Network Models and Basic Network Operations

Notes by Daniel Whitney
January 3, 2008

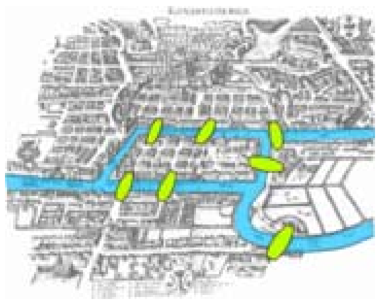
Introduction

Networks are arrangements of things and connections between them. The things can be physical, such as cities or people, or they can be abstract such as tasks in a process like designing a product. The connections can be physical, like roads between cities or blood relationships between people, or they can be abstract like "sister cities," friendships between people, or information flows between design tasks. In general, networks can represent systems, where the systems consist of things and their inter-relationships. Network representations can themselves be quite abstract, conveying little about the interconnected things or the connections. Or they can be more specific, identifying or differentiating between the things and between the connections. The things can be given names, levels of importance, size, etc., and the connections can be given lengths, capacities, probability of breakdown, etc.

Networks can be represented graphs containing nodes (the things) and links (the connections). If the links are two-way they are called edges; if they are one-way they are called arcs. There is a large body of theory called Graph Theory that deals with the properties of graphs.

Graphs and networks have a long history, dating to the time of the mathematician Euler. He represented the bridges of the city of Königsberg as a graph in 1736 and proved that it is impossible to walk over all the bridges without walking on at least one of them twice. See Images removed due to copyright restrictions.

Figure 1.



.

Images removed due to copyright restrictions.

Figure 1. The Bridges of Königsberg. Left: The map. Center: Abstraction of the map. Right: Network representation. The blue nodes represent land and the black links (edges) represent the bridges. Source: http://en.wikipedia.org/wiki/Seven_Bridges_of_Königsberg

Since that time, graphs have been used to represent transportation and communication networks, [Ahuja, Magnanti, and Orlin] social relationships, [Wasserman and Faust] ecological food webs, and partial order relationships such as priorities or constraints, among many other things. In turn, transportation networks have been analyzed to find shortest paths or maximum capacity paths. Social networks have been analyzed to find social groups or cliques, leading individuals, or patterns of relationships. Mathematical constraints have been analyzed to find means for solving equations. Physical systems and supply chains have been analyzed to find “modules.” Food webs and communication networks have been analyzed to determine their health and robustness.

Graph theorists and network analysts have developed a number of metrics to characterize networks. Along with these metrics are algorithms for calculating them. We will use Matlab to do these calculations where-ever possible. However, there is no complete set of calculating tools available, in part because new metrics and analytical methods keep getting invented. On the web one can find many matlab toolboxes containing graph analysis tools. In addition, there are several closed toolboxes. Prominent among these are UCINET and Pajek. UCINET is a set of social network analysis tools and is available for a modest fee from http://www.analytictech.com/. It also contains Netdraw, a convenient network visualization tool. Pajek is available free from http://vlado.fmf.uni-lj.si/pub/networks/pajek/ It provides a number of network analysis tools and drawing capabilities. Pajek and UCINET run on PCs and Pajek runs on Linux. Neither runs on a Mac. Matlab is available for every platform. Network analysis in Matlab is limited in three ways. One is the size of networks that can be stored. The second is the speed of the calculations. The third is the lack of a really good visual representation tool. Pajek in particular boasts of its ability to contain and analyze huge networks. Both Netdraw and Pajek have very good representation tools. Only recently have huge networks become available for analysis (the internet, Facebook, both having millions of nodes). Most real networks are much smaller. Matlab has the advantage that we can write our own algorithms or borrow from toolboxes on the Matlab web site or elsewhere on the web.

Network Representation Using Matlab

Networks can be represented conveniently using a matrix called the adjacency matrix. The rows and columns are numbered to represent the nodes, and a mark, usually the number 1, is placed at the (i,j) intersection if there is an arc from node i to node j. If the link is two-way then a mark is also placed at intersection (j,i). The matrix is then said to be symmetric. Intersections where there is no link contain the number 0. A graph with all two-way links is called undirected and the corresponding adjacency matrix is symmetric. If the graph has all one-way links, the graph is called directed and the adjacency matrix is asymmetric. A graph with both one-way and two-way links is called mixed. The adjacency matrix is asymmetric.

