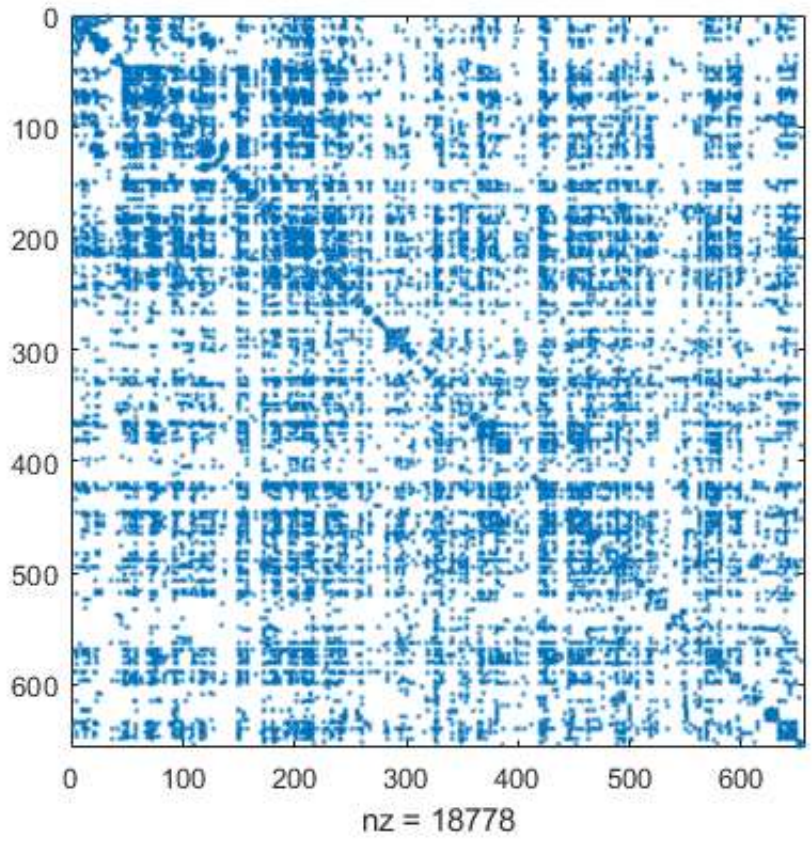
Contents


```
close all
clear all
clc

% IMPORT & Polishing Data%
G = csvread('Adj.csv',1,1);

party_name = readtable("party_name.csv");
party_name = party_name(:,2);
```

```
ideas_name = readtable("Ideas_name.csv");
ideas_name = ideas_name(:,2);

full_name = vertcat(party_name,ideas_name);

N = max(size(G));
A = sparse(G);
clear G;
```

pre-processing

```
Au = 1*(A+A'>0);                    % undirected network
Au = Au - diag(diag(Au));           % clear diagonal (you never know)

% Remove nodes which are NOT connected
pos = find(sum(Au)~=0);
A = A(pos,pos);
Au = Au(pos,pos);
full_name = full_name(pos,1);
spy(Au);
```



%%%%%%%%%%%%%%%%%% EXTRACT THE DISTRIBUTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
% Distribution
N = size(A,1);              % Number of Nodes
 display("Number of Nodes= "+N)
d = full(sum(A));           % Degree Vector
%d = d(d>0);                % avoid zero degrees
```

```
Links_num = sum(d);         % Total number of links --G (Let's decide if #node is this or
we want to divide by 2)
 display("Total number of links= "+Links_num)

k = unique(d);              % Degree Samples
pk = histc(d,k)';           % counts occurrences
pk = pk/sum(pk);            % normalize to 1

% Cumulative distribution
Pk = cumsum(pk,'reverse');

% Log Binning
klog = 10.^(0:0.1:ceil(log10(max(k))));
pklog = histc(d,klog)';     % counts occurrences
pklog = pklog/sum(pklog);   % normalize to 1
```

```
"Number of Nodes= 656"

"Total number of links= 18778"
```

%%%%%%%%%%%%%%%%%%% MOMENTS OF DEGREE DISTRIBUTION %%%%%%%%%%%%%%%%%%%%%%%

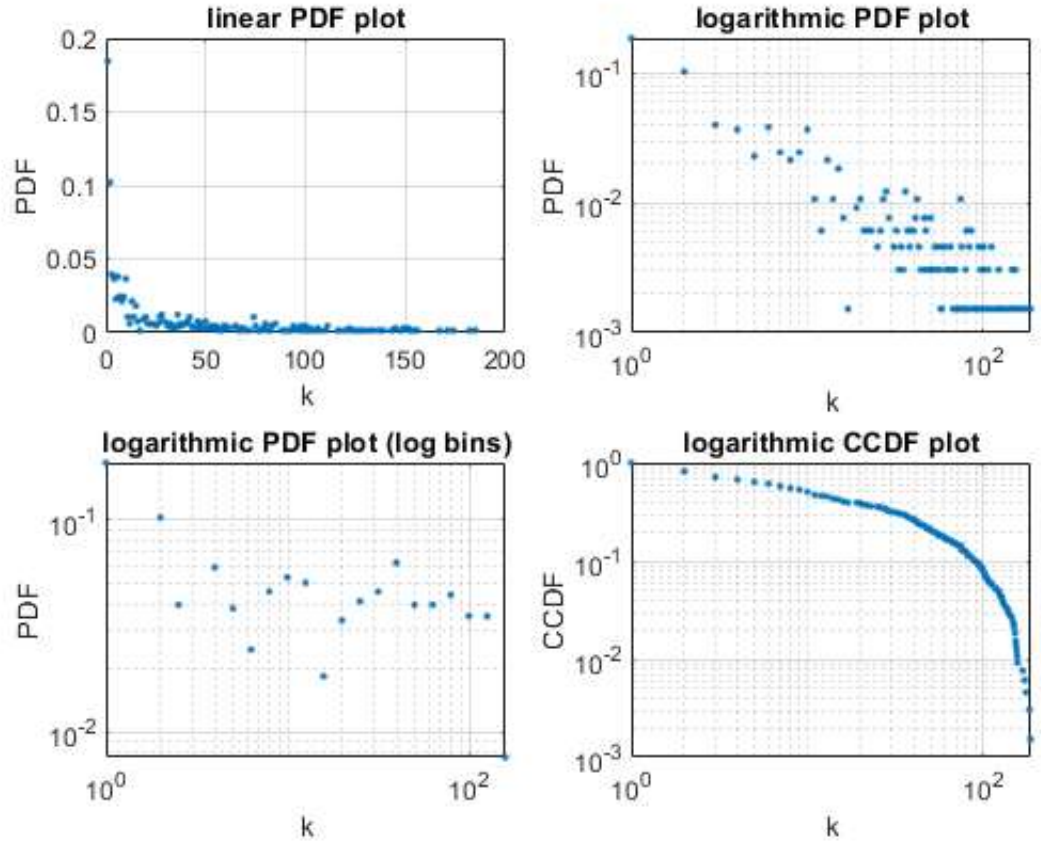
```
Mean_D = mean(d);           % First Moment of prob. distribution
display("First Moment of prob. distribution= "+Mean_D)

Var_D = var(d);             % Second Moment of Prob. distribution (Express the spread)
display("Second Moment of Prob. distribution= "+Var_D)

Skew_D = skewness(d);       % Third Moment of Prob. distribution (How symmetric around ave
rage)
display("Third Moment of Prob. distribution= "+Skew_D)
```

```
"First Moment of prob. distribution= 28.625"

"Second Moment of Prob. distribution= 1500.079"

"Third Moment of Prob. distribution= 1.7576"
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%% SHOW THE RESULTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
figure(2)
subplot(2,2,1)
plot(k,pk,'.')
grid
xlabel('k')
ylabel('PDF')
title('linear PDF plot')
subplot(2,2,2)
loglog(k,pk,'.')
grid
xlabel('k')
```

```
ylabel('PDF')
title('logarithmic PDF plot')
subplot(2,2,3)
loglog(klog,pklog,'.')
grid
xlabel('k')
ylabel('PDF')
title('logarithmic PDF plot (log bins)')
subplot(2,2,4)
loglog(k,Pk,'.')
grid
xlabel('k')
ylabel('CCDF')
title('logarithmic CCDF plot')
```



%%%%%%%%%%%%%%%%%%% PURE ML FITTING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
kmin = 40;
d2 = d(d>=kmin);                    % restrict range
ga = 1+1/mean(log(d2/kmin));        % estimate the exponent
disp(['gamma ML = ' num2str(ga)])
```

```
gamma ML = 2.5565
```

