Spectral Methods and Graph Embeddings

Chris Nemeth

23rd April 2025

Outline

- We begin with **spectral graph theory fundamentals** (graphs, adjacency matrices, Laplacians, eigenvalues and eigenvectors).
- **Spectral clustering**, including graph cut objectives (RatioCut, Normalised Cut) and their relaxation via eigenvectors.
- xc,./**Laplacian Eigenmaps** and connections to manifold learning.
- **Node embedding approaches:** matrix factorisation methods (like using adjacency spectra) and random-walk based methods (DeepWalk, node2vec), highlighting their connections to spectral techniques.

Outline

- We begin with **spectral graph theory fundamentals** (graphs, adjacency matrices, Laplacians, eigenvalues and eigenvectors).
- **Spectral clustering**, including graph cut objectives (RatioCut, Normalised Cut) and their relaxation via eigenvectors.
- xc,./**Laplacian Eigenmaps** and connections to manifold learning.
- **Node embedding approaches:** matrix factorisation methods (like using adjacency spectra) and random-walk based methods (DeepWalk, node2vec), highlighting their connections to spectral techniques.

Outline

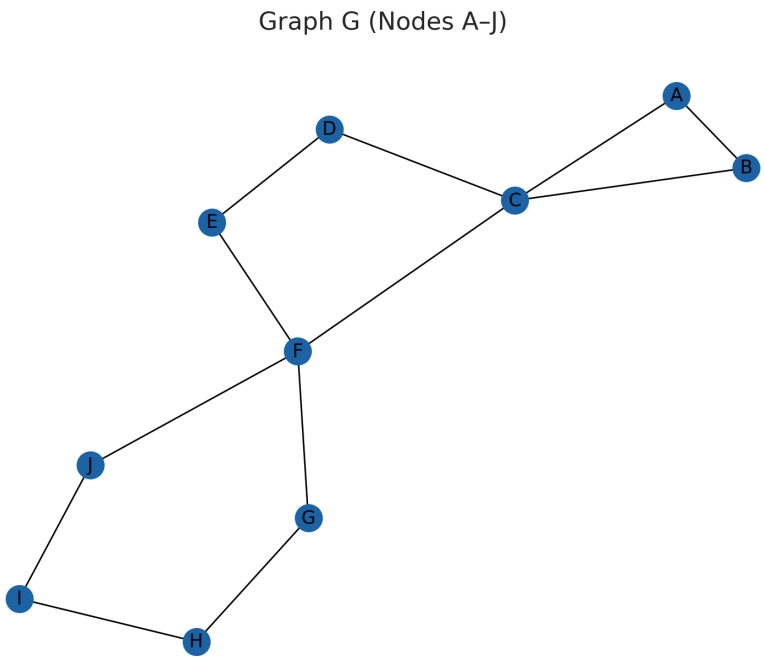
- We begin with **spectral graph theory fundamentals** (graphs, adjacency matrices, Laplacians, eigenvalues and eigenvectors).
- **Spectral clustering**, including graph cut objectives (RatioCut, Normalised Cut) and their relaxation via eigenvectors.
- xc,./**Laplacian Eigenmaps** and connections to manifold learning.
- **Node embedding approaches:** matrix factorisation methods (like using adjacency spectra) and random-walk based methods (DeepWalk, node2vec), highlighting their connections to spectral techniques.

Outline

- We begin with **spectral graph theory fundamentals** (graphs, adjacency matrices, Laplacians, eigenvalues and eigenvectors).
- **Spectral clustering**, including graph cut objectives (RatioCut, Normalised Cut) and their relaxation via eigenvectors.
- xc,./**Laplacian Eigenmaps** and connections to manifold learning.
- **Node embedding approaches:** matrix factorisation methods (like using adjacency spectra) and random-walk based methods (DeepWalk, node2vec), highlighting their connections to spectral techniques.

Spectral Graph Theory Fundamentals

Graphs and Matrices



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Graphs, Matrices, and Spectra

Graph $G = (V, E)$**:** V nodes, E edges (assume undirected, possibly weighted).