Matlab is an array-oriented system and is well-suited to represent matrices. In Matlab, anything contained in […] is an array. Arrays can be created by typing them in directly or they can be read from files. The files can have different delimiters between the entries, such as commas, spaces, or tab characters. Matlab usually can determine what the delimiter is, so one can use the command *dlmread* to bring an array in and use *dlmwrite* to write it out. Arrays stored as comma-delimited (.csv) can be created using Excel. Tab-delimited arrays can be created in Word or a text editor. [1]

Matlab can also read and write Lotus 1-2-3 formats called wk1 using *wk1read* and *wk1write*. This is convenient because UCINET can also read and write wk1 files. Excel matrices can be pasted directly into UCINET and UCINET can read Excel files.

Here is an example of array creation directly in Matlab:

```
>> row1=[1 2 5 8]
row1 =
    1     2     5     8
>> row2=[10 6 5 9]
row2 =
   10     6     5     9
>> matrix1=[row1;row2]
matrix1 =
    1     2     5     8
   10     6     5     9
>> matrix2=[row1 row2]
matrix2 =
    1     2     5     8    10     6     5     9
```

Table 1. Making Arrays in Matlab

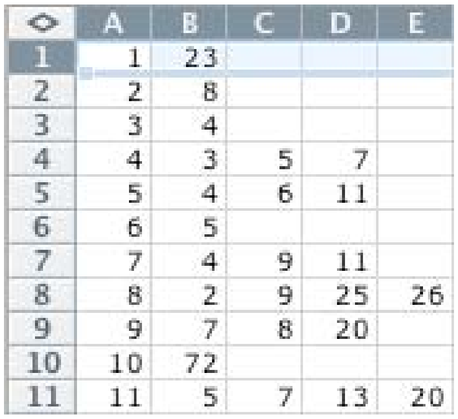
The above shows that the elements of a row are separated by spaces. Also, rows can be stacked by separating them with ;. Rows stacked this way must be of the same length. (Note: If you don't want Matlab to print on the screen the output of any line, end the line with a semicolon (;)).

Data Input

Typing in a large matrix is really tedious and prone to errors. Two standard alternatives exist, called the node list and the edge list. A node list is a list of each node and the nodes that it connects to with outgoing links. An edge list is a list of the edges containing the nodes linked by them, with the "from" node listed first. It is fairly easy to take a drawing of a network, number each node, and prepare a node list, checking off each edge on each node as one goes through the network.

[1] Any time you want to understand what a matlab routine or command does, just make up a simple vector or matrix having 5 or 10 elements and use the routine or command on it. Make sure you choose something simple so that you can see by yourself what the answer should be or so that you can trace the results and understand them.

Figure 2 contains examples of a node list and an edge list:

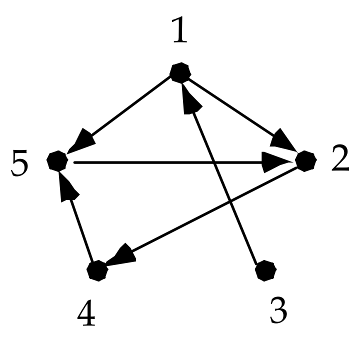


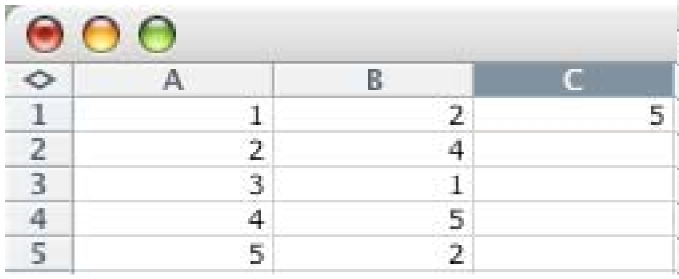
◇	A	B	C	D	E
1	1	23			
2	2	8			
3	3	4			
4	4	3	5	7	
5	5	4	6	11	
6	6	5			
7	7	4	9	11	
8	8	2	9	25	26
9	9	7	8	20	
10	10	72			
11	11	5	7	13	20

