clear variables;

% this is a tutorial script for introduction to graph theory

help graph;

**graph** Undirected Graph

G = **graph** builds an empty graph with no nodes and no edges.

G = **graph**(A) uses the square symmetric matrix A as an adjacency matrix

and constructs a weighted graph with edges corresponding to the nonzero

entries of A. The weights of the edges are taken to be the nonzero

values in A. If A is logical then no weights are added.

G = **graph**(A,NAMES) additionally uses NAMES as the names of

the nodes in G. NAMES must be a string vector or a cell array of

character vectors, and must have as many elements as size(A,1).

G = **graph**(A,...,TYPE) uses only a triangle of A to construct the graph.

TYPE can be:

'upper' - Use the upper triangle of A.

'lower' - Use the lower triangle of A.

G = **graph**(A,...,'omitselfloops') ignores the diagonal entries of the

adjacency matrix A and does not add self-loops to the graph.

G = **graph**(S,T) constructs a graph with edges specified by the node

pairs (S,T). S and T must both be numeric, string vectors, or cell

arrays of character vectors. S and T must have the same number of elements or be

scalars.

G = **graph**(S,T,WEIGHTS) also specifies edge weights with the numeric

array WEIGHTS. WEIGHTS must have the same number of elements as S and

T, or can be a scalar.

G = **graph**(S,T,WEIGHTS,NAMES) additionally uses NAMES as the names of

the nodes in G. NAMES must be a string vector or a cell array of character

vectors. All nodes in S and T must also be present in NAMES.

G = **graph**(S,T,WEIGHTS,NUM) specifies the number of nodes of the graph

with the numeric scalar NUM. NUM must be greater than or equal to the

largest elements in S and T.

G = **graph**(S,T,...,'omitselfloops') does not add self-loops to the

graph. That is, any edge k such that S(k) == T(k) is not added.

G = **graph**(EdgeTable) uses the table EdgeTable to define the graph. The

first variable in EdgeTable must be EndNodes, and it must be a

two-column array defining the edge list of the graph. EdgeTable can

contain any number of other variables to define attributes of the graph

edges.

G = **graph**(EdgeTable,NodeTable) additionally uses the table NodeTable to

define attributes of the graph nodes. NodeTable can contain any number

of variables to define attributes of the graph nodes. The number of

nodes in the resulting graph is the number of rows in NodeTable.

G = **graph**(EdgeTable,...,'omitselfloops') does not add self-loops to the

graph.

Example:

% Construct an undirected graph from an adjacency matrix.

% View the edge list of the graph, and then plot the graph.

A = [0 10 20 30; 10 0 2 0; 20 2 0 1; 30 0 1 0]

G = graph(A)

G.Edges

plot(G)

Example:

% Construct a graph using a list of the end nodes of each edge.

% Also specify the weight of each edge and the name of each node.

% View the Edges and Nodes tables of the graph, and then plot

% G with the edge weights labeled.

s = [1 1 1 2 2 3 3 4 5 5 6 7];

t = [2 4 8 3 7 4 6 5 6 8 7 8];

weights = [10 10 1 10 1 10 1 1 12 12 12 12];

names = {'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'};

G = graph(s,t,weights,names)

G.Edges

G.Nodes

plot(G,'EdgeLabel',G.Edges.Weight)

Example:

% Construct the same graph as in the previous example using two

% tables to specify edge and node properties.

s = [1 1 1 2 2 3 3 4 5 5 6 7]';

t = [2 4 8 3 7 4 6 5 6 8 7 8]';

weights = [10 10 1 10 1 10 1 1 12 12 12 12]';

names = {'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'}';

EdgeTable = table([s t],weights,'VariableNames',{'EndNodes' 'Weight'})

NodeTable = table(names,'VariableNames',{'Name'})

G = graph(EdgeTable,NodeTable)

graph properties:

Edges - Table containing edge information.