%%%%%%%%%%%%%%%%%%% ML FITTING WITH SATURATION %%%%%%%%%%%%%%%%%%%%

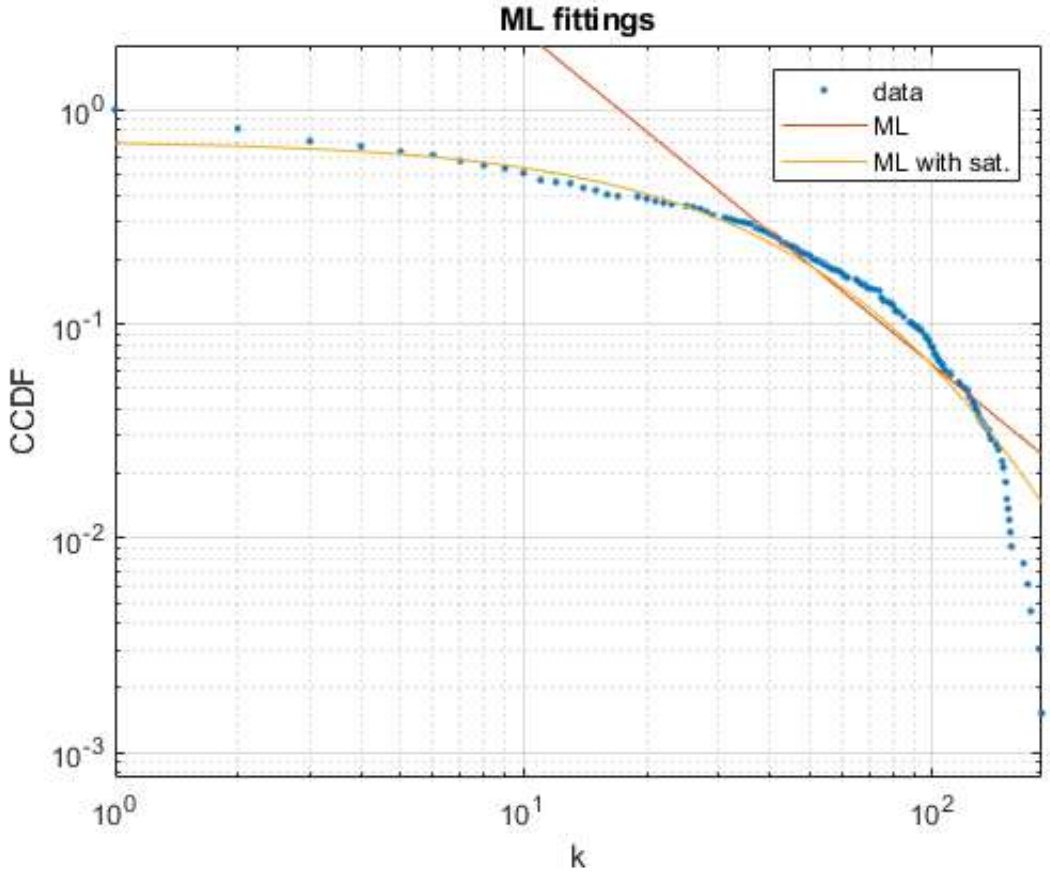
```
d1=d(d>30);
```

```
for ks = 1:max(k)
    kmin = min(d1);
    tmp = mean(log((d1+ks)/(kmin+ks)));
    ga2(ks) = 1+1/tmp;
    de(ks) = log(ga2(ks)-1)-log(kmin+ks)-ga2(ks)*tmp;
end
[~,ks] = max(de);
disp(['k_sat ML sat = ' num2str(ks)])
disp(['gamma ML sat = ' num2str(ga2(ks))])
```

```
k_sat ML sat = 185
gamma ML sat = 6.6197
```

%%%%%%%%%%%%%%%%%% SHOW THE RESULTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
figure(5)
% data
loglog(k,Pk,'.')
hold on
% ML fitting (we make sure that the plot follows the data)
s1 = k.^(1-ga); % build the CCDF signal
loglog(k,s1/s1(40)*Pk(40));
% ML fitting with saturation
s1 = ((k+ks)/(kmin+ks)).^(1-ga2(ks));
loglog(k,s1*exp(-1.2))
hold off
axis([xlim min(Pk/2) 2])
grid
xlabel('k')
ylabel('CCDF')
title('ML fittings')
legend('data','ML','ML with sat.')
```



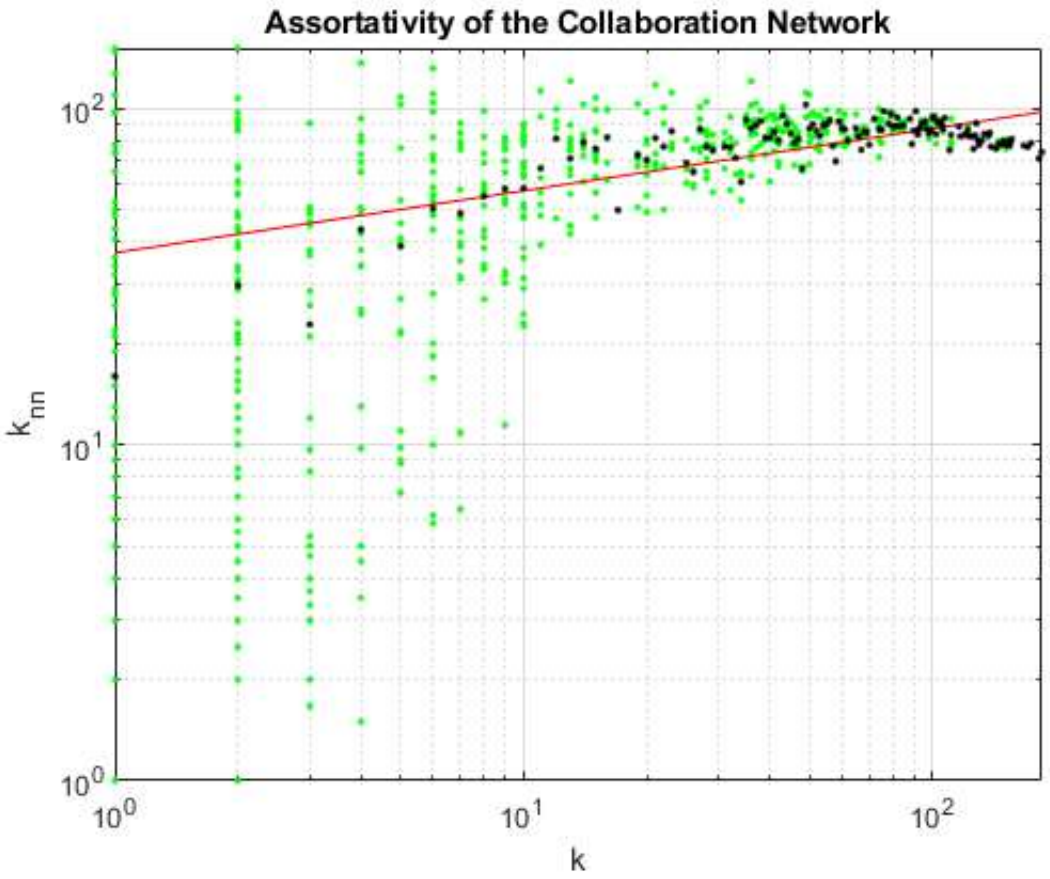
%%%%%%%%%%%%%%%%%%%%%%%%%%%% ASSORTATIVITY %%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
k_tmp = (A*d')./d'; % the average degree of neighbours

% extract averages for each value of k
u = unique(d');
for k = 1:length(u)
    k_nn(k) = mean(k_tmp(d'==u(k)));
end

% do the linear fitting
p = polyfit(log(u'),log(k_nn),1);
disp(['Assortativity factor =' num2str(p(1))])          %Assortativity factor
```

```
Assortativity factor =0.18504
```