1	2
1	8
1	43
1	49
1	50
1	51
1	52
1	53
1	54
1	55
2	3
2	9
2	44
2	56
3	4
3	10
3	45
3	57
3	58
4	5
4	11
4	46
4	59
4	60

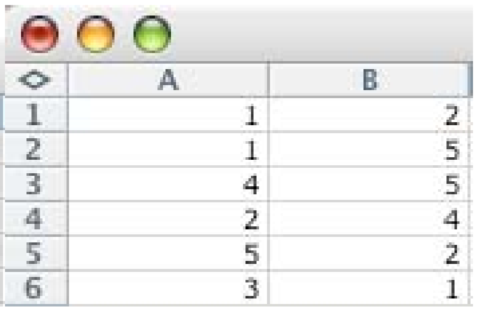
Figure 2. Left: A node list put into Excel. The source node numbers are in column A and the destination nodes are listed to the right. An arc extends from the source to each destination. In this case the network is symmetric: Node 2 is a source for node 8 and node 8 is a source for node 2. Right: An edge list. In the first column are source nodes. Each edge is listed separately and its destination node is in the second column. Thus node 1 is the source for 10 different nodes. This list is evidently not symmetric, since node 2 is a destination for node 1 but node 1 is not a destination for node 2.

Once Matlab has the nodelist or edgelist, one can use routines called *adjbuildn* or *adjbuilde* to create an adjacency matrix from the lists.





◇	A	B	C
1	1	2	5
2	2	4	
3	3	1	
4	4	5	
5	5	2	



◇	A	B
1	1	2
2	1	5
3	4	5
4	2	4
5	5	2
6	3	1

Figure 3. Top: Example Matrix. Bottom: Its nodelist and edgelist. The nodelist is saved as testmatrix.csv while the edgelist is saved as testmatrix2.csv

Here is the way the nodelist is read in by Matlab and converted by *adjbuildn* to the adjacency matrix testadj:

```
>> test=dlmread('testmatrix.csv')
test =
    1    2    5
    2    4    0
    3    1    0
    4    5    0
    5    2    0
>> testadj=adjbuildn(test)
testadj =
    0    1    0    0    1
    0    0    0    1    0
    1    0    0    0    0
    0    0    0    0    1
    0    1    0    0    0
```

Table 2. Using adjbuildn

Here is the way the edgelist is read in by Matlab and converted by *adjbuilde* to the (same) adjacency matrix testadj2. The routine *adjbuilde* creates it as a sparse matrix, to save space. The command *full* is used next to convert it to a full matrix using the Matlab command *full*.[2] The sparse representation holds and displays only the non-zero entries, along with the (i,j) where they occur. Since most networks have far more nodes than edges, most adjacency matrices have only a few non-zero entries, so sparse representation saves processing time and storage space when there are very many nodes, say thousands or tens of thousands.

```
test2=dlmread('testmatrix2.csv')
test2 =
    1    2
    1    5
    4    5
    2    4
    5    2
    3    1
>> test2adj=adjbuilde(test2)
numnodes =
    5
num_edges =
    6
test2adj =
   (3,1)       1
   (1,2)       1
   (5,2)       1
   (2,4)       1
   (1,5)       1
   (4,5)       1
>> test2adj=full(test2adj)
test2adj =
    0    1    0    0    1
    0    0    0    1    0
    1    0    0    0    0
    0    0    0    0    1
    0    1    0    0    0
```

Table 3. Using adjbuilde

Since the network is directed, the resulting adjacency matrix is not symmetric.

Several other routines are available for helping input and output. For example, *adj2pajek* converts an adjacency matrix to the input format needed by Pajek.

[2] Some Matlab functions do not work on sparse matrices. The error messages can be cryptic but usually mention “sparse” so that is your clue. For example, you can’t make a histogram of sparse data.

Simple Matrix Operations

Here are a few useful facts about Matlab operations on matrices:

A’ = the transpose of A
sum(A) adds up each column and stores the result as a row vector
sum(A’) adds up each row and stores the result as a row vector
sum(sum(A)) adds up all the entries in A
length(x) counts the number of entries in vector x
size(A) lists the lengths of the dimensions of matrix A. If A is 3x4 then $size(A) = 3 \quad 4$. If you want the size of A’s kth dimension, use $size(A,k)$. In the above example $size(A,2) = 4$.