Nodes - Table containing node information.

graph methods:

numnodes - Number of nodes in a graph.

numedges - Number of edges in a graph.

findnode - Determine node ID given a name.

findedge - Determine edge index given node IDs.

edgecount - Determine number of edges between two nodes.

addnode - Add nodes to a graph.

rmnode - Remove nodes from a graph.

addedge - Add edges to a graph.

rmedge - Remove edges from a graph.

ismultigraph - Determine whether a graph has multiple edges.

simplify - Reduce multigraph to simple graph.

degree - Degree of nodes in a graph.

neighbors - Neighbors of a node in a graph.

outedges - Edges connected to a node in a graph.

reordernodes - Reorder nodes in a graph.

subgraph - Extract an induced subgraph.

adjacency - Adjacency matrix of a graph.

incidence - Incidence matrix of a graph.

laplacian - Graph Laplacian.

shortestpath - Compute shortest path between two nodes.

shortestpathtree - Compute single source shortest paths.

distances - Compute all pairs distances.

nearest - Compute nearest neighbors of a node.

bfsearch - Breadth-first search.

dfsearch - Depth-first search.

maxflow - Compute maximum flows in a graph.

conncomp - Compute connected components of a graph.

biconncomp - Compute biconnected components of a graph.

bctree - Block-cut tree of a graph.

minspantree - Compute minimum spanning tree of a graph.

centrality - Node centrality for graph G.

isisomorphic - Determine whether two graphs are isomorphic.

isomorphism - Compute an isomorphism between G and G2.

allpaths - Compute all paths between two nodes.

allcycles - Compute all cycles in graph.

cyclebasis - Compute fundamental cycle basis of graph.

hascycles - Determine whether a graph has cycles.

plot - Plot an undirected graph.

See also digraph

Documentation for graph

# create a graph

the easiest way to create a graph in MATLAB is by using the adjacency matrix. let's create a 10 node graph

n=10; % # of nodes

A=zeros(n); % initialize the Adjacency matrix

% let's create a complete undirected graph. which means every node is

% connected to every other node. we will do this in for loops

for ii=1:n % for each node

% for each node after ii (this is because Adjacency matrix is symmetric

% and we can simply add the upper and lower diagonals later)

for jj=ii+1:n

A(ii,jj)=1; % they are connected

end

end

A

A = 10×10

0 1 1 1 1 1 1 1 1 1

0 0 1 1 1 1 1 1 1 1

0 0 0 1 1 1 1 1 1 1

0 0 0 0 1 1 1 1 1 1

0 0 0 0 0 1 1 1 1 1

0 0 0 0 0 0 1 1 1 1

0 0 0 0 0 0 0 1 1 1

0 0 0 0 0 0 0 0 1 1

0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0

% this is where we make the matrix symmetric by adding the transpose

A=A+A'

A = 10×10

0 1 1 1 1 1 1 1 1 1

1 0 1 1 1 1 1 1 1 1

1 1 0 1 1 1 1 1 1 1

1 1 1 0 1 1 1 1 1 1

1 1 1 1 0 1 1 1 1 1

1 1 1 1 1 0 1 1 1 1

1 1 1 1 1 1 0 1 1 1

1 1 1 1 1 1 1 0 1 1

1 1 1 1 1 1 1 1 0 1

1 1 1 1 1 1 1 1 1 0

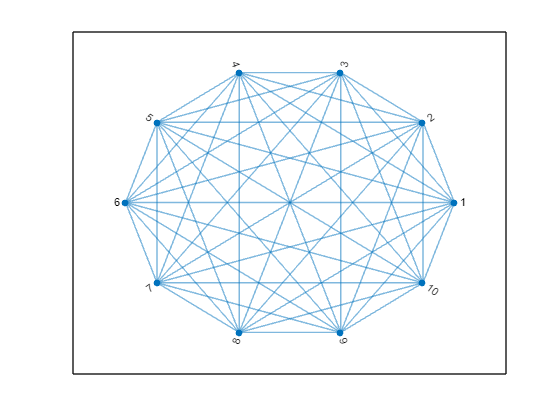
% create a graph

G=graph(A);

% plot the graph

figure(1); gcf; clf;

plot(G, 'layout', 'circle');