%%%%%%%%%%%%%%%%%% SHOW RESULTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
figure(6)
loglog(d,k_tmp,'g.');
hold on
loglog(u,exp(p(2)+log(u)*p(1)),'r-');
loglog(u,k_nn,'k.');
hold off
grid
xlabel('k')
ylabel('k_{nn}')
title('Assortativity of the Collaboration Network')
```



%%%%%%%%%%%%%%%% CLUSTRING COEFFICIENT 1 %%%%%%%%%%%%%%%%%%%%%%%%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clustring Coefficient: Measure density of links in the neighbourhood

cn = diag(full((Au*triu(Au)*Au)));          % Number of triangles for each node
Ei = zeros(size(d));
Ei = cn(d>1).';                             % Number of edges between nodes of neighbourho
od
%E_Max = 0.5*d(d>1).*(d(d>1)-1);             % Maximum Number of Edges (Pairs)
C1 = 2*Ei./d(d>1)./(d(d>1)-1);                                  % Clustring Coefficient
%C1 = C1(C1>0);
Cave1 = sum(C1)/N;
```

%%%%%%%%%%%%%%%% CLUSTRING COEFFICIENT 2 %%%%%%%%%%%%%%%%%%%%%%%%%

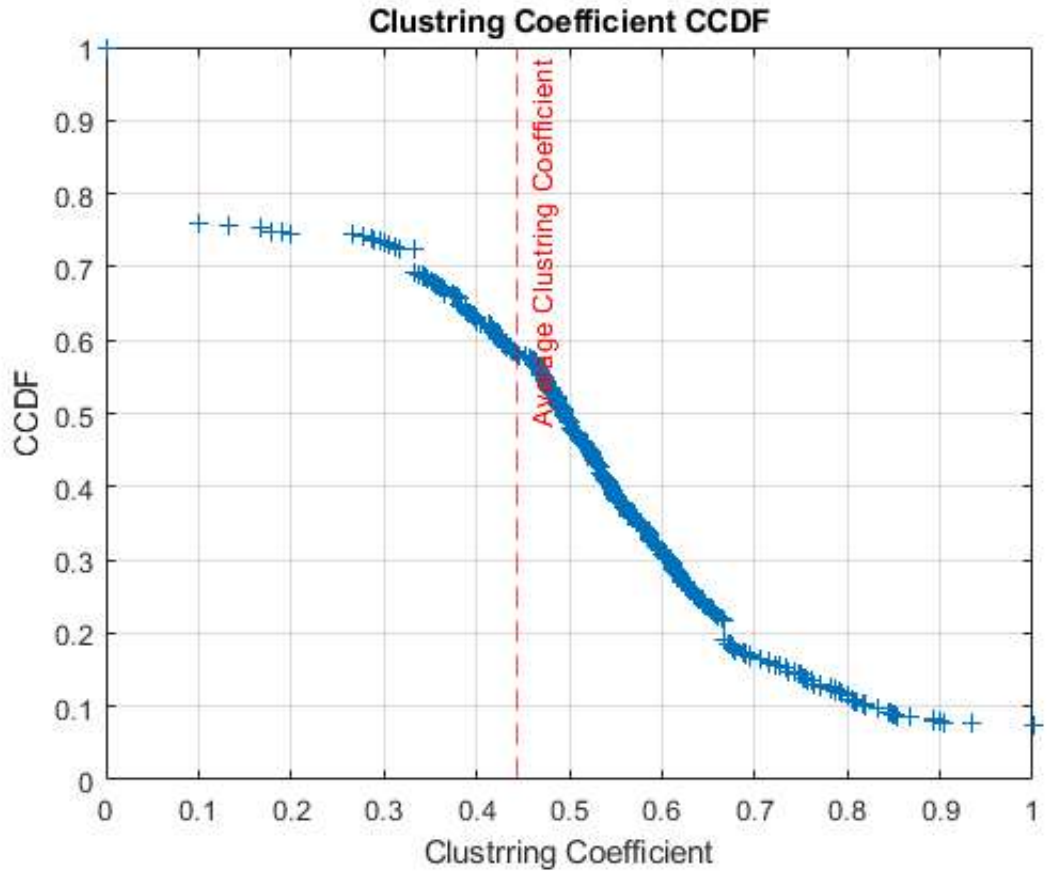
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:length(Au)
  str(i).child = find(Au(i,:)>0);
end
N = length(Au);
deg = sum(Au(:,:));

% find neighboring links

for i=1:N
    adjn = Au(str(i).child,str(i).child);
```

```
    E(i) = sum(sum(adjn))/2;
    if deg(i)==1 | deg(i)==0
        C(i) = 0;
    else
        C(i) = 2*E(i)/deg(i)/(deg(i)-1);
    end

end
%C = C(C>0);
C = full(C);
Cave2 = sum(C)/N;
 disp(['Average Clustering Coefficient = ' num2str(Cave2)])
```

```
Average Clustering Coefficient = 0.44409
```

CLUSTRING COEFFICIENT PROB. DISTRIBUTION

```
s = unique(C);                                  % Unique Occurrences
%k = unique(d);
Cpk = histc(C,s)';                              % counts occurrences
Cpk = Cpk/sum(Cpk);                             % normalize to 1

% Cumulative distribution
CPk = cumsum(Cpk,'reverse');
figure(7);
plot(s,CPk,'+');
xline(Cave2,'r--',{'Average Clustring Coefficient'});
grid
xlabel('Clustrring Coefficient')
ylabel('CCDF')
title('Clustring Coefficient CCDF')
```



%%%%%%%%%%%%%%%%%%%%%%%%%% ROBUSTNESS %%%%%%%%%%%%%%%%%%%%%%%%%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Robustness: if you knock out x% of nodes/edges, how many % survive ?
```

Inhomogeneity Ratio:

```
inhom_Ratio = mean(d.^2)./mean(d);
disp(['The Inhomogeneity Ratio = ' num2str(inhom_Ratio)])
```

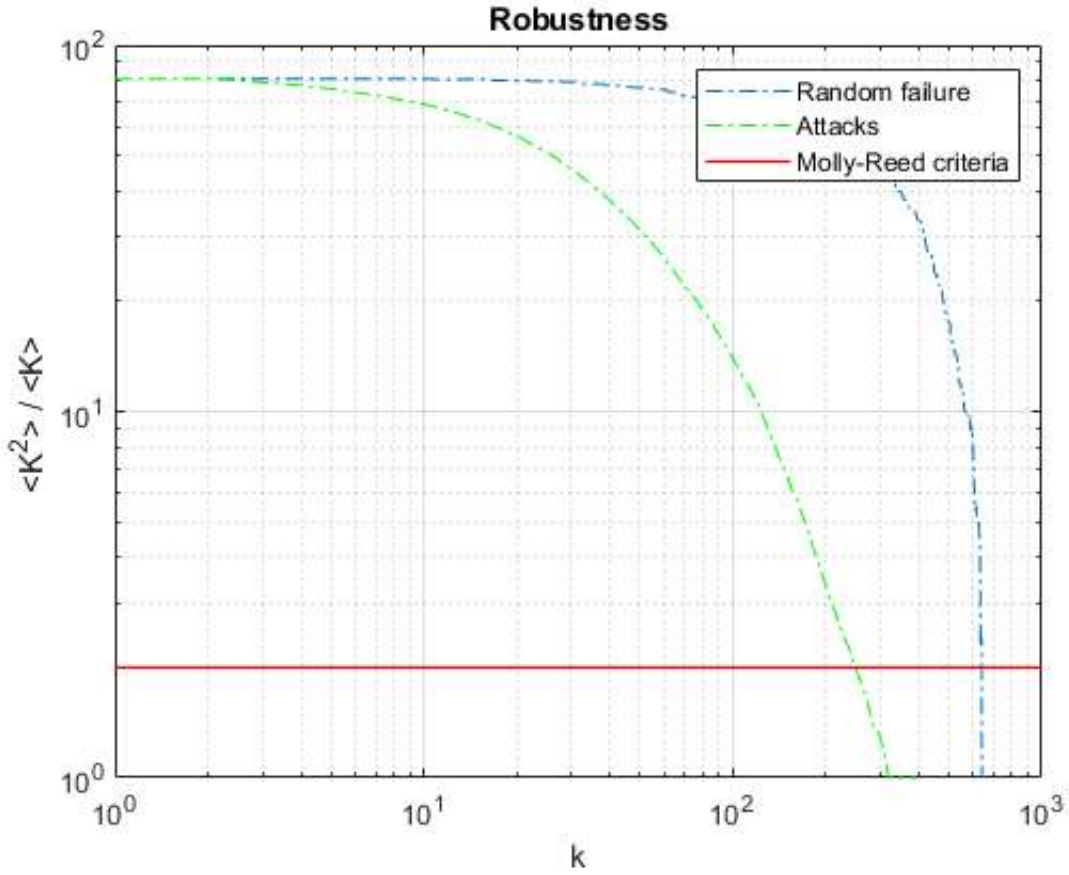
Robustness for Random Attack failure

```
Au_update = Au;
Rand_inhom_Ratio = mean(d.^2) ./ mean(d);

for i = 1:N-1
    j = ceil((N-i)*rand)+1;
    Au_update(:,j) = [];
    Au_update(j,:) = [];
    d_update = full(sum(Au_update));

    mom2_k = mean(d_update.^2);
    mom1_k = mean(d_update);

    Rand_inhom_Ratio_update = mom2_k ./mom1_k;
    Rand_inhom_Ratio = [Rand_inhom_Ratio Rand_inhom_Ratio_update];
end
```

