Crash course in Handling networks with MATLAB
MATLAB is…

… a numerical computing environment and high-level programming language

… fast with matrix manipulation
Network representation

Full matrix

\[
A = \begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix}
\]
Network representation

Sparse matrix

\[
A = \begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix}
\]

\[
A_{\text{sparse}} = \begin{bmatrix}
(2,1) & 1 \\
(4,1) & 1 \\
(1,2) & 1 \\
(4,2) & 1 \\
(4,3) & 1 \\
(1,4) & 1 \\
(2,4) & 1 \\
(3,4) & 1
\end{bmatrix}
\]

\[
A = \text{full}(A_{\text{sparse}}); \quad \text{Don’t do if matrix is large!}
\]
The “tiny network incident”

\[
A_{\text{sparse}} = (1000000, 1000001) \quad 1
\]

\[
>> A = \text{full}(A_{\text{sparse}})
\]
The “tiny network incident”

A_sparse = (1000000,1000001)  1

>> A = full(A_sparse)

Make sure your node ids are integers starting at 1

(Dramatization)
Network representation

Sparse matrix

>> A = [0 1 0 1;1 0 0 1;0 0 0 1;1 1 1 0];
>> A_sparse = sparse(A)
A_sparse =
(2,1) 1
(4,1) 1
(1,2) 1
(4,2) 1
(4,3) 1
(1,4) 1
(2,4) 1
(3,4) 1

>> A = full(A_sparse); % Don’t do if matrix is large!

>> A_sparse = sparse(i,j,ones(size(i)));
>> [i,j,w] = find(A_sparse);
Network representation

Adjacency list

```matlab
>> node = struct;
>> node(1).neighbors = [2 4];
>> node(2).neighbors = [1 4];
>> node(3).neighbors = [4];
>> node(4).neighbors = [1 2 3];
```
Degree distribution

\[
A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix};
\]

\[
k = \text{sum}(A, 1)
\]

\[
k = 2 \quad 2 \quad 1 \quad 3
\]
Degree distribution

\[
\text{[x,y]} = \text{distrib(input)}
\]
\[
x = \text{unique(input)};
\]
\[
y = \text{histc(input, x)};
\]
\[
y = y./\text{sum}(y);
\]

```
>> [x,y] = distrib(k);
>> plot(x, y, 'o'); ylim([0 1])
```

```
function [x,y] = distrib(input)
    % Your code here
    % Hint: unique(), histc()
end
```

```matlab
>> A = [0 1 0 1;1 0 0 1;0 0 0 1;1 1 1 0];
>> k = sum(A, 1)
k = 2 2 1 3
```

```matlab
>> A = [0 1 0 1;1 0 0 1;0 0 0 1;1 1 1 0];
>> k = sum(A, 1)
k = 2 2 1 3
```

```
x:
 y:
 1  2  3
 0.2500 0.5000 0.2500
```
Cumulative degree distribution

\[
A = \begin{bmatrix} 0 & 1 & 0 & 1; 1 & 0 & 0 & 1; 0 & 0 & 0 & 1; 1 & 1 & 1 & 0 \end{bmatrix};
\]

\[
k = \text{sum}(A, 1)
\]

\[
k = 2 \quad 2 \quad 1 \quad 3
\]

\[
[x, y] = \text{cumdistrib}(k);
\]

\[
[x, y] = \text{cumdistrib}(k);
\]

\[
\text{plot}(x, y, 'o'); \text{ylim([0 1])}
\]

\[
\text{function} \quad [x, y] = \text{cumdistrib}(\text{input})
\]

<table>
<thead>
<tr>
<th>x:</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>y:</td>
<td>1</td>
<td>0.7500</td>
<td>0.2500</td>
</tr>
</tbody>
</table>
How this looks in a real network

>> [x,y] = distrib(k);
>> loglog(x, y, 'o'); hold on
How this looks in a real network

```matlab
>> [x,y] = distrib(k);
>> loglog(x, y, 'o'); hold on

>> loglog([x(1) x(end)], [x(1) x(end)].^(-2), '--k');
```
How this looks in a real network

```
>> [x,y] = distrib(k);
>> loglog(x, y, 'o'); hold on

>> loglog([x(1) x(end)], [x(1) x(end)].^(-2), '--k);
>> text(10^2, 10^-3, 'slope 2');
>> xlabel('Degree k'); ylabel('P(k)');
```
How this looks in a real network

```matlab
>> [x,y] = cumdistrib(k);
>> loglog(x, y, '-o'); hold on

>> loglog([x(1) x(end)], [x(1) x(end)].^(-1), '--k');
>> text(10^2, 10^-1, 'slope 1');
>> xlabel('Degree k'); ylabel('P(\geq k)');
```
Proper power law fitting should use Maximum Likelihood method!

See details & code on Aaron Clauset’s website: http://tuvalu.santafe.edu/~aaronc/powerlaws/

Often “power-laws” are not power-laws

For least squares fitting, use polyfit() and polyval()
Logbinning (first try)

>> bins = logspace(0, 4, 40); % Generate logarithmic bins
Logbinning (first try)

>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
>> pbins = pbins(1:end-1); % Remove last pbin, due to how histc works
```
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
>> pbins = pbins(1:end-1); % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins); % Divide by bin widths
```
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
>> pbins = pbins(1:end-1); % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins); % Divide by bin widths
>> pbins = pbins./sum(pbins); % Normalize
```
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
>> pbins = pbins(1:end-1); % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins); % Divide by bin widths
>> pbins = pbins./sum(pbins); % Normalize
>> bins = bins(1:end-1); % Remove last bin
```
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40);  % Generate logarithmic bins
>> pbins = histc(k, bins);     % Count degrees in each bin
>> pbins = pbins(1:end-1);     % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins);  % Divide by bin widths
>> pbins = pbins./sum(pbins);  % Normalize
>> bins = bins(1:end-1);       % Remove last bin