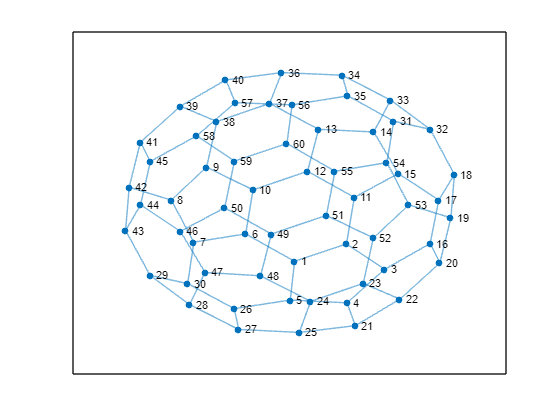
# plot another graph

Buckminster Fuller geodesic dome, which is also in the shape of a soccer ball or a carbon-60 molecule.

G=graph(bucky);

figure(2); gcf; clf;

plot(G)



# node degree

deg=degree(G)

deg = 60×1

3

3

3

3

3

3

3

3

3

3

⋮

% use this to create the degree matrix

D=diag(deg)

D = 60×60

3 0 0 0 0 0 0 0 0 0 0 0 0 ⋯

0 3 0 0 0 0 0 0 0 0 0 0 0

0 0 3 0 0 0 0 0 0 0 0 0 0

0 0 0 3 0 0 0 0 0 0 0 0 0

0 0 0 0 3 0 0 0 0 0 0 0 0

0 0 0 0 0 3 0 0 0 0 0 0 0

0 0 0 0 0 0 3 0 0 0 0 0 0

0 0 0 0 0 0 0 3 0 0 0 0 0

0 0 0 0 0 0 0 0 3 0 0 0 0

0 0 0 0 0 0 0 0 0 3 0 0 0

⋮

% get the adjacency matrix

% this gives the sparse form

A=adjacency(G)

