Computations.hypergraphCentrality

Table of Contents

Syntax 1
Input 1
Name-Value Arguments 1
Output 1
References 1
Code 2

Computes node and edge eigenvector centrality for a hypergraph.

Syntax

```
[N, E] = hypergraphCentrality(HG)
[N, E] = hypergraphCentrality(HG, 'Tolerance', tol)
[N, E] = hypergraphCentrality(HG, 'Tolerance', tol, 'MaxIter', maxIter)
[N, E] = hypergraphCentrality(HG, 'Model', modelName)
[N, E] = hypergraphCentrality(HG, 'Model', modelStruct)
[N, E] = hypergraphCentrality(HG, 'Model', 'Max', 'Alpha', alpha)
```

Input

- HG - Hypergraph object.

Name-Value Arguments

- Tolerance - Convergence tolerance. **Default** : 10e-4
- MaxIter - Maximum number of iterations of the eigenvector centrality algorithm that the function will run before stopping before convergence. **Default** : 3000
- Model: {'Linear', 'LogExp', 'Max'} or cell array of four anonymous funcions. The three string options represent three presets based on the models used in Tudisco's paper. A user can input any four functions for f, g, phi, and psi by passing those functions as a (1,4) cell array of anonymous functions. **Default**: 'LogExp'.
- Alpha: decimal value used in the 'Max' preset model. **Deafult** : 10.

Output

- nodeCentrality - Node centrality scores.
- edgeCentrality - Edge centrality scores.

References

This hypergraph centrality algorithm was obtained from Algorithm (1) in the paper below.

Tudisco, F., Higham, D.J. Node and edge nonlinear eigenvector centrality for hypergraphs. Commun Phys 4, 201 (2021). https://doi.org/10.1038/s42005-021-00704-2

Code

```
function [nodeCentrality, edgeCentrality] = hypergraphCentrality(HG,
 NameValueArgs)
arguments
    HG
    NameValueArgs.Tolerance = 1e-4
    NameValueArgs.MaxIter = 3000
    NameValueArgs.Model = 'LogExp'
    NameValueArgs.Alpha = 10
end
presetModels = {'Linear', 'LogExp', 'Max'};

% default parameters
tol = NameValueArgs.Tolerance;
maxiter = NameValueArgs.MaxIter;
a = NameValueArgs.Alpha; % parameter for 'Max' preset model

% nonlinear functions for f, g, phi, psi. Default is the log-exp scheme
% described in Tudisco's paper.
model = NameValueArgs.Model;
if class(model) == "cell"
    f = model{1};
    g = model{2};
    phi = model{3};
    psi = model{4};
else
    preset = validatestring(model, presetModels);
    switch preset % three presets from the paper.
        case 'Linear'
            f = @(x) x;
            g = @(x) x;
            phi = @(x) x;
            psi = @(x) x;
        case 'LogExp'
            f = @(x) x;
            g = @(x) sqrt(x);
            phi = @(x) log(x);
            psi = @(x) exp(x);
        case 'Max'
            f = @(x) x;
            g = @(x) x;
            phi = @(x) x.^a;
            psi = @(x) x.^(1/a);
    end
end

B = HG.IM;
W = HG.edgeWeights;
N = HG.nodeWeights;
```

```
B = sparse(B);
[n, m] = size(B);

x0 = ones(n,1)./n;
y0 = ones(m,1)./m;
nodeCentrality = nan;
edgeCentrality = nan;

W = sparse(diag(W));
N = sparse(diag(N));

for i = 1:maxiter
    if i<10 || mod(i, 10)==0
        %disp(i);
    end
    u = sqrt(x0.*g(B*W*f(y0)));
    v = sqrt(y0.*psi(B'*N*phi(x0)));
    x = u./norm(u, 1);
    y = v./norm(v, 1);

    check = norm(x-x0, 1)+norm(y-y0, 1);
    if check < tol
        nodeCentrality = full(x);
        edgeCentrality = full(y);
        return
    else
        x0 = x;
        y0 = y;
    end
end

end
```

Published with MATLAB® R2021b