Robustness for Attacks (Adversary which removes all hubs first)

```
Au_update = Au;
Attack_inhom_Ratio = mean(d.^2) ./ mean(d);

for i = 1:N-1
    [hub_degree,hub_index] = max(d_update);
    j = hub_index;
    Au_update(:,j) = [];
    Au_update(j,:) = [];
    d_update = full(sum(Au_update));

    mom2_k = mean(d_update.^2);
    mom1_k = mean(d_update);

    Attack_inhom_Ratio_update = mom2_k ./mom1_k;
    Attack_inhom_Ratio = [Attack_inhom_Ratio Attack_inhom_Ratio_update];
end

% Showing the results (with Molly-Reed criteria)

figure(8);
loglog(Rand_inhom_Ratio,'-.');
hold on
loglog(Attack_inhom_Ratio,'g-.');
hline = refline([0 2]);
hline.Color = 'r';
hline.LineWidth = 1;
hold off
grid;
legend('Random failure','Attacks','Molly-Reed criteria');
xlabel('k');
ylabel('<K^2> / <K> ');
title('Robustness');
```

```
The Inhomogeneity Ratio = 80.9496
```



%%%%%%%%%%%%%%%%%%% Community Detection %%%%%%%%%%%%%%%%%%%%%%%%%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G = csvread('gauss_Adj.csv',1,1);
N = max(size(G));
B = sparse(G);
clear G;

Bu = 1*(B+B'>0);                    % undirected network
Bu = Bu - diag(diag(Bu));           % clear diagonal (you never know)

% remove nodes which are NOT connected
pos = find(sum(Bu)~=0);
B = B(pos,pos);
Bu = Bu(pos,pos);
N = size(Bu,1);

% Laplacian Matrix generation
d = full(sum(Bu));                   % degree vector
D = sum(d);                          % degrees sum
I = spdiags(ones(N,1),0,N,N);        % identity matrix
Di = spdiags(1./sqrt(d'),0,N,N);     % diagonal degrees square-rooted
L = I - Di*Bu*Di;                    % normalized Laplacian
M = Bu*Di*Di;                        % normalized adjacancy matrix
```

%%%%%%%%%%%%%%%%%%%%%%%%% spectral approach %%%%%%%%%%%%%%%%%%%%%%%%%

```
% extract eigenvectors
```

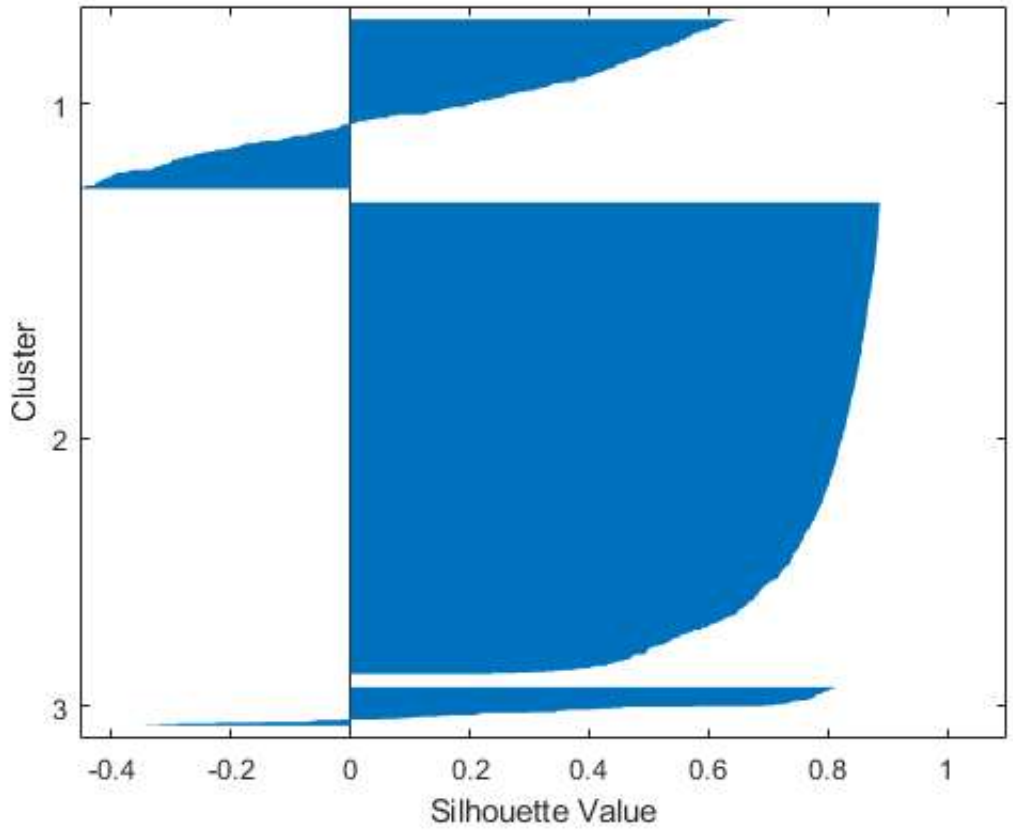
```
[V,DD] = eigs(L,6,'SA');
Vv = Di*V;                          % normalize eigenvectors
v1 = Vv(:,2)/norm(Vv(:,2));         % Fiedler's vector
% sweep wrt the ordering identified by v1
% reorder the adjacency matrix
[v1s,pos] = sort(v1,'descend');
Bu1 = Bu(pos,pos);
% evaluate the conductance measure
a = sum(triu(Bu1));
b = sum(tril(Bu1));
assoc = cumsum(a+b);
assoc = min(assoc,D-assoc);
cut = cumsum(b-a);
conduct = cut./assoc;
conduct = conduct(1:end-1);
% identify the minimum -> threshold
[~,mpos] = min(conduct);
threshold = mean(v1s(mpos:mpos+1));
disp([' '])
disp('spectral approach')
disp(['   Minimum conductance: ' num2str(conduct(mpos))])
disp(['   Cheeger''s upper bound: ' num2str(sqrt(2*DD(2,2)))])
disp(['   # of links: ' num2str(D/2)])
disp(['   Cut value: ' num2str(cut(mpos))])
disp(['   Assoc value: ' num2str(assoc(mpos))])
disp(['   Community size #1: ' num2str(mpos)])
disp(['   Community size #2: ' num2str(N-mpos)])
disp([' '])
```

```
spectral approach
   Minimum conductance: 0.50045
   Cheeger's upper bound: 1.4148
   # of links: 619941
   Cut value: 310249
   Assoc value: 619941
   Community size #1: 557
   Community size #2: 557
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Kmean Approach %%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
LRC = kmeans(V(:,1:3),3);
disp([' '])
disp('Kmean approach')
disp("  Community size #1: "+sum(LRC == 1))
disp("  Community size #2: "+sum(LRC == 2))
disp("  Community size #3: "+sum(LRC == 3))
silhouette(V(:,1:3),LRC)
```

