MBRW_SPECTRAL_DIMENSION_DEMO

Table of Contents

```
Author: Adam Craig
Created: 2021-09-06.
Last Updated: 2021-09-06
```

```
This code demonstrates the following workflow:
1. Download a protein-protein interaction network file.
2. Extract the direct protein-protein interactions.
3. Extract the 2-core (largest subgraph with minimum degree of 2).
4. Separate the 2-core into its connected components.
5. Save the connected components to a file.
6. Run a memory-biased random walk on the largest connected component.
7. Plot log_2 segment mass generalized means vs segment length.
8. Compute generalized means of walk segments with power-of-2 lengths.
9. Use generalized means to estimate generalized spectral dimensions.
10. Plot the generalized means.
11. Take the range of finite-order generalized spectral dimensions.
Dependencies: The C++ MBRW utility uses methods from the Boost library.
You do not need to install it, just download the C++ header files.
I have tested this code with Boost versions 1.64.0 and 1.75.0.
https://www.boost.org/
The MATLAB scripts make use of the graph object class.
We assume behavior from MATLAB v 2018a or later.
https://www.mathworks.com/help/matlab/graph-and-network-algorithms.html
```

Path to Boost library.

Set this to where you keep the boost header files on your system.

```
boost_path = ['..' filesep 'boost_1_75_0'];
```

Download a protein-protein interaction network file.

```
The network on which we are working is from
Gunsalus, K. C., Ge, H., Schetter, A. J., Goldberg,
D. S., Han, J. D. J., Hao, T., ... & Piano, F. (2005).
```

```
        Predictive models of molecular machines
        involved in Caenorhabditis elegans early embryogenesis.
        Nature, 436(7052), 861-865.
```

```
original_graph_file_url = ['https://static-content.springer.com/' ...
    'esm/art%3A10.1038%2Fnature03876/MediaObjects/' ...
    '41586_2005_BFnature03876_MOESM4_ESM.sif'];
base_network_name = 'gunsalus_2005_c_elegans_ppi';
print_network_name = 'C. elegans PPI from Gunsalus et al, 2005';
networks_dir = 'networks';
gene_names_dir = 'gene_name_lists';
original_graph_file_name = [networks_dir filesep ...
    base_network_name '_original.txt'];
if ~exist(original_graph_file_name,'file')
    disp('Downloading original network...')
    websave(original_graph_file_name, original_graph_file_url);
end
```

Extract the direct protein-protein interactions.

```
direct_interactions_graph_file_name = [networks_dir filesep ...
    base_network_name '_direct.txt'];
direct_interactions_gene_names = [gene_names_dir filesep ...
    base_network_name '_direct_gene_names.txt'];
if ~exist(direct_interactions_graph_file_name,'file') || ...
        ~exist(direct_interactions_gene_names,'file')
    extract_full_gunsalus_et_al_2005( ...
        original_graph_file_name, ...
        direct_interactions_graph_file_name, ...
        direct_interactions_gene_names );
end
```

Extract the largest MBRW-able subgraph.

```
        3. Extract the 2-core (largest subgraph with minimum degree of 2).
        4. Separate the 2-core into its connected components.
        5. Save the connected components to a file.
```

```
mbrw_able_edge_file_name = [networks_dir filesep
 base_network_name '_direct_cc_1.txt'];
mbrw_able_node_names_file_name = [gene_names_dir filesep
 base_network_name '_direct_gene_names_cc_1.txt'];
if ~exist(mbrw_able_edge_file_name,'file') || ...
        ~exist(mbrw_able_node_names_file_name,'file')
    [new_edge_file_names, new_node_name_file_names] = ...
        make_mbrw_able(direct_interactions_graph_file_name, ...
        direct_interactions_gene_names);
    % Just work with the largest connected component.
    mbrw_able_edge_file_name = new_edge_file_names{1};
    mbrw_able_node_names_file_name = new_node_name_file_names{1};
end
G =
 read_graph(mbrw_able_edge_file_name,mbrw_able_node_names_file_name);
```

