Graph Diffusion

Stefano Fochesatto

May 2, 2023

1 Introduction

Throughout our course we have considered the heat equation for many examples; simple problems with square boundaries and straight forward discretization schemes. The goal of this project is to study diffusion equation with graphs as the domain.

This problem can be thought of as a continuous version of the network flow problems discussed in optimization. Suppose there exists some flow, in most examples we've seen that flow represents heat and in the context of optimization goal was to move flow from one vertex to another in an underlying network as efficiently as possible. Here our goal is to model the state of this flow as a closed system which respects conservation laws.

2 Continuum Model [or Continuum Problem]

For this problem we will be considering weighted undirected graphs. We let each vertex be denoted by v_i, weighted edge by ω_{ij}, and flow (heat) at a vertex by ϕ_i. By Newton's law of cooling, applied between two vertices v_i and v_j we can model the change in flow from v_i to v_j by the following,

$$\frac{\partial \phi_{ij}}{\partial t} = \alpha\omega_{ij}(\phi_i(t) - \phi_j(t)).$$

Note that when $\phi_i(t) > \phi_j(t)$, this quantity is positive and therefore ascribes a positive sign to heat leaving v_i. Describing the change in flow at a single vertex v_i requires us to sum the previous relation across all vertices in the neighborhood of v_i. Note that inverting the sign on the sum gives us a negative quantity when flow is leaving the vertex. Doing so we get the following,

$$\frac{\partial \phi_i}{\partial t} = -\sum_{j:v_j \in \mathcal{N}(v_i)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t)).$$

This results in a system of coupled differential equations, defined by the paths in the graph. Consider small, three vertex path v_a, v_b, v_c and suppose $v_c \notin \mathcal{N}(v_a)$. Note that since $v_b \in \mathcal{N}(v_a), \mathcal{N}(v_c)$ we have the following equations,

$$\begin{aligned}\frac{\partial \phi_a}{\partial t} &= -\sum_{j:v_j \in \mathcal{N}(v_a)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t)) \\ &= -\alpha\omega_{ab}(\phi_a(t) - \phi_b(t)) - \left(\sum_{j:v_j \in \mathcal{N}(v_a)\setminus\{v_b\}} \alpha\omega_{aj}(\phi_a(t) - \phi_j(t))\right),\end{aligned}$$

$$\frac{\partial \phi_c}{\partial t} = -\sum_{j:v_j \in \mathcal{N}(v_c)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t))$$
$$= -\alpha\omega_{cb}(\phi_c(t) - \phi_b(t)) - \left(\sum_{j:v_j \in \mathcal{N}(v_c)\setminus\{v_b\}} \alpha\omega_{cj}(\phi_c(t) - \phi_j(t))\right).$$

Since $\phi_b(t)$ is contained in both equations we can write $\frac{\partial \phi_a}{\partial t}$ with respect to ϕ_c and $\frac{\partial \phi_a}{\partial t}$, and a similar substitution can be extended to paths of arbitrary length. Physically this means that the coupled nature of our system is modeling the fact that flow is moving across path connected vertices not just adjacent vertices.

Since we are working with a weighted undirected graph, in the context of this problem the adjacency and degree matrices are weighted. So the adjacency matrix is filled with the corresponding ω_{ij}, and the degree matrix can just be thought of as a diagonal matrix where the diagonal is a column or row sum of the adjacency. Therefore an edge weighted sum over the neighborhood of a vertex v_i can be described with the i^{th} row of the adjacency matrix like so,

$$\frac{\partial \phi_i}{\partial t} = -\sum_{j:v_j \in \mathcal{N}(v_i)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t)) = -\alpha \sum_{j=1}^{n} A_{ij}(\phi_i(t) - \phi_j(t))$$