```
Kmean approach
  Community size #1: 278
  Community size #2: 773
  Community size #3: 63
```



%%%%%%%%%%%%%% Sovrapposition clustering and manual communities

```
comt = readtable("community_truth.csv");
comt = table2array(comt(:,2));
err=0;

%suppose we assign 1 to left, 2 to center and 3 to right

for i = 1:length(LRC)
    if comt(i)~=LRC(i)
        err=err+1;
    end
end

disp("Sovrapposition cluster and manually computed communities=" + (1-err/length(LRC)))
```

```
Sovrapposition cluster and manually computed communities=0.1912
```

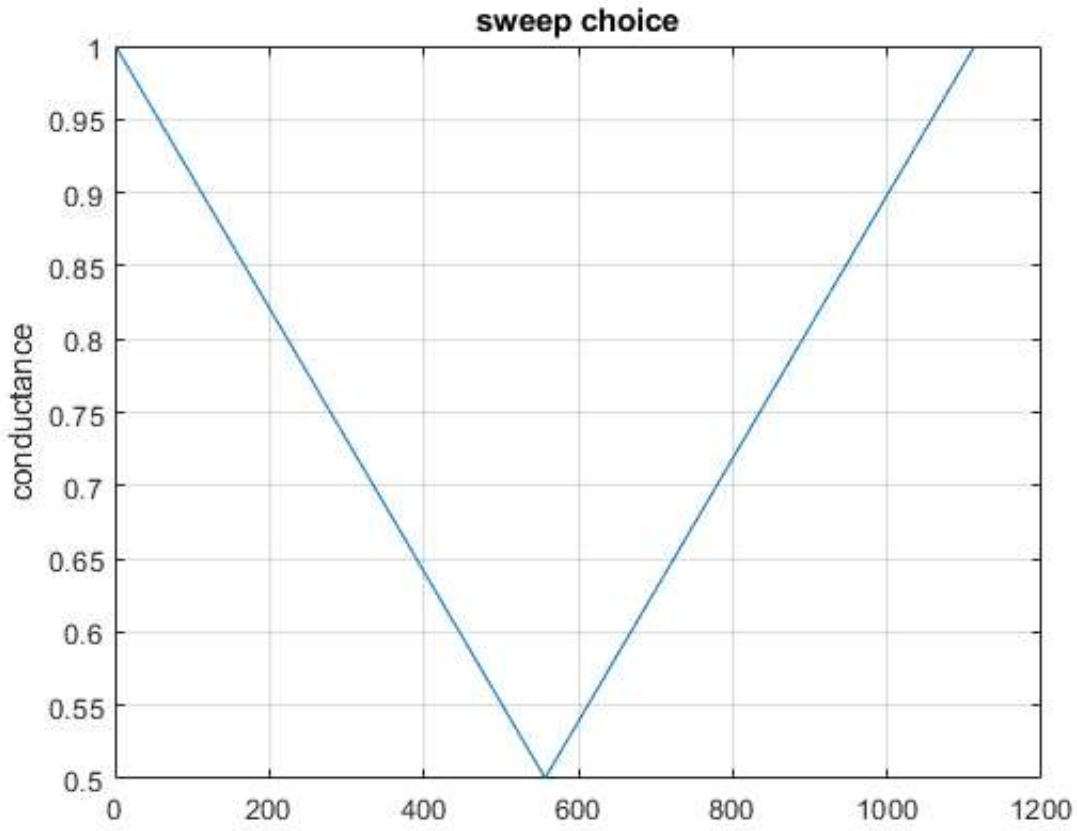
%%%%%%%%%%%%%%%%%%%%%%%%%%%% PageRank-nibble approach %%%%%%%%%%%%%%%%%%%%

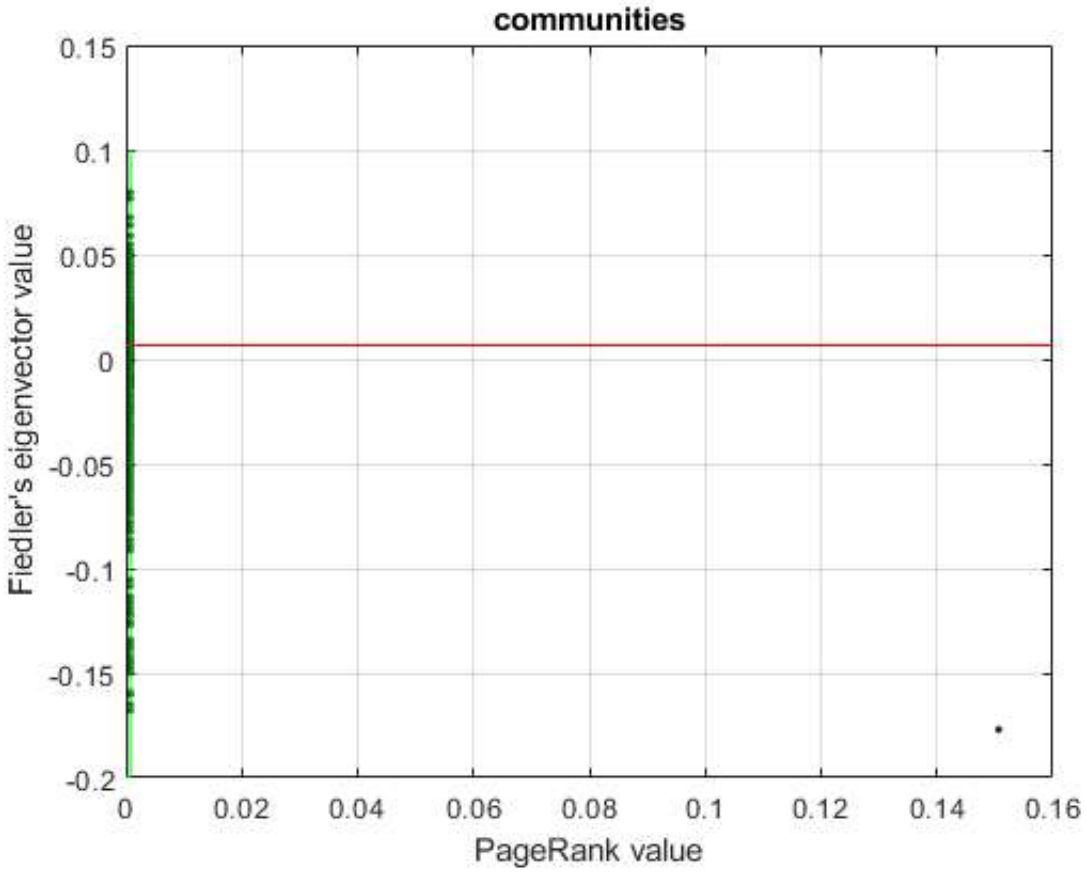
```
if mpos<N-mpos              % select seed node from the smaller group
    i = pos(1);             % we select the more relevant from the perspective of the spec
tral approach
else
    i = pos(end);
end
q = zeros(N,1);
q(i) = 1;                   % teleport vector
c = 0.85;
```

```
r = (I-c*M)\((1-c)*q);      % ranking vector
ep = 1e-3;                  % precision

% run PageRank-nibble
u = zeros(N,1);             % starting point
v = q;                      % starting point
th = full(ep*d/D)';         % thresholds
count = 0;                  % exit counter
complexity = 0;             % complexity value (# of operations)
ii = i;                     % starting index used for Push operation
while (count<N)
    if v(ii)>th(ii)         % push if above threshold
        tmp = v(ii);
        u(ii) = u(ii)+(1-c)*tmp;
        v(ii) = 0;
        v = v + c*M(:,ii)*tmp;
        complexity = complexity + d(ii); % update complexity
        count = 0;          % reset the exit counter
    else                    % go to next entry if below threshold
        count = count + 1;  % increase the exit counter
        ii = mod(ii,N)+1;   % update the index used for Push
    end
end