Run a memory-biased random walk on the largest connected component.

```
bias: How much more likely are we to take a remembered edge
vs an unremembered one, must be a positive integer
In general, values much less than 1000 make the effect of memory weak.
Values much past 10,000 do not lead to
improved sensitivity to community detection.
memory: Number of steps for which to remember an edge,
must be a non-negative integer
In general, past a minimum of 4 or 5,
making memory longer only weakly affects sensitivity.
Most communities in networks contain shorter loops of 3 to 5 edges,
and the walk agent is far more likely to find these than longer ones.
special values:
1 -> no memory except prevention of backtracking.
0 -> no memory, allow backtracking;
(Otherwise, MBRW disallows return to the previous node.)
rand_seed: seed with which to initialize the RNG
We include this for reproducibility.
log_num_steps: log_2(the number of steps for which to walk),
must be a positive integer
Using more steps
increases the accuracy of the estimates of spectral dimension
but also increases the run time.
2^23 steps offers a reasonable compromise for this network.
Larger networks generally need more steps.
The C++ utility prints output that can help you check for convergence.
As a general rule, using a value such that the number after
'min segment mass=' is the number of nodes on the last 3 lines of output
is adequate.
```

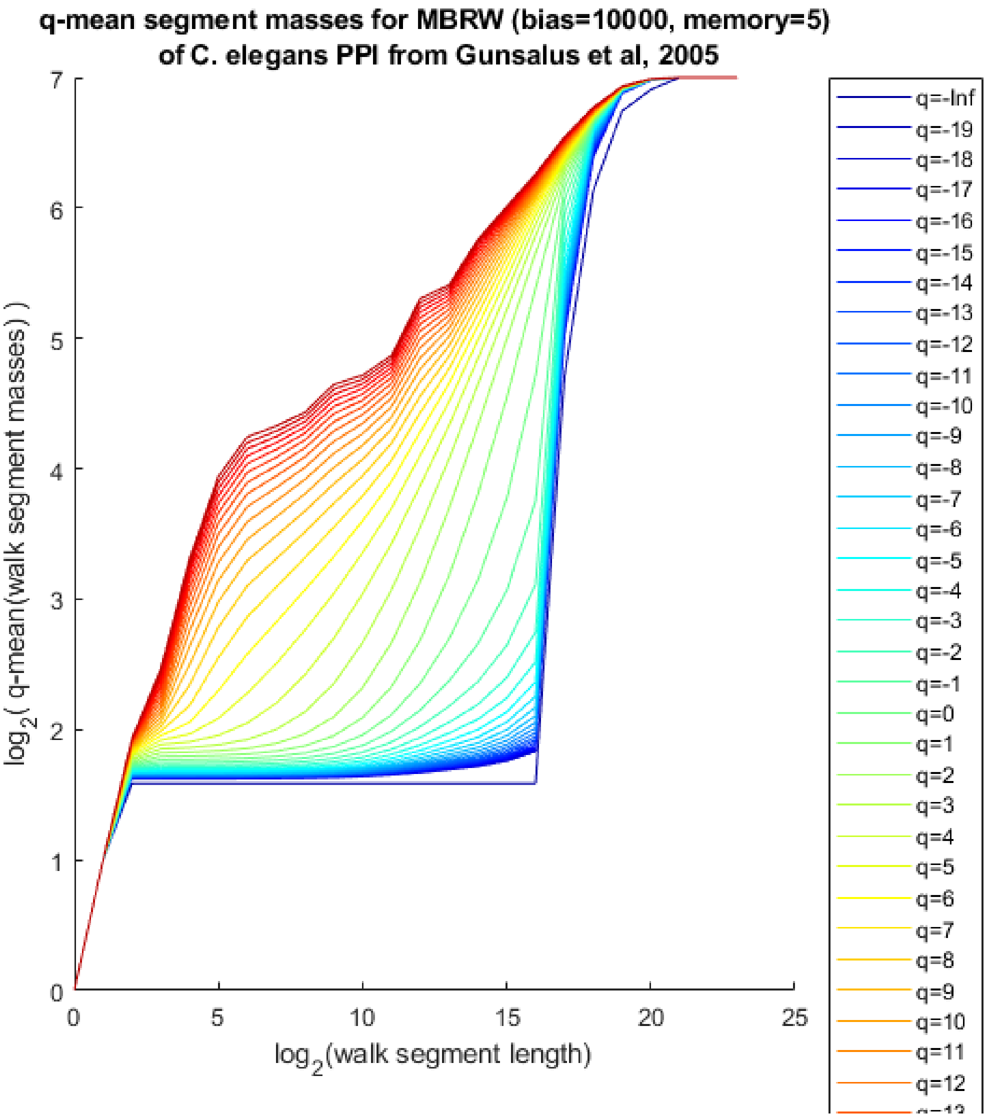
```
% MBRW run parameters.
bias = 10000;
memory = 5;
rand_seed = 0;
log_num_steps = 23;

% Save the orders of generalized mean to a file
% so that the MBRW executable can read them.
finite_orders = -19:19;
orders = [-Inf finite_orders Inf];
num_orders = numel(orders);
order_file_name = 'orders.txt';
% We have to save it as a column vector,
% since the C++ executable tries to parse each line as a single
 number.
writematrix(orders',order_file_name);

segment_mass_log_multimean_file_name = [ ...
    'segment_mass_log_multimeans' filesep
 base_network_name '_cc_1' ...
```

```
    '_b_' num2str(bias) '_m_' num2str(memory) ...
    '_r_' num2str(rand_seed) '_c_' num2str(log_num_steps) '.csv'];
c_executable_name = 'mbrw_and_save_segment_mass_log_multimeans_2';