Table 4. Matlab Operations on Matrices

Logical Operations and Extraction of Submatrices

find(x logical expr) returns the subscripts of entries in x that satisfy the logical expression using linear indexing. (Linear indexing gives every entry one subscript.) The answer, when placed in x(ans), returns the values of x corresponding to the subscripts.
length(find(x logical expr)) tells how many entries in x satisfy the logical expression
[i,j]= find(x logical expr) returns the i and j subscripts of entries in matrix x that satisfy the logical expression
unitize(A) makes all the non-zero entries in A equal to 1.
Here are some examples using row1 and testadj from Table 1: >> find(row1>1) ans =   2   3   % the 2nd and 3rd entries of row1 satisfy the condition that they are > 1[3] >> row1(ans)   % the value of “ans” is whatever was last output ans =   2   5   % these are the 2nd and 3rd entries of row1

Table 5. Use of Logical Expressions

Note that the result of a logical operation like find is a logical matrix, not a numerical matrix, so you can’t do arithmetic on the result. This makes it useless for many of the calculations described below. You can convert a logical matrix B into a numerical one simply by adding 0: $B = B + 0$. The *unitize* routine makes use of this:

[3] % is the delimiter that begins a comment in Matlab.

```
function unitize=unitize(A)
% unitizes a matrix, makes all non zero entries = 1
unitize = A > 0; % entries of unitize are logical 1 where A
> 0
unitize = unitize + 0; % now entries of unitize are
numerical 1 where previously they were logical 1
```

The syntax is $Au = unitize(A)$

The following examples show how to extract a range of entries from an array:

```
>> testadj(1,:)          % the : represents the entire row, so it says to extract row 1
ans =
    0    1    0    0    1
>> testadj(:,2)          % the : represents the whole column, so it says to extract column 2
ans =
    1
    0
    0
    0
    1
>> testadj(2:5,2:5)              % this extracts rows 2 – 5 and columns 2 - 5
ans =
    0    0    1    0
    0    0    0    0
    0    0    0    1
    1    0    0    0
```

Non-adjacent elements of a matrix may be extracted as follows:

```
>> A=[1 2 3 4 5;6 7 8 9 10;11 12 13 14 15;16 17 18 19 20;21 22 23 24 25]
A =
    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
   16   17   18   19   20
   21   22   23   24   25

>> idx=[3,5]
idx =
    3    5
>> A(idx,idx)
ans =
   13   15
   23   25
>>
```

The array idx is used as an argument in matrix A to extract all the entries in A that are combinations of the individual entries in the array idx. So the (3,3), (3,5), (5,3), and (5,5) entries of A are extracted by the above code. For example, if A were a network adjacency matrix, then A(idx,idx) would be the submatrix containing the links between the nodes listed in idx, in this case, between nodes 3 and 5.

Basic Facts About Undirected Graphs

Undirected graphs have symmetric adjacency matrices. The number of rows in the matrix is the number of nodes, while the number of non-zero entries in the matrix is twice the number of edges. If a row and column of the matrix have no entries then the corresponding node has no edges and is called an *isolate*. The number of edges on a node, called the *average nodal degree*, is denoted by k and the average number of edges per node in the network is alternately denoted $<k>$ (which is typical statistics notation) or by z (this is typical physics notation, used by the many mathematical physicists who have entered the network field.) If n is the number of nodes and m is the number of edges, then

Equation 1 $$<k> = z = \frac{2m}{n}$$

Each network has a *degree sequence*, which is simply a list of the nodes giving their respective degrees: $D=[d_1,d_2,\ldots]$. Sometimes this list is sorted with the largest degree first. In an undirected graph, the sum of the members of the degree sequence is an even number: $\sum_k d_k = 2m$. The degree sequence D of an undirected network whose adjacency matrix is A can be obtained in matlab as