Note that the graph Laplacian is defined by $L = D - A$, where D and A are the weighted adjacency and degree matrices we've been working with.[5] With some further algebraic manipulation we find that this whole computation can be written in terms of the graph Laplacian matrix,

$$\begin{aligned}
\frac{\partial \phi_i}{\partial t} &= -\alpha \sum_{j=1}^{n} A_{ij}(\phi_i(t) - \phi_j(t)) \\
&= -\alpha \left(\sum_{j=1}^{n} A_{ij}\phi_i(t) + \sum_{j=1}^{n} A_{ij}\phi_j(t) \right) \\
&= -\alpha \left(\phi_i(t) \sum_{j=1}^{n} A_{ij} + \sum_{j=1}^{n} A_{ij}\phi_j(t) \right) \\
&= -\alpha \left(\deg(\phi_i)\phi_i(t) + \sum_{j=1}^{n} A_{ij}\phi_j(t) \right) \\
&= -\alpha \sum_{j=1}^{n} (\delta_{ij} \deg(\phi_i) - A_{ij})\phi_j(t)
\end{aligned}$$

Where δ_{ij} denotes the kronecker delta function, allowing us to represent an entire row of the graph Laplacian matrix with a single sum. When applied to every vertex in the graph, we form a column vector of flow, $\phi(t)$ and arrive at the formulation of our problem.[1]

$$\frac{\partial \phi}{\partial t} = -\alpha L\phi(t).$$

Connection to the ∇^2 Operator

Interestingly this problem can be seen as a more general discretization stencil for an arbitrary dimension Laplacian. We can see this by considering the following reformulation of $\frac{\partial \phi_i}{\partial t}$,

$$\frac{\partial \phi_i}{\partial t} = -\sum_{j:v_j \in \mathcal{N}(v_i)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t))$$

$$
\begin{aligned}
&= -\sum_{j:v_j \in \mathcal{N}(v_i)} \alpha\omega_{ij}(\phi_i(t) - \phi_j(t)) \\
&= -\alpha\left(\phi_i(t)\sum_{j:v_j \in \mathcal{N}(v_i)} \omega_{ij} + \sum_{j:v_j \in \mathcal{N}(v_i)} \omega_{ij}\phi_j(t)\right) \\
&= \alpha\left(\sum_{j:v_j \in \mathcal{N}(v_i)} \omega_{ij}\right)\left(\frac{\sum_{j:v_j \in \mathcal{N}(v_i)} \omega_{ij}\phi_j(t)}{\sum_{j:v_j \in \mathcal{N}(v_i)} \omega_{ij}} - \phi_i(t)\right)
\end{aligned}
$$

Now consider the 5-point stencil for the Laplacian in two spatial dimensions with a constant diffusivity coefficient i.e. $\alpha\nabla^2$. Recall that using centered finite differences results in the following approximation of the Laplacian at a point $u_{i,j}$,[4]

$$
\alpha\nabla^2(u_{i,j}) = \alpha\frac{4}{h^2}\left(\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{4} - u_{i,j}\right).
$$

This stencil resembles a row of the graph Laplacian corresponding to a vertex with 4 neighbors, where the weights ω_j have all been forced to $1/h^2$. Here, adding more spatial dimensions to the Laplacian would correspond to a vertex with more neighbors, and incorporating unequal spacing would force unequal weights.

Analytical Solution

First we will show that the graph Laplacian L is positive semi-definite. Showing this, along with the fact that L is symmetric would imply that L has real, non-negative eigenvalues. We proceed by considering the following, let $\phi \in \mathbb{R}^n$ and consider $\phi^T L\phi$. From our previous computations we know that the i^{th} term in $(L\phi)$ contains a sum over the corresponding row in the adjacency matrix A. We consider computing $\phi^T(L\phi)$ entry-wise and by substitution we get the following,

$$
\begin{aligned}
\phi^T L\phi &= \sum_{i=1}^{n} \phi_i(L\phi)_i \\
&= \sum_{i=1}^{n} \phi_i\left(\sum_{j=1}^{n} A_{ij}(\phi_i - \phi_j)\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i^2 - \phi_i\phi_j) \\
&= \frac{2}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i^2 - \phi_i\phi_j) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}2(\phi_i^2 - \phi_i\phi_j) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i^2 - 2\phi_i\phi_j + \phi_i^2)
\end{aligned}
$$