A =

(2,1) 1

(5,1) 1

(6,1) 1

(1,2) 1

(3,2) 1

(11,2) 1

(2,3) 1

(4,3) 1

(16,3) 1

(3,4) 1

(5,4) 1

(21,4) 1

(1,5) 1

(4,5) 1

(26,5) 1

(1,6) 1

(7,6) 1

(10,6) 1

(6,7) 1

(8,7) 1

(30,7) 1

(7,8) 1

(9,8) 1

(42,8) 1

(8,9) 1

(10,9) 1

(38,9) 1

(6,10) 1

(9,10) 1

(12,10) 1

(2,11) 1

(12,11) 1

(15,11) 1

(10,12) 1

(11,12) 1

(13,12) 1

(12,13) 1

(14,13) 1

(37,13) 1

(13,14) 1

(15,14) 1

(33,14) 1

(11,15) 1

(14,15) 1

(17,15) 1

(3,16) 1

(17,16) 1

(20,16) 1

(15,17) 1

(16,17) 1

(18,17) 1

(17,18) 1

(19,18) 1

(32,18) 1

(18,19) 1

(20,19) 1

(53,19) 1

(16,20) 1

(19,20) 1

(22,20) 1

(4,21) 1

(22,21) 1

(25,21) 1

(20,22) 1

(21,22) 1

(23,22) 1

(22,23) 1

(24,23) 1

(52,23) 1

(23,24) 1

(25,24) 1

(48,24) 1

(21,25) 1

(24,25) 1

(27,25) 1

(5,26) 1

(27,26) 1

(30,26) 1

(25,27) 1

(26,27) 1

(28,27) 1

(27,28) 1

(29,28) 1

(47,28) 1

(28,29) 1

(30,29) 1

(43,29) 1

(7,30) 1

(26,30) 1

(29,30) 1

(32,31) 1

(35,31) 1

(54,31) 1

(18,32) 1

(31,32) 1

(33,32) 1

(14,33) 1

(32,33) 1

(34,33) 1

(33,34) 1

(35,34) 1

(36,34) 1

(31,35) 1

(34,35) 1

(56,35) 1

(34,36) 1

(37,36) 1

(40,36) 1

(13,37) 1

(36,37) 1

(38,37) 1

(9,38) 1

(37,38) 1

(39,38) 1

(38,39) 1

(40,39) 1

(41,39) 1

(36,40) 1

(39,40) 1

(57,40) 1

(39,41) 1

(42,41) 1

(45,41) 1

(8,42) 1

(41,42) 1

(43,42) 1

(29,43) 1

(42,43) 1

(44,43) 1

(43,44) 1

(45,44) 1

(46,44) 1

(41,45) 1

(44,45) 1

(58,45) 1

(44,46) 1

(47,46) 1

(50,46) 1

(28,47) 1

(46,47) 1

(48,47) 1

(24,48) 1

(47,48) 1

(49,48) 1

(48,49) 1

(50,49) 1

(51,49) 1

(46,50) 1

(49,50) 1

(59,50) 1

(49,51) 1

(52,51) 1

(55,51) 1

(23,52) 1

(51,52) 1

(53,52) 1

(19,53) 1

(52,53) 1

(54,53) 1

(31,54) 1

(53,54) 1

(55,54) 1

(51,55) 1

(54,55) 1

(60,55) 1

(35,56) 1

(57,56) 1

(60,56) 1

(40,57) 1

(56,57) 1

(58,57) 1

(45,58) 1

(57,58) 1

(59,58) 1

(50,59) 1

(58,59) 1

(60,59) 1

(55,60) 1

(56,60) 1

(59,60) 1

% to get the full form, type

% A=full(A);

L=laplacian(G);

# let's work with another type of graph

this creates a square lattice with 8 nodes on each side

n = 10;

A = delsq(numgrid('L',n+2));

G = graph(A,'OmitSelfLoops');

G.Edges.Weight = ones(numedges(G),1);

figure(1); gcf; clf;

h = plot(G);

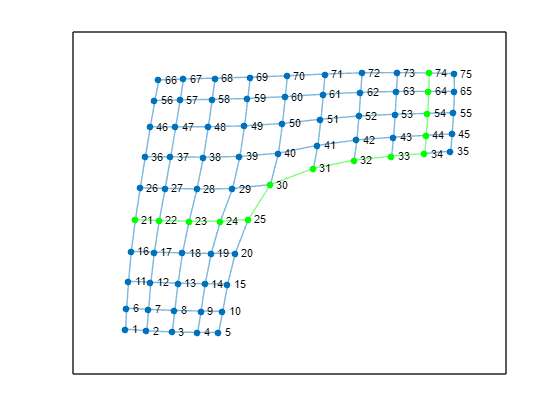
% get the node degree

deg=degree(G);

% highlight the shortest path between 74 and 21

path = shortestpath(G,74,21);

highlight(h,path,'NodeColor','g','EdgeColor','g')



# a type of random graph

n=10;

p=.5; % probability of an edge between any two nodes

% let's create a complete undirected graph. which means every node is

% connected to every other node.

% generate a random number and if it is less than p then make an

% edge

A=rand(n)<p;

A=triu(A)+triu(A)';

A=A-diag(diag(A));

% can alternatively do this in a for loop

% A=zeros(n);

% for ii=1:n

% for jj=ii+1:n

% if rand < p

% A(ii,jj)=1; % they are connected

% end

% end

% end

% this is where we make the matrix symmetric by adding the transpose

% A=A+A';

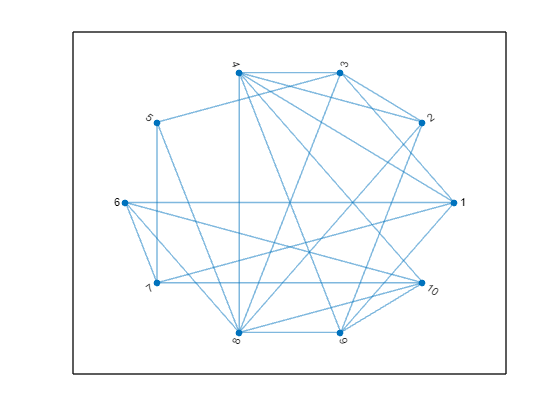
% create a graph

G=graph(A);

% plot the graph

figure(1); gcf; clf;

plot(G, 'layout', 'circle');



# network properties (reachability)

% help distances

geo=distances(G);