- **Adjacency matrix** A**:** $A_{ij} = 1$ if edge i-j exists (or weight if weighted), else 0. Symmetric for undirected graphs.
- **Degree matrix** D**:** diagonal matrix with $D_{ii} = \deg(i) = \sum_j A_{ij}$.
- **Graph Laplacian** L**:** $L = D - A$ (combinatorial Laplacian).
- **Properties:**
  - L is symmetric (for undirected G) and positive semi-definite.
  - $L\mathbf{1} = \mathbf{0}$ (since $D\mathbf{1} = A\mathbf{1}$), so 0 is an eigenvalue.
- **Normalised Laplacians:**
  - $L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ (symmetrised)
  - $L_{\text{rw}} = D^{-1} L = I - D^{-1} A$ (random-walk Laplacian).

Graphs, Matrices, and Spectra

Graph $G = (V, E)$**:** V nodes, E edges (assume undirected, possibly weighted).

- **Adjacency matrix** A**:** $A_{ij} = 1$ if edge i-j exists (or weight if weighted), else 0. Symmetric for undirected graphs.
- **Degree matrix** D**:** diagonal matrix with $D_{ii} = \deg(i) = \sum_j A_{ij}$.
- **Graph Laplacian** L**:** $L = D - A$ (combinatorial Laplacian).
- **Properties:**
  - L is symmetric (for undirected G) and positive semi-definite.
  - $L\mathbf{1} = \mathbf{0}$ (since $D\mathbf{1} = A\mathbf{1}$), so 0 is an eigenvalue.
- **Normalised Laplacians:**
  - $L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ (symmetrised)
  - $L_{\text{rw}} = D^{-1} L = I - D^{-1} A$ (random-walk Laplacian).

Graphs, Matrices, and Spectra

Graph $G = (V, E)$**:** V nodes, E edges (assume undirected, possibly weighted).

- **Adjacency matrix** A**:** $A_{ij} = 1$ if edge i-j exists (or weight if weighted), else 0. Symmetric for undirected graphs.
- **Degree matrix** D**:** diagonal matrix with $D_{ii} = \deg(i) = \sum_j A_{ij}$.
- **Graph Laplacian** L**:** $L = D - A$ (combinatorial Laplacian).
- **Properties:**
  - L is symmetric (for undirected G) and positive semi-definite.
  - $L\mathbf{1} = \mathbf{0}$ (since $D\mathbf{1} = A\mathbf{1}$), so 0 is an eigenvalue.
- **Normalised Laplacians:**
  - $L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ (symmetrised)
  - $L_{\text{rw}} = D^{-1} L = I - D^{-1} A$ (random-walk Laplacian).

Graphs, Matrices, and Spectra

Graph $G = (V, E)$**:** V nodes, E edges (assume undirected, possibly weighted).

- **Adjacency matrix** A**:** $A_{ij} = 1$ if edge i-j exists (or weight if weighted), else 0. Symmetric for undirected graphs.
- **Degree matrix** D**:** diagonal matrix with $D_{ii} = \deg(i) = \sum_j A_{ij}$.
- **Graph Laplacian** L**:** $L = D - A$ (combinatorial Laplacian).
- **Properties:**
  - L is symmetric (for undirected G) and positive semi-definite.
  - $L\mathbf{1} = \mathbf{0}$ (since $D\mathbf{1} = A\mathbf{1}$), so 0 is an eigenvalue.
- **Normalised Laplacians:**
  - $L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ (symmetrised)
  - $L_{\text{rw}} = D^{-1} L = I - D^{-1} A$ (random-walk Laplacian).

Graphs, Matrices, and Spectra

Graph $G = (V, E)$**:** V nodes, E edges (assume undirected, possibly weighted).

- **Adjacency matrix** A**:** $A_{ij} = 1$ if edge i-j exists (or weight if weighted), else 0. Symmetric for undirected graphs.
- **Degree matrix** D**:** diagonal matrix with $D_{ii} = \deg(i) = \sum_j A_{ij}$.
- **Graph Laplacian** L**:** $L = D - A$ (combinatorial Laplacian).
- **Properties:**
  - L is symmetric (for undirected G) and positive semi-definite.
  - $L\mathbf{1} = \mathbf{0}$ (since $D\mathbf{1} = A\mathbf{1}$), so 0 is an eigenvalue.
- **Normalised Laplacians:**
  - $L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ (symmetrised)
  - $L_{\text{rw}} = D^{-1} L = I - D^{-1} A$ (random-walk Laplacian).