Now note that by switching the order of the sums, swapping the indexing variables, and applying the fact that A is a symmetric matrix we get,

$$
\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i^2) = \sum_{j=1}^{n}\sum_{i=1}^{n} A_{ij}(\phi_i^2) = \sum_{j=1}^{n}\sum_{i=1}^{n} A_{ji}(\phi_j^2) = \sum_{j=1}^{n}\sum_{i=1}^{n} A_{ij}(\phi_j^2).
$$

So by substitution we get the following,

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i^2 - 2\phi_i\phi_j + \phi_j^2),$$
$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(\phi_i - \phi_j)^2.$$

Note that this computation becomes a weighted sum of square differences across all adjacent vertices which will always be greater than or equal to zero. Therefore we conclude that $\phi^T L\phi \geq 0$ and L is positive semi-definite.

An even stronger result states that $\lambda_1 = 0$ with multiplicity equal to the number of connected components of the graph, and that the first non-null eigenvalue (Fiedler Value) is a measure of how connected the graph is.[3]

We also find that for connected graphs, like the one in question the eigenvectors form an orthonormal basis.[2]

Finally we formulate the analytical solution, in the same way we would construct a solution for a system of ODE using the matrix exponential. Let $\eta = \phi(0)$, and since L is diagonalizable with $|\lambda_i| \geq 0$ since it is SPD, and with orthogonal eigenvectors we get the following,

$$\phi(t) = Re^{-\alpha\Lambda t}R^T\eta.$$

3 Numerical Scheme(s)

As stated this problem is a system of ODE IVP, and therefore an application of any of the ODE IVP schemes discussed throughout the course would be suitable. As we have previously mentioned, the first non-null eigenvalue of the Laplacian (Fiedler Value) gives a measurement of how well connected the graph is. For our connected graph we know that $\lambda_1 = 0$ and since our eigenvalues are all real, the stiffness ratio of L will actually be a useful measure of stiffness. Since the stiffness ratio will be at least as large as λ_2, it follows that for highly connected graphs this problem becomes stiff.

For our numerical analysis portion of this project we will proceed on two variation of this problem, a minimally connected graph and a highly connected graph applying both forward and backward Euler.

The following are the codes for implementing forward and backward Euler on this system.

Code:

```
function [tt,zz] = GraphFEuler(a, f,eta,t0,tf,N)
% GraphLaplacianForwardEuler  Solve
%   phi'(t) = -a*L*phi(t) ,  phi(0) = eta
% for phi(t) on the interval [t0,tf] with N steps in time.
% f(t,phi) @(t, x) = -alpha*L*x
%
% Usage:[tt,zz] = GraphFEuler(f,eta,t0,tf,N)


dt = (tf - t0) / N;
tt = t0:dt:tf;          % row vector of times
eta = eta(:);           % force to be column vector
s = length(eta);
zz = zeros(s,N+1);      % jth column is U at t_{j-1}
zz(:,1) = eta;
```

```
for j = 1:N           % forward Euler is (5.19) in LeVeque
    zz(:,j+1) = zz(:,j) + dt .* -a.*f(tt(j),zz(:,j));
end
```

Code:

```
function [tt,zz] = GraphBEuler(a,L,eta,t0,tf,N)
% GraphLaplacianBackwardEuler  Solve
%   phi'(t) = -a*L*phi(t) ,  phi(0) = eta
% for phi(t) on the interval [t0,tf] with N steps in time.
% f(t,phi) @(t, x) = -alpha*L*x
%
% Usage:[tt,zz] = GraphFEuler(a,f,eta,t0,tf,N)


dt = (tf - t0) / N;
tt = t0:dt:tf;          % row vector of times
eta = eta(:);           % force to be column vector
s = length(eta);
zz = zeros(s,N+1);      % jth column is U at t_{j-1}
zz(:,1) = eta;
fU = speye(s,s) - dt*-a*L;
for j = 1:N         % Backward Euler Solve
    zz(:,j+1) = (fU)\zz(:,j);
end
```