% sweep wrt the ordering identified by v1
% reorder the adjacency matrix
[u1s,pos2] = sort(u,'descend');
Nmax = find(u1s>0,1,'last'); % discard nodes with 0 values (never used in Push)
Bu1 = Bu(pos2,pos2(1:Nmax));
% evaluate the conductance measure
a = sum(triu(Bu1));
b = sum(tril(Bu1));
assoc = cumsum(a+b);
assoc = min(assoc,D-assoc);
cut = cumsum(b-a);
conduct = cut./assoc;
conduct = conduct(1:Nmax-1);
% identify the minimum -> threshold
[~,mpos2] = min(conduct);
threshold2 = mean(u1s(mpos2:mpos2+1));
disp('PageRank-nibble approach')
disp(['   complexity/D: ' num2str((complexity/D))])
disp(['   epsilon: ' num2str(ep)])
disp(['   prec: ' num2str(norm(r-u,1))])
disp(['   Minimum conductance: ' num2str(conduct(mpos2))])
disp(['   # of links: ' num2str(D/2)])
disp(['   Cut value: ' num2str(cut(mpos2))])
disp(['   Assoc value: ' num2str(assoc(mpos2))])
disp(['   Community size #1: ' num2str(mpos2)])
disp(['   Community size #2: ' num2str(N-mpos2)])

% show sweep choice
figure(9)
plot(conduct)
grid
ylabel('conductance')
title('sweep choice')

% show network with partition
figure(10)
plot(u,v1,'k.')
```

```
hold on
% plot(u(pos2(1:mpos2)),v1(pos2(1:mpos2)),'go')
plot(threshold2*[1,1],ylim,'g-')
plot(xlim,threshold*[1,1],'r-')
hold off
grid
ylabel('Fiedler''s eigenvector value')
xlabel('PageRank value')
title('communities')
```

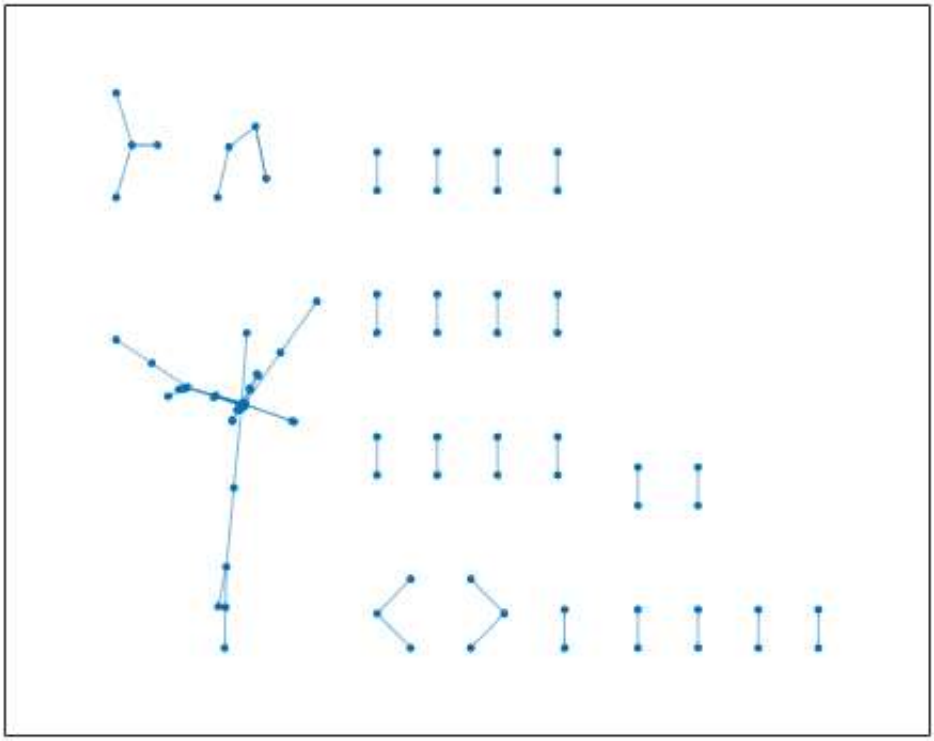
```
PageRank-nibble approach
   complexity/D: 23.8384
   epsilon: 0.001
   prec: 0.00047369
   Minimum conductance: 0.50045
   # of links: 619941
   Cut value: 310249
   Assoc value: 619941
   Community size #1: 557
   Community size #2: 557
```





Community identified by connected components in our disconnected graph

```
G = graph(Au);
[bins,binsizes] = conncomp(G);
mask=(bins==1);
figure(11);
plot(G)
part_A=Au(mask,mask);
```



Clustering on biggest connected component%%

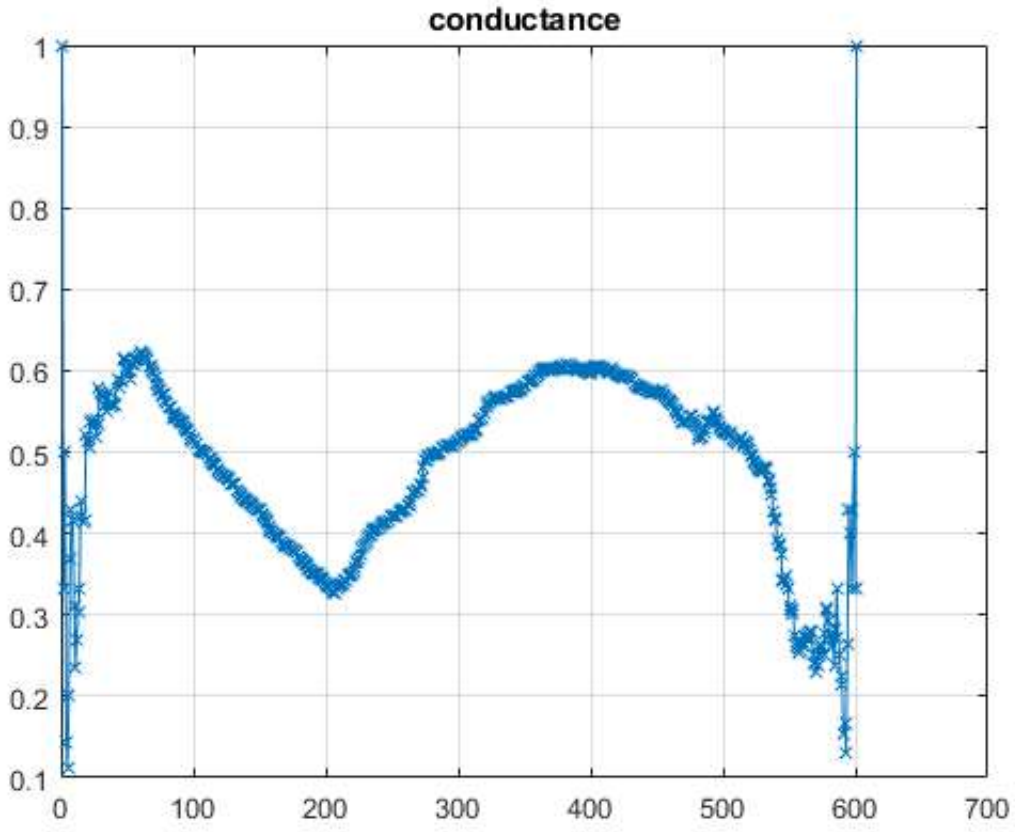
```
N = size(part_A,1);
d = full(sum(part_A)); % degree vector
D = sum(d); % degrees sum
I = spdiags(ones(N,1),0,N,N); % identity matrix
Di = spdiags(1./sqrt(d'),0,N,N); % diagonal degrees square-rooted
L = I - Di*part_A*Di; % normalized Laplacian
M = part_A*Di*Di; % normalized adjacancy matrix
```

spectral approach on part_A

```
% extract eigenvectors
[V,DD] = eigs(L,6,'SA');
Vv = Di*V; % normalize eigenvectors
v1 = Vv(:,2)/norm(Vv(:,2)); % Fiedler's vector
% sweep wrt the ordering identified by v1
% reorder the adjacency matrix
[v1s,pos] = sort(v1,'descend');
Au1 = part_A(pos,pos);
% evaluate the conductance measure
a = sum(triu(Au1));
b = sum(tril(Au1));
assoc = cumsum(a+b);
assoc = min(assoc,D-assoc);
cut = cumsum(b-a);
conduct = cut./assoc;
conduct = conduct(1:end-1);
% show the conductance measure
figure(12)
plot(conduct,'x-')
grid
```