Equation 2 $$D = sum(A)$$

From the above equations,

Equation 3 $$sum(sum(A)) = 2m$$

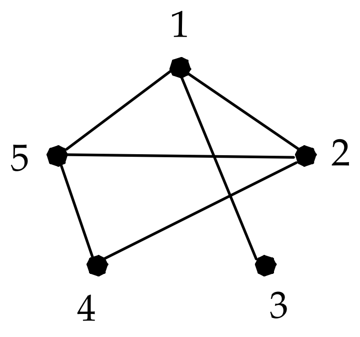


Figure 4. Example Undirected Network

The function $kvec(A)$ finds the degree sequence using Equation 2. The syntax is $kvA = kvec(A)$ where A is the adjacency matrix and kvA is the name of the resulting degree sequence. For the network in Figure 4, the (sorted) degree sequence is $D = [3\ 3\ 3\ 2\ 1]$, whose sum is 12. For this network $<k>= 12/5 = 2.4$. A list can be sorted in Matlab using its sort routine. To sort in descending order, use $Dsorted = sort(D,'descend');$ To sort an adjacency matrix so that the first row has the node with the most edges, use the routine sortbyk: $Asorted = sortbyk(A);$ To visualize a degree sequence, use Matlab's plot routine. The result of doing this on a random matrix with 1000 nodes and $<k>= 5$ is shown in Figure 5:

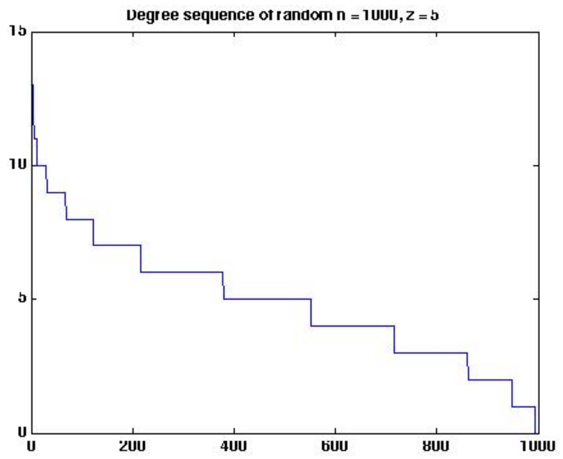


Figure 5. Plot of the Sorted Degree Sequence of a Random Matrix

Not all sequences of numbers are valid degree sequences, even if they add to an even number because the edge list and the node list must be consistent. The Erdös-Gallai theorem performs the necessary calculations and the function *isgraphic* implements them. The syntax is *isgraphic(D)* where D is the degree sequence to be tested. If it passes, the answer is 1 (meaning true in Matlab notation), otherwise the answer is 0 (false).

Network Metrics

In this section we calculate basic measures of a network, such as the number of nodes, edges, average nodal degree, and others defined as we go.

The routine *numnodes* calculates the number of nodes:

```
function numnodes=numnodes(A)
%finds number of nodes in A including isolates
numnodes=size(A,1);
```

If you want to exclude any isolates, use *numnonisonodes*:

```
function nodes = numnonisonodes(A)
```

```
% counts non-isolated nodes in a matrix
A=unitize(A+A');
nodes=min(length(find(sum(A')~=0)),length(find(sum(A)~=0)));
```

To find the number of edges, we showed above that it is just *sum(A)*. But if the network is mixed (having some directed and some undirected links) then we can find the sum of directed and undirected by making the adjacency matrix symmetric first:

```
function numedges = numedges(A)
%counts edges in matrix A, symmetric or not
%works when not all nodes have edges
AT=A+A';
numedges=sum(sum(AT~=0))/2;
```

To see if a matrix is symmetric, use Matlab's *issymmetric* function: $issymmetric(A)$. The answer will be 1 if A is symmetric and 0 if not.

The clustering coefficient is a metric that seeks to measure the extent to which nodes are linked to each other. It is one of many metrics that seek to do this in one way or another. Its origins are in the social network community where the goal is to see how many of your friends are friends; that is, how many of the nodes linked to a node are linked to each other? The essence of the calculation is to see how many potential triangles are in fact complete. Newman's review paper shows that there are in fact two similar ways to calculate this, as indicated in Figure 6.