compile_command = sprintf( 'g++ -std=c++0x -I %s %s.cpp -o %s -
O2', ...
    boost_path, c_executable_name, c_executable_name );
% Windows appends .exe to the file name automatically
% and does not expect ./ before the executable name when running.
% Linux does not append anything to the file name
% and expects ./ before the executable name when running.
if ispc
    dot_if_linux = '';
    exe_if_pc = '.exe';
else
    dot_if_linux = './';
    exe_if_pc = '';
end
run_command = sprintf('%s%s -i %s -q %s -o %s -b %u -m %u -r %u -c
 %u', ...
    dot_if_linux, c_executable_name, ...
    mbrw_able_edge_file_name, ...
    order_file_name, ...
    segment_mass_log_multimean_file_name, ...
    bias, memory, rand_seed, log_num_steps);
if ~exist(segment_mass_log_multimean_file_name,'file')
    if ~exist([c_executable_name exe_if_pc],'file')
        fprintf('compiling %s...\n',c_executable_name)
        tic
        compile_result = system(compile_command);
        toc
        if compile_result
            error('failed to compile %s', c_executable_name)
        end
    end
    fprintf('running %s...\n', c_executable_name)
    tic
    run_result = system(run_command);
    toc
    if run_result
        error('failed to run %s', c_executable_name);
    end
end
```

Plot log_2 segment mass generalized means vs segment length.

```
log2_generalized_means = readmatrix( ...
    segment_mass_log_multimean_file_name, ...
    'FileType','text','Delimiter',',');
num_segment_lengths = size(log2_generalized_means,1);
% Segment lengths are consecutive powers of 2, starting with 2^0 = 1.
log2_segment_lengths = (0:num_segment_lengths-1)';
```

```
line_colors = jet(num_orders);
legend_items = cell(num_orders,1);
figure('Position',[0 0 600 600])
hold on
for o = 1:num_orders
    plot( log2_segment_lengths, log2_generalized_means(:,o), ...
        'Color', line_colors(o,:) )
    legend_items{o} = sprintf( 'q=%i', orders(o) );
end
hold off
legend(legend_items,'Location','northeastoutside')
xlabel('log_2(walk segment length)')
ylabel('log_2( q-mean(walk segment masses) )')
title(  sprintf( ...
    'q-mean segment masses for MBRW (bias=%u, memory=%u)\n of %s', ...
    bias, memory, print_network_name )  )
```