```
title('conductance')
% identify the minimum -> threshold
[~,mpos] = min(conduct);
threshold = mean(v1s(mpos:mpos+1));
disp([' '])
disp('spectral approach')
disp(['   Minimum conductance: ' num2str(conduct(mpos))])
disp(['   Cheeger''s upper bound: ' num2str(sqrt(2*DD(2,2)))])
disp(['   # of links: ' num2str(D/2)])
disp(['   Cut value: ' num2str(cut(mpos))])
disp(['   Assoc value: ' num2str(assoc(mpos))])
disp(['   Community size #1: ' num2str(mpos)])
disp(['   Community size #2: ' num2str(N-mpos)])
disp([' '])

% community identified by sign of v1

disp('spectral approach')
disp("based only on sign of Fiedler's vector")
disp(['   Community size #1: ' num2str( sum(v1>0))])
disp(['   Community size #2: ' num2str(sum(v1<0))])
disp([' '])
```

```
spectral approach
   Minimum conductance: 0.11111
   Cheeger's upper bound: 0.32749
   # of links: 9356
   Cut value: 1
   Assoc value: 9
   Community size #1: 5
   Community size #2: 597

spectral approach
based only on sign of Fiedler's vector
   Community size #1: 47
   Community size #2: 555
```

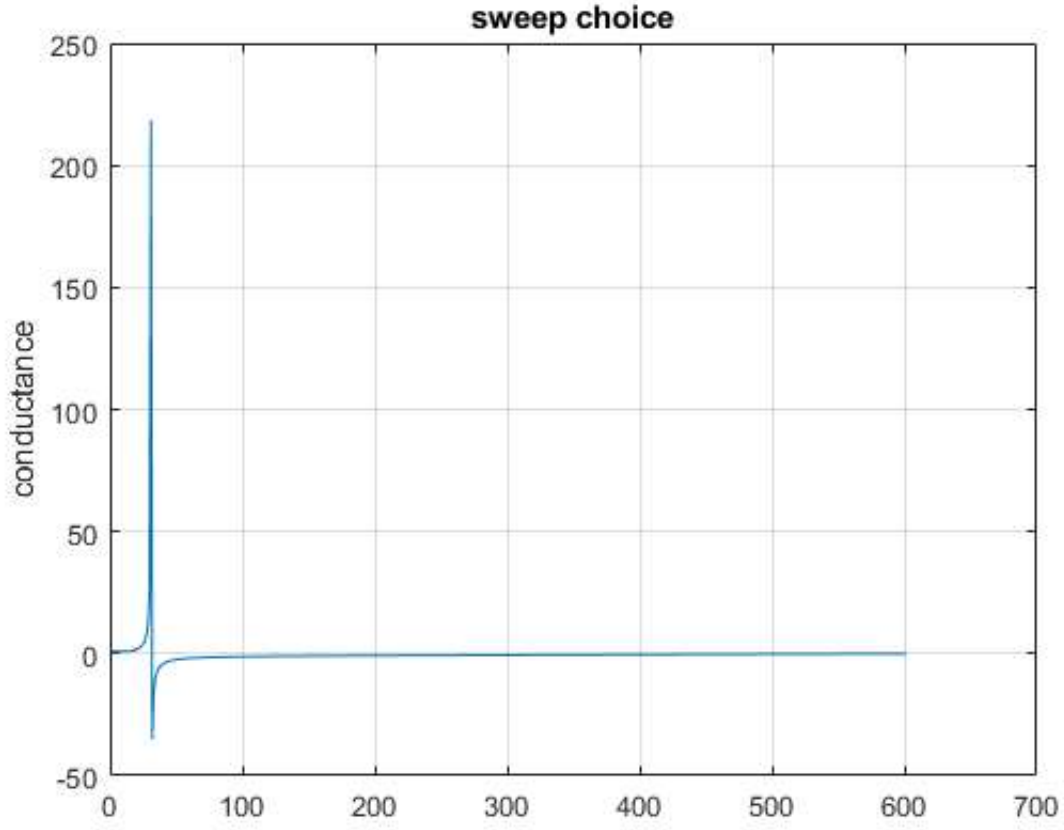


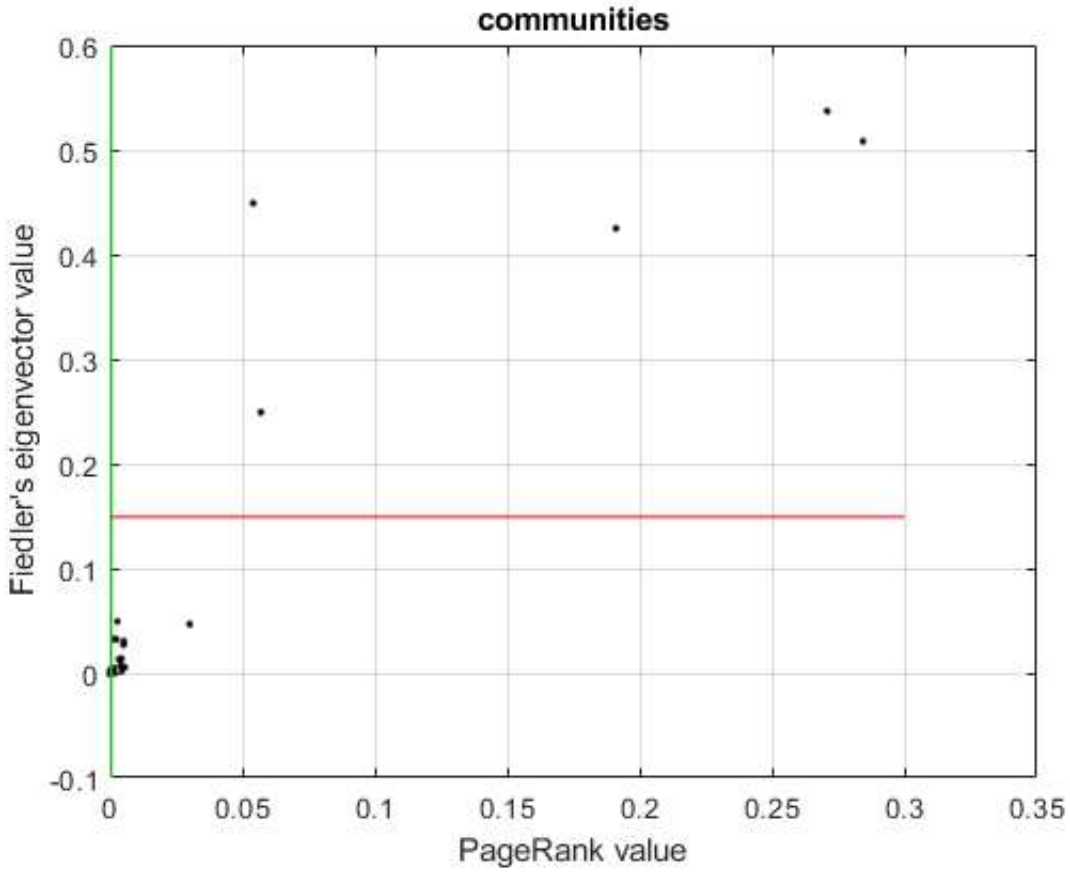
%%%%%%%%%%%%%%%% PageRank-nibble approach for the biggest component %%%%

```
if mpos<N-mpos  % select seed node from the smaller group
    i = pos(1); % we select the more relevant from the perspective of the spectral approac
h
else
    i = pos(end);
end
q = zeros(N,1);
q(i) = 1; % teleport vector
c = 0.85;
r = (I-c*M)\((1-c)*q); % ranking vector
ep = 1e-3; % precision

% run PageRank-nibble
u = zeros(N,1); % starting point
v = q; % starting point
th = full(ep*d/D)'; % thresholds
count = 0; % exit counter
complexity = 0; % complexity value (# of operations)
ii = i; % starting index used for Push operation
while (count<N)
    if v(ii)>th(ii) % push if above threshold
        tmp = v(ii);
        u(ii) = u(ii)+(1-c)*tmp;
        v(ii) = 0;
        v = v + c*M(:,ii)*tmp;
        complexity = complexity + d(ii); % update complexity
        count = 0; % reset the exit counter
    else % go to next entry if below threshold
        count = count + 1; % increase the exit counter
        ii = mod(ii,N)+1; % update the index used for Push
    end
```