Figure 6. Illustrating the two ways to calculate the clustering coefficient [Newman]

The Matlab routine library for ESD.342 contains two routines called respectively *clustEq3* and *clustEq5*. The text of the first is below, while the opening lines of the second follow. *clustEq5* returns the Newman value if it is invoked simply as *clustEq5*. It returns three items if invoked as *[clustNewman,clustSchneiderman,clustbynode] = clustEq5(A).* The last item is Newman's equation 5, the local clustering coefficient of each node. If the matrix is large then this will be a big list. To suppress printing of this list, put a ; at the end of the call.

```
function clust3=clustEq3(B)
% finds clustering coefficient according to Eq 3 in Newman
review paper
tr=0;
for i = 1:size(B,1)
if sum(B(i,:))>1
tr(i)=nchoosek(sum(B(i,:)),2); %finds number of connected
```

```
triples of each node
end
end
triangles3=sum(.5*diag(B^3)); %finds 3*number of triangles
clust3=triangles3/sum(tr);
```

```
function [clustNewman,clustSchneiderman,clustbynode] =
clustEq5(A)
%  calculates the clustering coefficient according to Eq 5
in Newman review paper
%
% adapted from code by Ed Schneiderman of Johns Hopkins U.
% Schneiderman calculates the average clustering coefficient
ONLY for those
% vertices where the number of neighbors is >1. Newman
calculates it for all vertices.
```

Another interesting property of a graph is the average shortest distance between nodes. Included in this is the concept of network diameter, which is the largest of shortest distances between all pairs of nodes. Calculating true shortest distances can be computationally intensive, requiring a shortest path algorithm. Simpler methods can be use if the graph obeys some assumptions. Chief among these is that the distance between adjacent nodes is taken to be unity. This will not do for geographic networks like roads or railways but it works for social interactions or other unweighted binary relationships.

If adjacent nodes are separated by unit distance, then the distances between all pairs of nodes can be found by taking powers of the adjacency matrix. If a unit entry in $a(i,j)$of adjacency matrix A means that nodes i and j are directly adjacent, then a non-zero entry in $a2(i,j)$ of $A2 = A^2$ means that there is a two-step path from i to j. Similarly, a non-zero entry in $a3(i,j)$ of $A3 = A^3$ means that there is a three-step path from i to j. (The clustering coefficient routines find triangles, three-step paths from node i to itself, by finding non-zero elements in the (i,i) or diagonal entries in A^3.) If we observe every element $ak(i,j)$ in successive $A^k$and note the value of k at the moment each element becomes non-zero for the first time, we can tell when the first path of length k has been established between nodes i and j. This first path is the shortest, since no shorter one appeared during the process. The routine *distmat* uses this method, which is described on page 151 of [Wasserman and Faust]. The first few lines of *distmat* are:

```
function [diagdist,avgpath,diam] = distmat(B)
% Routine to find the distance matrix of symmetric adjacency
matrix B
% and then calculate the average distance between all pairs
of nodes.
% Matrix B cannot have any isolated nodes and must have
entries = 0 or 1.
```

Diagdist is the matrix of distances, which can be large if the network is large. Avgpath is the average of all shortest paths and diam is the diameter of the network. For the network in Figure 4, the result of applying *distmat* is

```
[d,e,f]=distmat(As)
d =
    0    1    1    2    1
    1    0    2    1    1
    1    2    0    3    2
    2    1    3    0    1
    1    1    2    1    0
e =
   1.5000
f =
    3
```

distmat can be used on any network, but analytical solutions can be written if the network has a regular structure. Leonard Miller, late of NIST, found such solutions for a variety of regular lattices in the interests of understanding cell phone network performance, where the number of hops between antennas or central stations is important in signal quality. For example, for a rectangular grid of NxM nodes, the average hop distance $\overline{m}$ between any pair of nodes is

Equation 4 $$\overline{m} = \frac{M+N}{3}$$

Function *distmat* works only if the network is connected. If a network is disconnected then one has to find all the isolated components and separately calculate their diameter and average path length. Before discussing how to find these components, we will show how to find out if a network is connected. The relevant routine is called *isconnected* and its use on the network in Figure 4 gives

```
isconnected(As)
ans =
   1
```

The first few lines of the routine are

```
function yn = isconnected(g)
% isconnected(g) -- determine if g is a connected graph
% Gives the right answer only if g is undirected
% Works by asking if node 1 can be reached from every node
```

