed for the field of bioinformatics, called Bioinformatics Toolbox^{TM}. Users can treat gene sequences as a network, where nodes represent proteins and directed arrows represent an interaction between those proteins (the nature of the interaction depends on the object of study). This toolbox can be used to perform a number of field-specific tasks, such as next-generation sequencing, which identifies patterns in protein sequences. Additionally, MathWorks recommends using this toolbox with its toolbox that assists with the analysis of artificial neural networks (called Neural Network Toolbox^{TM}), which trains neural networks to recognize patterns given a large enough amount of data [4*].

More generally, a number of third parties offer toolboxes which, while not designed for a specific type of complex adaptive system, includes a number of helpful .m files for analysis. For example, MIT Strategic Engineering has a toolbox for analyzing complex adaptive systems using MATLAB [3*]. Downloading this toolbox helps adjust MATLAB for specific needs by allowing the user to utilize programming that analyzes a variety of aspects of a complex adaptive system. The toolbox includes multiple measures for centrality, allowing for a more appropriate analysis of the specific system. For example, the toolbox's ability to measure closeness centrality is useful for the analysis of many social networks, whereas its ability to measure eigenvector centrality could be more insightful for an analysis of the World Wide Web. The toolbox also measures a network's mean degree, node and edge betweenness, and clustering coefficients, and includes other features, such as finding conversion and distance measures.

FInally, MATLAB also has programs which can analyze dynamic networks [2*]. For example, Lev Muchnik has created a package called Complex Network Package for MATLAB. The tutorial for this package demonstrates how to use the program to analyze a random dynamic network that models disease spread. In the example, nodes represent individuals and a directed link from node *j* to node *i* represents individual *i* being infected by individual *j*. Using the toolbox, somebody attempting to analyze disease spread under certain conditions can construct programs in MATLAB to create randomly generated networks with certain pre-determined properties, such as the number of nodes, the number of initially infected individuals, and the rate at which an infection spreads. After generating the complex adaptive system, the toolbox can be used to plot results, including the number of infected individuals over time, changes in the rate of infection over time, and the average degree of the graph over time.

Can you extend the package and write additional code for it?

Matlab is a programming language, so in principle it can compute anything that any other language can. As an example, let's try to replicate the triad profiles seen in the paper "Superfamilies of Evolved and Designed Networks".[6*]

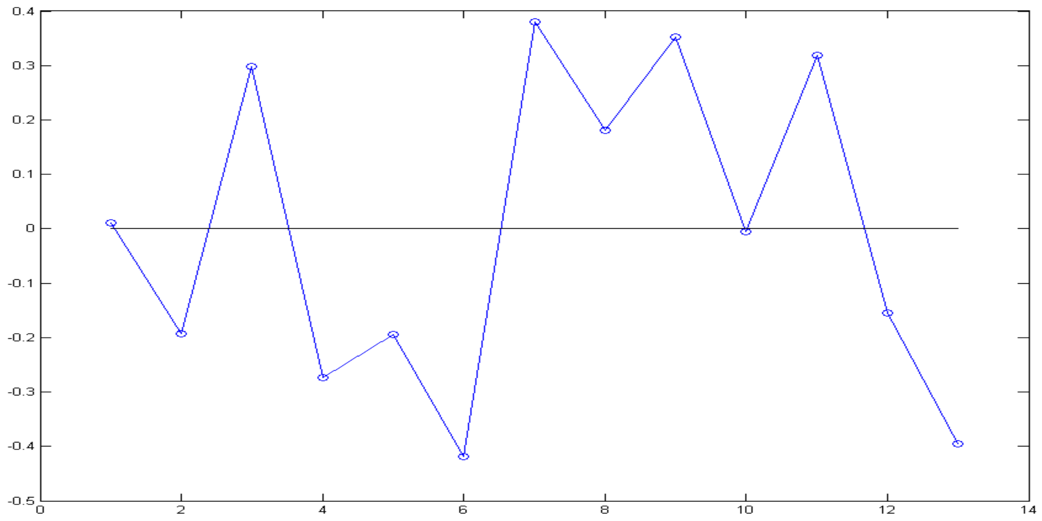
The program used to generate the profiles in the article is “mfinder”, a closed source Windows executable written previously by some of the authors of the paper. To replicate this program in Matlab, we start by looking at the methodology in the article.

The profile is defined as the normalized Z score of the number of triads of each type in the target network compared to a random network of the same degree distribution. Specifically, $Z_i = (Nreal_i - <Nrand_i>)/std(Nrand_i)$, where i indexes over the 13 possible triads, Z is a 13 dimensional vector, $Nreal_i$ is the number of triads of type i in the target network, and $Nrand_i$ is a vector of the number of triads of type i for some number (not specified in the article, I chose 100) of randomly chosen networks with the same degree distribution as the target. Then, Z is normalized to produce the triad profile. These operations can be completed quickly with built in Matlab commands.

```
for i = 1:13
        z(i) = (Nreal(i) - mean(Nrand(:,i)))/std(Nrand(:,i));
end
profile = z./(norm(z));
```

Now the question is how to generate $Nreal_i$, which requires counting the number of triads in a given network, and $Nrand_i$, which requires generating random networks with the same degree distribution as the target network and also counting their triads. Luckily, Matlab has a large number of functions already written, and using Google I was able to find a webpage from Brown University[7*] with the file “gibbs.m” which outputs a random adjacency matrix with the same degree distribution as an input matrix. The code was written in response to the paper in question, so it seems appropriate.