4 Analysis

For the analysis I generated two random graphs on 10 vertices. Both graphs where forced to be connected, and the weights ω_i were chosen from a normal distribution with $\mu = 0$ and $\sigma = 4$ then the the absolute value was taken. Connectivity was controlled using samples from a uniform distribution. The following are the graphs that were generated for the convergence analysis,

Figure 1: Low Connectivity Graph. Edge weights ω_i are shown.

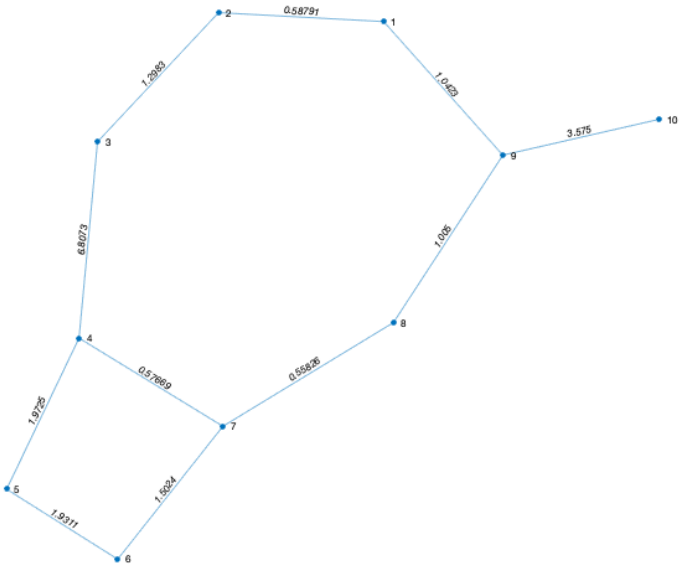
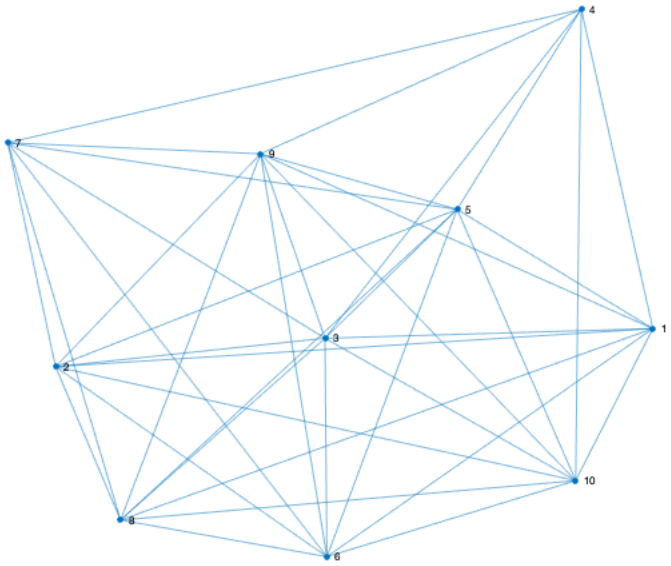


Figure 2: High Connectivity Graph. Edge weights ω_i are omitted.



Here is all the code used to run the convergence analysis and generate these graphs.