If the network is directed, then routine *isconnectedasym* must be used. The first few lines of this routine are

```
function ynn = isconnectedasym(g)
% isconnectedasym(g) -- determine if g is a connected graph
% Gives the right answer when g is directed
% Works by applying the method of isconnected successively
to every node.
```

As promised, here is the routine that finds disconnected components, written by ESD PhD Dr Mo-Han Hsieh. Here are the first few lines:

```
% [componentCount] is used to generate the component partition
of a matrix.
% Its input is the adjacency matrix, A.
% Its output are partition, componentList, mainNum, and
singletonNum.
% /partition/ maps each node to different components.
% /componentList/ is a list of components and the number of
nodes in the
% components.  Its format is: [component ID, number of nodes in
it].
% /mainNum/ is the number of components which has members of at
least two, and
% /singletonNum/ is the number of singletons.

function
[partition,componentList,mainNum,singletonNum]=componentCount(A)
```

Here is an example disconnected network with 2 components:

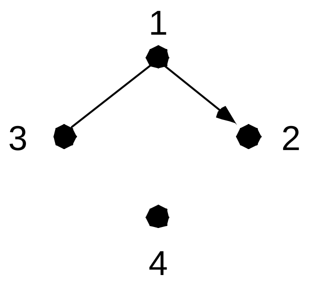


Figure 7. Example network with two components

Here is how to use *componentCount* to find out about this network:

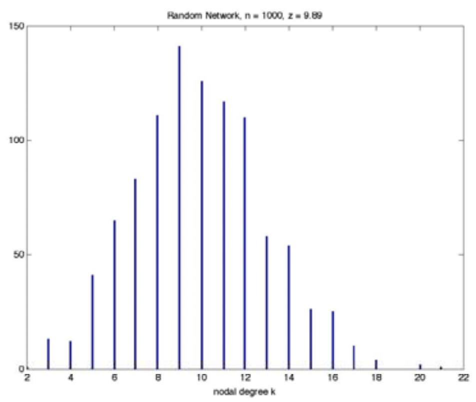
```
AC =
   0    1    1    0
   0    0    0    0
   1    0    0    0
   0    0    0    0
>> [aa,bb,cc,dd]=componentCount(AC)
aa =
   1
```

```
    1
    1
    2
bb =
    1    3
    2    1
cc =
    1
dd =
    1
>> maxcompsize=max(bb(:,2))
maxcompsize =
    3
maxcompnum=find(bb(:,2)==maxcompsize)
maxcompnum =
    1
>> maxcompnodes=find(aa==maxcompnum)
maxcompnodes =
    1
    2
    3
maxcompgraph=AC(maxcompnodes, maxcompnodes)
maxcompgraph =
    0    1    1
    0    0    0
    1    0    0
```

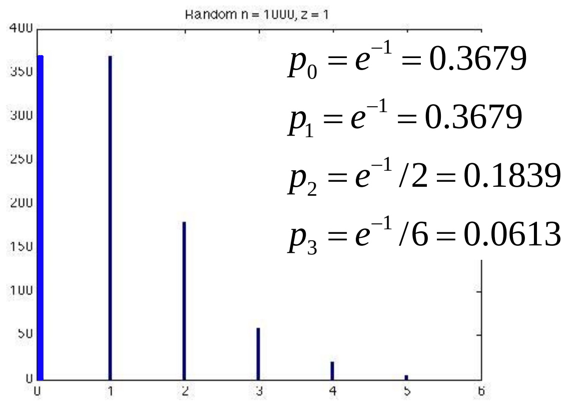
The above code finds aa, the list of nodes, assigning each to a component. (The nodes are numbered consecutively so the list of node numbers is not printed. All you get is the component number assignment.) This list says that nodes 1, 2, and 3 belong to component 1 while node 4 belongs to component 2. It then finds bb, the list containing the component numbers in the first column and the number of nodes in each component in column 2. This list says that component 1 has 3 nodes while component 2 has 1 node. cc and dd are respectively the number of components that have more than one node and the number of nodes with no neighbors. Following this is code that uses bb and aa to obtain the size of the largest component, its number, and a list of its nodes. Finally there is code that extracts the adjacency matrix comprising this component, called *maxcompgraph*, from the original adjacency matrix. Neither of these adjacency matrices is symmetric because the original graph has one directed link.