% to avoid counting self-distances

for ii=1:n

geo(ii,ii)=nan;

end

% characteristic path length

L=1/(n\*(n-1))\*nansum(nansum(geo))

L = 1.4889

% harmonic mean

E=1/(n\*(n-1))\*nansum(nansum(1./geo))

E = 0.7630

# network properties (connectivity)

deg=degree(G);

mean(deg)

ans = 4.8000

% degree distributions

figure(2); gcf; clf;

subplot(1,2,1);

hist(deg, 1:n-1);

xlabel('node connnectivity');

ylabel('# of nodes');

% show in the form of P(k)

[N, k]=hist(deg, 1:n-1);

% this makes sure that the sum(P(k))=1

Pk=N/sum(N);

subplot(1,2,2);

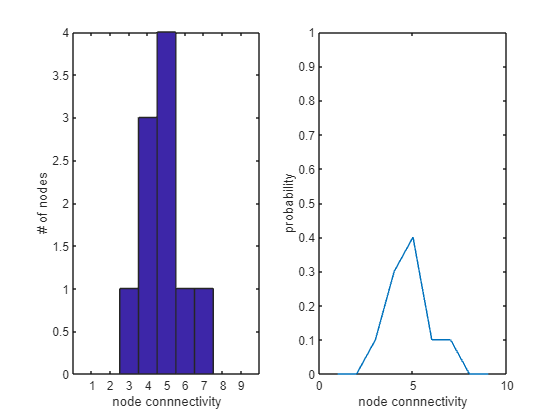
plot(k, Pk);

% set the limits from 0 to 1 so that we compare properly

set(gca, 'ylim', [0 1]);

xlabel('node connnectivity');

ylabel('probability');



% laplacian

L=laplacian(G);

lambda=eig(L);

lambda(1)

ans = -1.4218e-15

lambda(2)

ans = 2.3836

% number of connected components = C

% rank of L = N-C

rank(full(L))

ans = 9

conn=n-rank(full(L))

conn = 1

% first and second moments

% first moment, same as mean

sum(k.\*Pk)

ans = 4.8000

% second moment, same as variance

% note the high value showing that this graph has very high and low

% connected nodes

sum(k.^2.\*Pk)

ans = 24.2000

% check

max(deg)

ans = 7

min(deg)

ans = 3

# more network properties

% correlation

% the second argument is the dimension of the matrix, which is rows in this

% case

N=size(A,1);

% the node degree

% sum along columns

k=sum(A,2);

figure(2); gcf; clf;

knn=zeros(numel(k),1);

for i=1:N

% vectorize the sum aij\*kj

% j are all the columns

knn(i)=sum(A(i,:).\*k')/k(i);

end

subplot(1,2,1); gca;

% plot knn against k

plot(k, knn, '.');

set(gca, 'fontsize', 20);

xlabel('k');

ylabel('knn');

subplot(1,2,2); gca;

% this creates a loglog plot

plot(log(k), log(knn), '.');

hold on;

krv=unique(k)';

for jj=krv

mean\_knn(jj==krv)=mean(knn(k==jj));

end

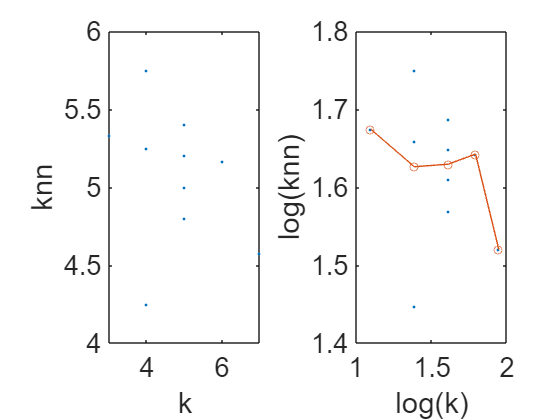
plot(log(krv), log(mean\_knn), '-o');

% errorbar(krv, mknn, sknn, 'k', 'linewidth', 2);

set(gca, 'fontsize', 20);

xlabel('log(k)');

ylabel('log(knn)');



I have run all commands/scripts in the graph\_theory\_intro file assigned with this homework.