Code:

```
% Generate Random Graphs.
% Generating some random weights
```

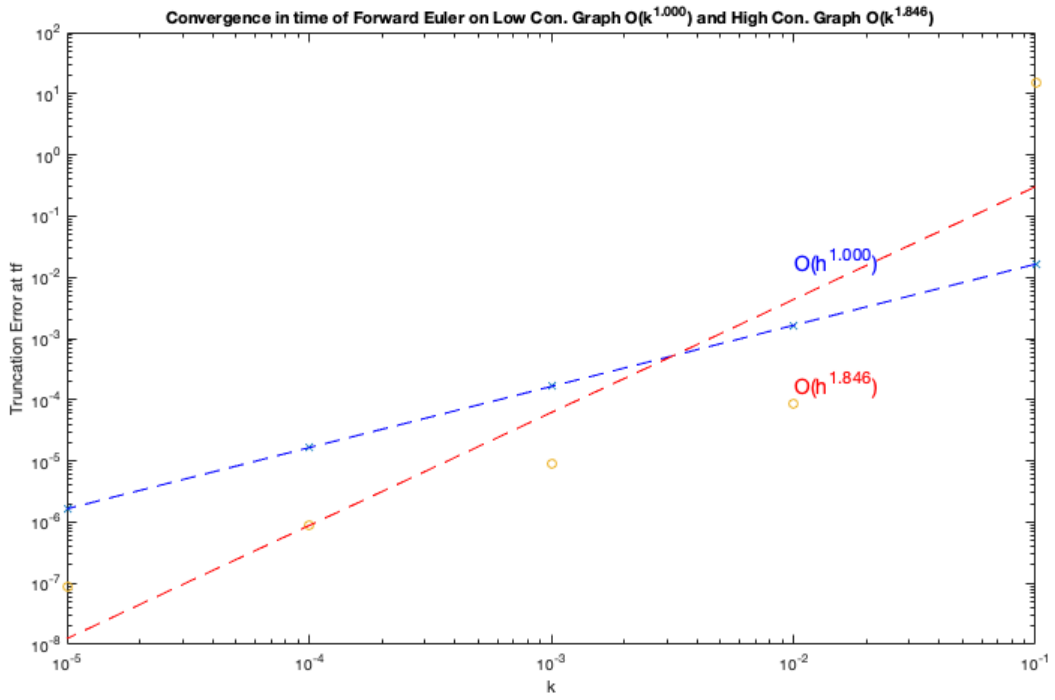
```
n = 10; % number of vertices
Weights = abs(normrnd(0,4, [n,n]));


AHigh = zeros(n, n);
ALow = zeros(n, n);

% Force connected graphs
AHigh(2,1)= Weights(2,1);
ALow(2,1)= Weights(2,1);
for i= 3:n
    AHigh(i,i - 1)= Weights(i,i - 1);
    ALow(i,i - 1)= Weights(i,i - 1);
end


% Iterate and add weights randomly
for i = 2:n
    for j = 1:i-1
        if rand(1) <.75
            AHigh(i, j) = Weights(i, j);
        end
        if rand(1) <.1
            ALow(i, j) = Weights(i, j);
        end

    end
end

% Force symmetric for an actual adjacency matrix
AHigh = (AHigh + AHigh.'-(diag(AHigh)));
ALow = (ALow + ALow.'-(diag(ALow)));

% Compute degree matrix
DHigh = diag(sum(AHigh));
DLow = diag(sum(ALow));

% Compute graph Laplacian
LHigh = DHigh-AHigh;
LLow = DLow-ALow;

% Graph objects for plotting
GHigh = graph(AHigh);
GLow = graph(ALow);


% TimeStepping and Convergence Analysis
eta = [0 0 0 0 0 1 0 0 0 0]';
fH = @(t, x) LHigh*x;
fL = @(t, x) LLow*x;
alpha = .5;

[VL,DL] = eig(LLow);
[VH,DH] = eig(LHigh);

zzExactLow = @(t)(VL*diag(exp(-.5*diag(DL)*t))*VL')*eta;
zzExactHigh = @(t)(VH*diag(exp(-.5*diag(DH)*t))*VH')*eta;

TimeStepRefinement = [10, 100, 1000, 10000, 100000];
```

```
errorFLow = [];
errorBLow = [];
errorFHigh = [];
errorBHigh = [];
for i = 1:length(TimeStepRefinement)
    [ttfLow,zzfLow] = GraphFEuler(alpha, ...
        fL,eta,0,1,TimeStepRefinement(i));
    [ttbLow,zzbLow] = GraphBEuler(alpha, ...
        LLow,eta,0,1,TimeStepRefinement(i));

    [ttfHigh,zzfHigh] = GraphFEuler(alpha, ...
        fH,eta,0,1,TimeStepRefinement(i));
    [ttbHigh,zzbHigh] = GraphBEuler(alpha, ...
        LHigh,eta,0,1,TimeStepRefinement(i));