Appendix 1: Random Networks

Random networks represent the opposite in regularity from grids and lattices discussed above. Random networks can be analyzed and there is a large literature on them [Erdös and Rényi]. A random network can be built by selecting pairs of nodes and linking them with some probability. (The Matlab code for doing this is discussed in the notes called Basic Network Metrics, where routines for generating various kinds of networks are discussed.) When one builds a random network this way one gets a degree sequence that has a Poisson distribution. That is, $p(k) = e^{-z} z^k / k!$. Two example degree distributions are shown in Figure 8.



z = 10



z = 1.006

Figure 8. Degree Distributions for Two Random Networks, One with $z = 10$ and the Other with $z = 1.006$

To arrive at the basic statistics of such a network, consider first how many edges the network would have if each of the n nodes were linked: $m = n(n-1)/2$. If we connect pairs of nodes with probability p then the number of edges will be $m = pn(n-1)/2$. Then the average nodal degree of a random network will be $z = \langle k \rangle = 2m/n = pn$ (approximately, if n is large. The clustering coefficient c measures the probability that two nodes are linked, given that they have a common neighbor. But in a random network, the probability that two nodes are linked is simply p regardless of what other nodes they link to, so $c = p$. Since $p = z/n$ and we usually work with large networks with modest values of z, the clustering coefficient of random networks is usually very small.

If $z \approx 1$ or not too much bigger, then the network is tree-like and with probability approaching one there is a path between any randomly selected pair of nodes. Each node has on average z neighbors, and if the network is tree-like, then each of them has z neighbors, and so on, so that there are z^ℓ nodes at a distance ℓ from a typical node. Suppose d is the shortest distance all the way across a network of n total nodes. Then $n \sim z^d$ or $d \sim \ln(n)/\ln(z)$, providing a scaling rule for the diameter and average path length of a random network.

Appendix 2: Matlab Syntax

The following is the Matlab code for the routine *adjbuildn*, which builds a network from a nodelist.

```
function adjbuildn = adjbuildn(NL)
%builds an adjacency matrix from a directed nodelist
%column 1 is the node number, nodes it links to are in the
row
[rows,colms]=size(NL);
numnodes=rows;
maxk=colms;

B=zeros(numnodes,numnodes);

for i = 1:numnodes
    for j = 2:maxk
        if NL(i,j)~=0
            B(i, NL(i,j))=1;
        end
    end
end
adjbuildn=B;
```

The first line of code tells how to use the routine. The left side of the = is the name of the variable that the routine calculates and passes back to the user. On the right is the name of the routine followed by a list of input arguments. The routine is called *adjbuildn* and is stored as file *adjbuildn.m* To call the routine, type the following into Matlab

```
>> testadj=adjbuildn(test)
```

Test is your name for the nodelist you want converted to an adjacency matrix, while testadj is your name for the resulting matrix. If there is any discrepancy between the name of the routine in the "function" line and the filename under which the routine is stored, use the filename when you call the routine in Matlab.

If a routine has multiple input or output arguments, their names are separated by commas. But the list of output arguments is enclosed in […], as the following example shows:

```
function
[partition,componentList,mainNum,singletonNum]=componentCount(A)
```

This routine finds the isolated connected components of network A and returns the information in four separate arrays.

References

[Ahuja, Magnanti, and Orlin] *Network Flows: Theory, Algorithms, and Applications*, Englewood Cliffs: Prentice-Hall, 1993

[Erdös and Rényi] Erdös, P. and Rényi, A. On Random Graphs. *Publl. Math.* **6,** 290–297 (1959).
[Miller] L. E. Miller, "Connectivity Properties of Mesh and Ring/Mesh Networks," NIST Wireless Communications Technology Group, April 2, 2001.
[Newman] "The Structure and Function of Complex Networks," *SIAM Review* **45**, 167–256 (2003)
[Wassermann and Faust] Wasserman, S. & Faust, K., *Social Network Analysis: Methods and Applications*,
Cambridge University Press, 1994