The last function required is one to compute the number of triads of each type of a given network. Since I was unable to find a script to do this, I needed to write it myself. This was accomplished simply by looping over the possible triads and using nested if statements to determine which of the 13 triads they fit into if any. Combining the built in commands, the code scavenged from the internet, and the code I wrote myself is very easy because they all become functions in Matlab. After about 2 hours of research and searching, and another 2 hours of coding, I was able to generate a triad profile for a given network. As an example, below is the profile for a random network of 50 nodes.



Choose one particular algorithm, find its code and explain the algorithm that is coded.

One useful algorithm provided by the MIT Strategic Engineering Research Group [1] is the node betweenness centrality measure [5]. It takes in an adjacency matrix (n x n) of the distances between nodes and outputs a betweenness vector (n x 1) for which lists the betweenness for each node. It begins by counting the number of nodes, *n*, in the adjacency matrix and then creating a matrix (n x n) of shortest paths *spaths* with each entry set to infinity. With this action, the algorithm proceeds to determine the shortest path going through each node.

This step requires 3 nested loops: the first will create an adjacency matrix, *adjk*, for each value of *k* from 1 to *n* - 1. The matrix *adjk* is set to the value of *adj^k.* This is done such that the square matrix of distances, *adj*, has every combination of distances evaluated. The algorithm will then instantiate two loops for *i* and *j* for each value from 1 to *n*. If *adjk*(*i*, *j*) of this matrix is greater than 0 then *spaths*(*i, j*) is set to the smaller value of *spaths*(*i*, *j*) and *adjk*(*i, j*). During the first loop, each entry of *spaths* is infinity so obviously each minimum will be set to the value of *adj1*(*i, j*). After each value of *i* and *j* has been tested (each entry in the matrix *adjk*), the first loop will then create a new adjacency matrix *adj2*, and then repeat the process for each *entry*. Essentially, for each node in the set the algorithm checks every possible combination of distances to each other node, including backtracking, to find the shortest path possible path between two nodes. At the conclusion of these nested loops, the *spaths* matrix will consist of the shortest distance between (*i, j*).

At this point the algorithm creates *btwn*, a (n x 1) array and sets each value to zero. The last main part of the algorithm is to adjust this *btwn* array to reflect accurate values of the betweenness of each node. It does this by nesting two more loops; the first of which calls a different MATLAB algorithm, shortest_pathDP.m (also in the MIT package), and the second of which adjusts the *btwn* vector. The first loop's purpose is to call shortest_pathDP.m which returns the optimal path from node *j* to each other node. The second loop adds to the betweenness score of the node being analyzed. During the second loop, if *i* and *j* are at the same value, the remainder of the second loop will be skipped since no shortest path exists, obviously, between the same node. Otherwise, it creates the vector [*J_ji*, *step_ind*] where *step_ind* is the number of independent steps taken to get from *j* to *i* and *J_ji* is the path length from *j* to *i*. It is set to the smallest value of any node to *j* from the shortest_pathDP.m function.

Then *route_ji* is created (the route from node *j* to the destination in *i* steps) and is assigned to this vector. So for the first loop, it is looking for each node *i*=1 step away in the network. The *btwn* array is then assigned to its current value plus 1/*spaths*(*i, j*). The repetition of this loop results in the *n*th node of the *btwn* array being the sum of how many shortest paths run through node *n*. This continues for each node in the set. Once the nested loops have concluded, the final step is to divide each value of the *btwn* array by *n*-choose-2. This step is done because we are looking at the betweenness of each node relative to the total possible shortest paths which exist. The algorithm is complete and the *btwn* array is finalized. Each value in *btwn* is the betweenness centrality of the corresponding node.

References

1. Bounova, G., de Weck, O.L. "Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles", Phys. Rev. E 85, 016117 (2012).
2. Lev Muchnik (2013). Complex Networks Package for MatLab (Version 1.6). http://www.levmuchnik.net/Content/Networks/ComplexNetworksPackage.html
3. MIT Strategic Engineering (2011). “Matlab Tools for Network Analysis (2006-2011)”, http://strategic.mit.edu/downloads.php?page=matlab_networks (last updated March 5, 2014)
4. MathWorks (2014). “Bioinformatics Toolbox^{TM} User’s Guide,” http://www.mathworks.com/help/pdf_doc/bioinfo/bioinfo_ug.pdf
5. MIT Strategic Engineering (2011). “Matlab Tools for Network Analysis (2006-2011)”, http://strategic.mit.edu/docs/matlab_networks/node_betweenness_slow.m (last updated October 13, 2009)
6. “Superfamilies of Evolved and Designed Networks”, Milo et al, Science March 2004. https://math.la.asu.edu/~dieter/courses/APM_598/Alon_science.pdf
7. https://wiki.brown.edu/confluence/pages/viewpage.action?pageId=71884521