    errorFLow = [errorFLow max(abs(zzfLow(:,end) ...
        - zzExactLow(1)))];
    errorBLow = [errorBLow max(abs(zzbLow(:,end) ...
        - zzExactLow(1)))];


    errorFHigh = [errorFHigh max(abs(zzfHigh(:,end) ...
        - zzExactHigh(1)))];
    errorBHigh = [errorBHigh max(abs(zzbHigh(:,end) ...
        - zzExactHigh(1)))];
end

TimeStepSize = 1./TimeStepRefinement;

pFLow = polyfit(log(TimeStepSize),log(errorFLow),1);
pBLow = polyfit(log(TimeStepSize),log(errorBLow),1);


pFHigh = polyfit(log(TimeStepSize),log(errorFHigh),1);
pBHigh = polyfit(log(TimeStepSize),log(errorBHigh),1);


loglog(TimeStepSize,errorFLow,'X',TimeStepSize, ...
    exp(pFLow(2) + pFLow(1)*log(TimeStepSize)),'b--')
hold on
loglog(TimeStepSize,errorFHigh,'o',TimeStepSize, ...
    exp(pFHigh(2) + pFHigh(1)*log(TimeStepSize)),'r--')
xlabel k,  ylabel('Truncation Error at tf')

text(0.01,0.02,sprintf('O(h^{%.3f})',pFLow(1)),'Color','b','FontSize',14)
text(0.01,0.0002,sprintf('O(h^{%.3f})',pFHigh(1)),'Color','r','FontSize',14)
title(sprintf(['Convergence in time of ' ...
    'Forward Euler on Low Con. Graph ' ...
    'O(k^{%.3f}) and High Con. ' ...
    'Graph O(k^{%.3f})'],pFLow(1),pFHigh(1)))



hold off



loglog(TimeStepSize,errorBLow,'X',TimeStepSize,exp(pBLow(2) + pBLow(1)*log(TimeStepSize)),'b--')
hold on
```

```
loglog(TimeStepSize,errorBHigh,'o',TimeStepSize,exp(pBHigh(2) + pBHigh(1)*log(TimeStepSize)),'r--')
xlabel k,  ylabel('Truncation Error at tf')

text(0.01,0.02,sprintf('O(h^{%.3f})',pBLow(1)),'Color','b','FontSize',14)
text(0.01,0.0002,sprintf('O(h^{%.3f})',pBHigh(1)),'Color','r','FontSize',14)
title(sprintf(['Convergence in time ' ...
    'of Backward Euler on Low Con. ' ...
    'Graph O(k^{%.3f}) and High Con. ' ...
    'Graph O(k^{%.3f})'],pBLow(1),pBHigh(1)))
```