>> loglog(x, y, '-o'); hold on % Plot
>> loglog(bins, pbins, '-sr', 'MarkerFaceColor', [1 0 0]);
```
Logbinning (first try)

```matlab
>> bins = logspace(0, 4, 40); % Generate logarithmic bins
>> pbins = histc(k, bins); % Count degrees in each bin
>> pbins = pbins(1:end-1); % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins); % Divide by bin widths
>> pbins = pbins./sum(pbins); % Normalize
>> bins = bins(1:end-1); % Remove last bin

>> loglog(x, y, '-o'); hold on % Plot
>> loglog(bins, pbins, '-sr', 'MarkerFaceColor', [1 0 0]);
```

<table>
<thead>
<tr>
<th>bins:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pbins:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0.7963</td>
</tr>
</tbody>
</table>
Logbinning

```matlab
>> bins = [linspace(1, 9, 9) logspace(1, 4, 30)];
>> pbins = histc(k, bins);  % Count degrees in each bin
>> pbins = pbins(1:end-1);  % Remove last pbin, due to how histc works
>> pbins = pbins./diff(bins);  % Divide by bin widths
>> pbins = pbins./sum(pbins);  % Normalize
>> bins = bins(1:end-1);  % Remove last bin

>> loglog(x, y, '-o'); hold on  % Plot
>> loglog(bins, pbins, '-sr', 'MarkerFaceColor', [1 0 0]);
```

![Logbinning Graph](image)

<table>
<thead>
<tr>
<th>bins:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pbins:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0.6370</td>
</tr>
</tbody>
</table>
MatlabBGL: Boost Graph Library is...

...a MATLAB package for working with large graphs

...written in C → very fast!

https://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/
MatlabBGL: Boost Graph Library is…

…a MATLAB package for working with large graphs

…written in C → very fast!

https://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/

I have tested MatlabBGL 4.0 with every system in green.

Win32 Win64 Linux32 Linux64 MacPPC Maci386

Matlab 7.0
Matlab 7.1 SP3
Matlab R2006a
Matlab R2006b
Matlab R2007a
Matlab R2007b
Matlab R2008a
Matlab R2008b

It generally works with other specs
MatlabBGL has most algorithms you will need

Works with both full and sparse matrices, weighted and directed networks

```matlab
>> C = clustering_coefficients(A)
C = 1.0000   1.0000       0   0.3333
```
MatlabBGL has most algorithms you will need

Works with both full and sparse matrices, weighted and directed networks

```matlab
>> C = clustering_coefficients(A)
C = 1.0000    1.0000         0    0.3333

>> SP = all_shortest_paths(A)
SP =
    0     1     2     1
    1     0     2     1
    2     2     0     1
    1     1     1     0

>> asp = mean(SP(SP~=0)) % average shortest path
asp = 1.3333
```
MatlabBGL has most algorithms you will need

```matlab
>> B = betweenness_centrality(A)
B =     0     0     0     4

>> [B,B1] = betweenness_centrality(A_sparse); B1 = full(B1)
B1 =     0     1     0     2
     1     0     0     2
     0     0     0     3
     2     2     3     0
```
MatlabBGL has most algorithms you will need

```matlab
>> B = betweenness_centrality(A)
B =     0     0     0     4

>> [B,Bl] = betweenness_centrality(A_sparse); Bl = full(Bl)
Bl =     0     1     0     2
     1     0     0     2
     0     0     0     3
     2     2     3     0

>> T = mst(A) % Minimum spanning tree
T =     (2,1)     1
     (1,2)     1
     (4,2)     1
     (4,3)     1
     (2,4)     1
     (3,4)     1
```
MatlabBGL has most algorithms you will need

```matlab
>> ER = full(erdos_reyni(3,0.5))
ER =
     0     1     0
     1     0     1
     0     1     0

>> SG = full(star_graph(4))
SG =
     0     0     0     1
     0     0     0     1
     0     0     0     1
     1     1     1     0
```

```matlab
>> CG = cycle_graph(4)
CG =
     (2,1)        1
     (3,2)        1
     (4,3)        1
     (1,4)        1
```

and much more…
MatlabBGL also implements simple layouts

% circle_graph_layout
G = cycle_graph(8);
X = circle_graph_layout(G);
gplot(G,X);

% fruchterman_reingold_force_directed_layout
G = grid_graph(8,7);
X = fruchterman_reingold_force_directed_layout(G);
gplot(G,X);

% kamada_kawai_spring_layout
G = grid_graph(8,7);
X = kamada_kawai_spring_layout(G);
gplot(G,X);

% random_graph_layout
G = cycle_graph(1500);
X = random_graph_layout(G);
gplot(G,X); hold on; plot(X(:,1),X(:,2),’r.’); hold off;
% Layout on the grid
X = random_graph_layout(G,int32([0 0 5 5])); % random grid layout
gplot(G,X); grid on; hold on; plot(X(:,1),X(:,2),’r.’); hold off;
More useful MATLAB functions for working with data/networks

accumarray(): Like an SQL “GROUP BY”

triu(), tril(): Upper/Lower triangular matrix

imagesc(): Scale data and display image

eig(): Eigenvalues and eigenvectors

load(), save(), fscanf(), fprintf(), csvread(), csvwrite(), dlmread(), dlmwrite(): Read and write data from/to files

Example: Loading the lesmiserables data

```
>> el = dlmread('edge_list.csv', '\t', 1, 0); % read data (skip header)
>> el = [el; el(:, [2 1 3])]; % make undirected
>> A = sparse(el(:, 1)+1, el(:, 2)+1, el(:, 3)); % +1 to ids to start at 1
```