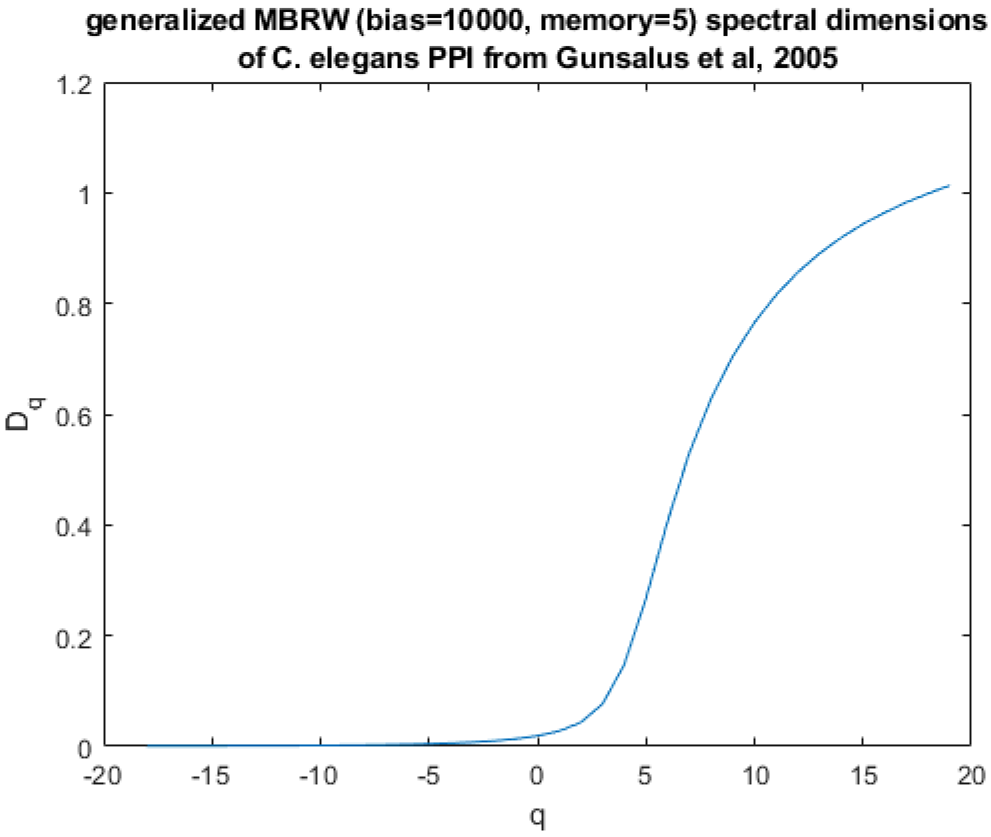
Use generalized means to estimate generalized spectral dimensions.

```
        We need to select a region of segment lengths
        over which to fit a line to each
        log_2(q-mean segment mass) vs log_2(segment length) curve.
        We start with minimum length 4,
        since the no-backtracking rule means all segments have mass at least 3.
        We stop at the last length less than the number of nodes in the network,
        since this is generally short enough that
        walk segments do not exhaust the network.
        For much longer segments,
        the growth rate of segment mass levels off due to finite network size.
```

```
num_nodes = numnodes(G);
Dq = spectral_dimensions_v2(log2_generalized_means, 'length',
 num_nodes);
```

Plot the generalized means.

```
% Do not plot the values at infinity.
order_is_finite = ~isinf(orders);
finite_order_Dqs = Dq(order_is_finite);
figure
plot(finite_orders, finite_order_Dqs)
xlabel('q')
ylabel('D_q')
title(  sprintf( ...
    'generalized MBRW (bias=%u, memory=%u) spectral dimensions\n of %s
 ', ...
    bias, memory, print_network_name )  )
```



Take the range of finite-order generalized spectral dimensions.

We use this range as an order of multi-spectrality.

```
deltaDq = range(finite_order_Dqs);
fprintf('\x0394\x0044=%g\n',deltaDq)
```

```
#D=1.01227
```

Published with MATLAB® R2020b