5 Results

For the convergence analysis we used a refinement grid of $N = [10, 100, 1000, 10000, 100000]$, a one hot initial value η and a $t_f = 1$. Below is a comparison of the convergence of t_f of the two graphs using forward Euler,



This analysis showed that for the higher connectivity graph the convergence was better. Looking at the eigenvalues of the graph Laplacians we find that the higher connectivity graph has a larger eigenvalue by a magnitude of 3,

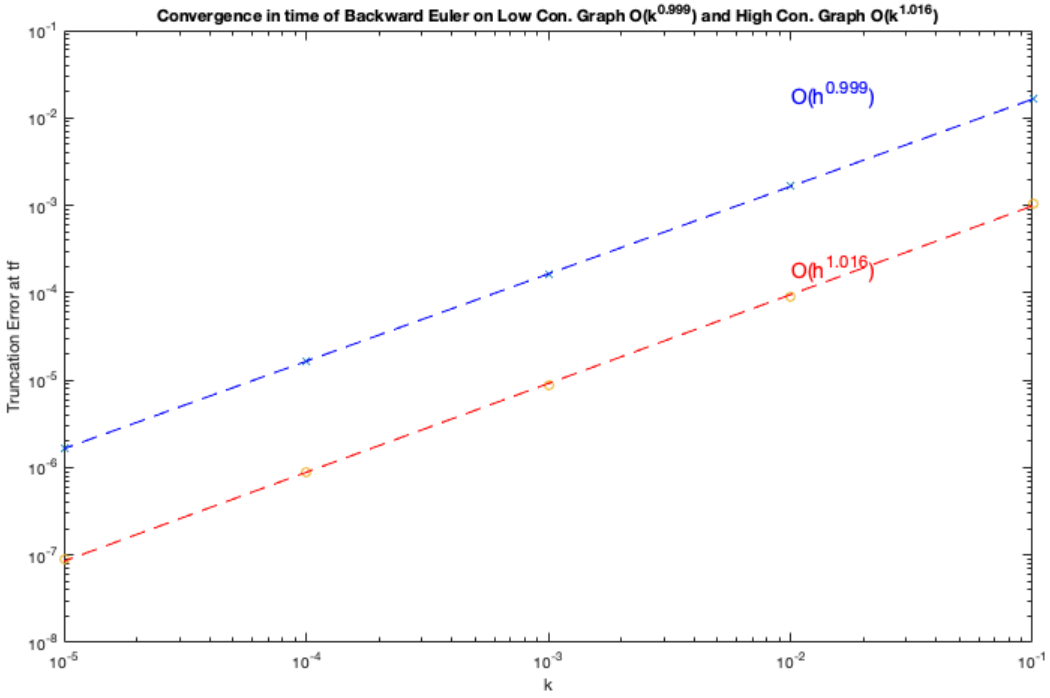
Console:

```
>> LLP = eig(LLow)          >> LHP = eig(LHigh)

    0.000000000000002        0.000000000000004
    0.339195563115088        9.238197399381781
    0.942195998227006        13.918834877793714
```

1.797150818398418	20.069177852110169
1.935679944052389	21.475489129372754
2.792485530496582	25.100651808687552
3.479685801908807	30.191819136112116
6.090471995616983	33.151749992155352
8.515148524375055	43.442742334878872
15.821819211362842	47.315781895946444

Despite the higher connectivity graph producing a more stiff problem, it's convergence rate, using forward Euler was faster than the less stiff problem produced by the lower connectivity graph.

The backward Euler scheme produced a more expected result, but still the higher connectivity graph has faster convergence.



References

[1] Cory Simon. (n.d.). *Diffusion on graphs*. Retrieved April 3, 2023, from https://simonensemble.github.io/pluto_nbs/graph_diffusion_blog.jl.html

[2] Horaud, R., & Horaud, R. (n.d.). A Short Tutorial on Graph Laplacians, Laplacian Embedding, and Spectral Clustering.

[3] Poignard, C., Pereira, T., & Pade, J. P. (2017). Spectra of Laplacian matrices of weighted graphs: Structural genericity properties (arXiv:1704.01677). arXiv. https://doi.org/10.48550/arXiv.1704.01677

[4] R. LeVeque (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*, SIAM Press.

[5] Naumann, U.& Schenk, O. (2012). *Combinatorial Scientific Computing*. CRC Press.