```
end

% sweep wrt the ordering identified by v1
% reorder the adjacency matrix
[u1s,pos2] = sort(u,'descend');
Nmax = find(u1s>0,1,'last'); % discard nodes with 0 values (never used in Push)
Bu1 = Bu(pos2,pos2(1:Nmax));
% evaluate the conductance measure
a = sum(triu(Bu1));
b = sum(tril(Bu1));
assoc = cumsum(a+b);
assoc = min(assoc,D-assoc);
cut = cumsum(b-a);
conduct = cut./assoc;
conduct = conduct(1:Nmax-1);
% identify the minimum -> threshold
[~,mpos2] = min(conduct);
threshold2 = mean(u1s(mpos2:mpos2+1));
disp('PageRank-nibble approach')
disp(['   complexity/D: ' num2str((complexity/D))])
disp(['   epsilon: ' num2str(ep)])
disp(['   prec: ' num2str(norm(r-u,1))])
disp(['   Minimum conductance: ' num2str(conduct(mpos2))])
disp(['   # of links: ' num2str(D/2)])
disp(['   Cut value: ' num2str(cut(mpos2))])
disp(['   Assoc value: ' num2str(assoc(mpos2))])
disp(['   Community size #1: ' num2str(mpos2)])
disp(['   Community size #2: ' num2str(N-mpos2)])

% show sweep choice
figure(13)
plot(conduct)
grid
ylabel('conductance')
title('sweep choice')

% show network with partition
figure(14)
plot(u,v1,'k.')
hold on
% plot(u(pos2(1:mpos2)),v1(pos2(1:mpos2)),'go')
plot(threshold2*[1,1],ylim,'g-')
plot(xlim,threshold*[1,1],'r-')
hold off
grid
ylabel('Fiedler''s eigenvector value')
xlabel('PageRank value')
title('communities')
```

```
PageRank-nibble approach
   complexity/D: 20.5076
   epsilon: 0.001
   prec: 0.00048918
   Minimum conductance: -35.0769
   # of links: 9356
   Cut value: 18240
   Assoc value: -520
   Community size #1: 32
   Community size #2: 570
```





Page rank

```
n=size(A,1);
e=ones(n,1);
oo=1./sum(A);
```

```
oo(oo==Inf)=0;
D=diag(oo);
Ad=sparse(A*D);
eAd=sparse(e'*Ad);
check=1;
iter=0;
v=sparse((1/n).*ones(n,1));
toll=1e-4;
alpha=0.85;
toll=0.0001;

while check>toll
    vold=v;
    v=alpha*Ad*vold+1/n*(1-alpha*eAd*vold)*e ;
    check=sum(abs(v-vold));
    iter=iter+1;
end

pr=v;
```

%%%%%%%%%%%%%%%%%%%%%% Show page rank results %%%%%%%%%%%%%%%%%%%%%%%%%%

```
[spr,per]=sort(pr,'descend');
result = table;
result.Parties_name=full_name(per,1);
result.PageRank=spr;
result.Degree = sum(A,2);

result(1:25,:)
```

```
ans =

  25×3 table

                                Parties_name                                PageRank
    Degree
                                      x

    ____________________________________________________________________    _________
    ______

    'Social Democratic Party'                                               0.0072297
       2
    'Democratic Peasants' Party of Germany'                                 0.0071579
      28
    'Human Shield'                                                          0.0064549
      15
    'Latvian Farmers' Union'                                                0.0064145
      65
    'Geneva Citizens' Movement'                                             0.0063279
       1
    'Alliance of Free Democrats'                                            0.0061446
       7
    'Popular Unity'                                                         0.0060599
      10
    'Portuguese Communist Party'                                             0.005937
       4
```

```
'None-of-the-above'                                                    0.0059335
  46
'Homeland Union - Lithuanian Christian Democrats'                      0.0059093
  65
'Direction-Social Democracy'                                           0.0058928
  27
'Alliance of the Christian Democratic People's Front'                  0.0058324
  80
'Radical Political Party'                                              0.0057698
   7
'Blue Coalition'                                                       0.0057555
  42
'Key Coalition'                                                        0.0056872
  36
'Francophone Christian Social Party and Flemish Christian People’s Party'    0.0056668
   3
'Communist Party of Spain'                                             0.0056051
  13
'Democratic Intervention'                                              0.0055033
  13
'People's Party'                                                       0.0052936
  10
'Centre Democrats'                                                     0.0052826
  39
'Coalition of Bulgarian Socialist Party'                               0.0052583
  60
'Aleksandar Vucic - The future in which we believe'                    0.0052325
  70
'Democratic Party'                                                     0.0051644
  52
'Lithuanian Peasants Party'                                            0.0051374
  50
'Belgian Socialist Party'                                              0.0051245
   5
```

Using the function fun_kryl.m compute the total comunicability of every node

```
addpath('.\funm_kryl\')
param.function = @expm;       % other choices: 'expBA', 'expCF', ...
param.restart_length = 10;
param.max_restarts = 50;
param.hermitian = 0;          % set 0 if A is not Hermitian
param.V_full = 0;             % set 1 if you need Krylov basis
param.H_full = 1;             % if using rational functions you can set this 0
param.exact = [];             % if not known set to []
param.bound = 0;              % returns upper and lower bounds (after some cycles)
param.stopping_accuracy = 1e-10;  % stopping accuracy
param.inner_product = @inner_product;
param.thick = [];             % thick-restart function
param.min_decay = 0.95;       % we desire linear error reduction of rate < .95
param.waitbar = 1;            % show waitbar
param.reorth_number = 0;      % #reorthogonalizations
param = param_init(param);    % check and correct param structure

[Tc,out1] = funm_kryl(A,ones(n,1),param);
```

%%%%%%%%%%%%%%%%%%% show result total comunicability

%%%%%%%%%%%%%%%%%%%

```
[sper,peer]=sort(Tc,'descend');

rescom = table;
rescom.PartyName=full_name(peer,1);
rescom.Communicability=sper;
rescom.PageRank=pr(peer);
rescom.Degree = sum(A,2);

rescom(1:25,:)
```

```
ans =

  25×4 table

                                    PartyName                                      Communica
bility     PageRank      Degree
                                        x

    _____________________________________________________________________________    _________
______    _________     ______

    'Human Shield'                                                                    6.2799e
+36        0.0064549       2
    'Democratic Peasants’ Party of Germany'                                           6.2727e
+36        0.0071579      28
    'Latvian Farmers’ Union'                                                          6.0237e
+36        0.0064145      15
    'Geneva Citizens' Movement'                                                       5.9176e
+36        0.0063279      65
    'Social Democratic Party'                                                         5.8828e
+36        0.0072297       1
    'None-of-the-above'                                                               5.8163e
+36        0.0059335       7
    'Communist Party of Spain'                                                        5.5811e
+36        0.0056051      10
    'Portuguese Communist Party'                                                      5.5631e
+36         0.005937       4
    'Francophone Christian Social Party and Flemish Christian People’s Party'         5.5433e
+36        0.0056668      46
    'Radical Political Party'                                                         5.5395e
+36        0.0057698      65
    'Popular Unity'                                                                   5.4342e
+36        0.0060599      27
    'Alliance of the Christian Democratic People's Front'                             5.3917e
+36        0.0058324      80
    'Livable Netherlands'                                                             5.3775e
+36        0.0050172       7
    'Homeland Union - Lithuanian Christian Democrats'                                 5.3487e
+36        0.0059093      42
    'Democratic Intervention'                                                         5.2979e
+36        0.0055033      36
    'Socialist Workers’ Party of Luxembourg'                                          5.2861e
+36        0.0046844       3
    'Direction-Social Democracy'                                                      5.2777e
+36        0.0058928      13
    'Alliance of Free Democrats'                                                       5.166e
+36        0.0061446      13
```

```
    'People's Party'                                                    5.1241e
+36       0.0052936      10
    'Democratic Party'                                                  5.0961e
+36       0.0051644      39
    'Blue Coalition'                                                    5.0145e
+36       0.0057555      60
    'The Left'                                                           4.936e
+36       0.0044422      70
    'Centre Democrats'                                                  4.8786e
+36       0.0052826      52
    'Key Coalition'                                                     4.8747e
+36       0.0056872      50
    'Popular Democratic Movement'                                       4.8313e
+36       0.0050177       5
```

Published with MATLAB® R2018b