Eigenvalues and Eigenvectors of L

For an n-node connected graph:

- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V\Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Eigenvalues and Eigenvectors of L

For an n-node connected graph:

- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V \Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Eigenvalues and Eigenvectors of L

For an n-node connected graph:

- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V\Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Eigenvalues and Eigenvectors of L

For an n-node connected graph:

- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V\Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Eigenvalues and Eigenvectors of L

For an n-node connected graph:

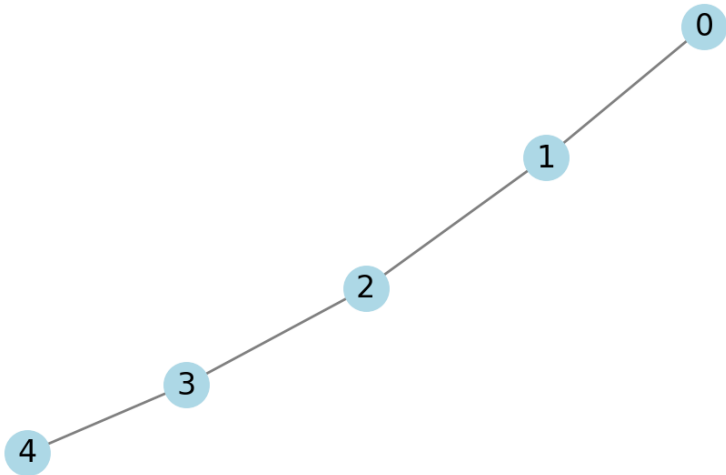
- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V\Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Eigenvalues and Eigenvectors of L

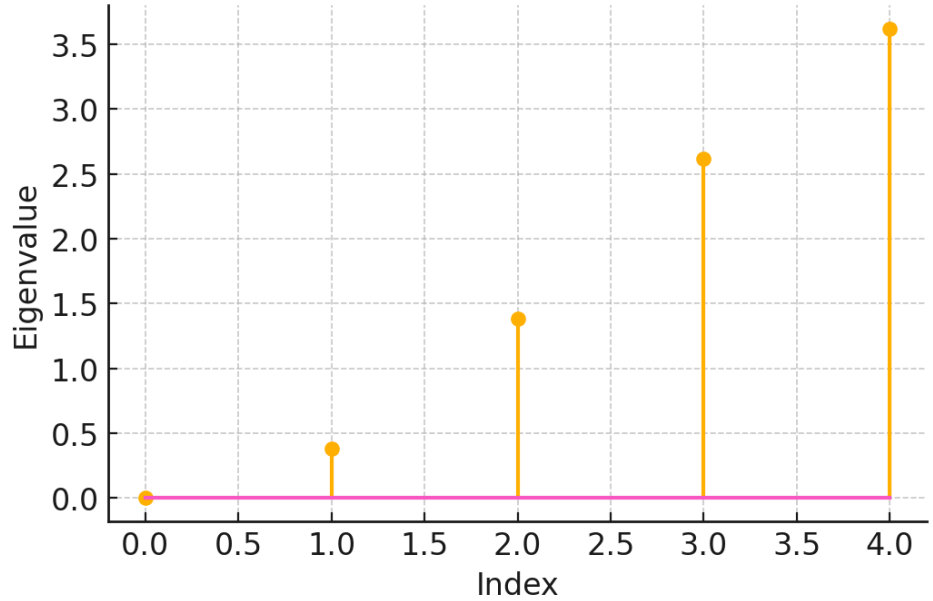
For an n-node connected graph:

- L has n real eigenvalues $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.
- $\lambda_1 = 0$ (eigenvector $\mathbf{1}$).
- Multiplicity of $\lambda = 0$ equals number of connected components.
- **Orthogonal eigenvectors:** $L = V\Lambda V^\top$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.
- **Second-smallest eigenvector:** $\mathbf{v}_2$ (Fiedler vector) associated with λ_2. Often reveals community structure (more later).

Connected Graph (Path of 5 nodes)

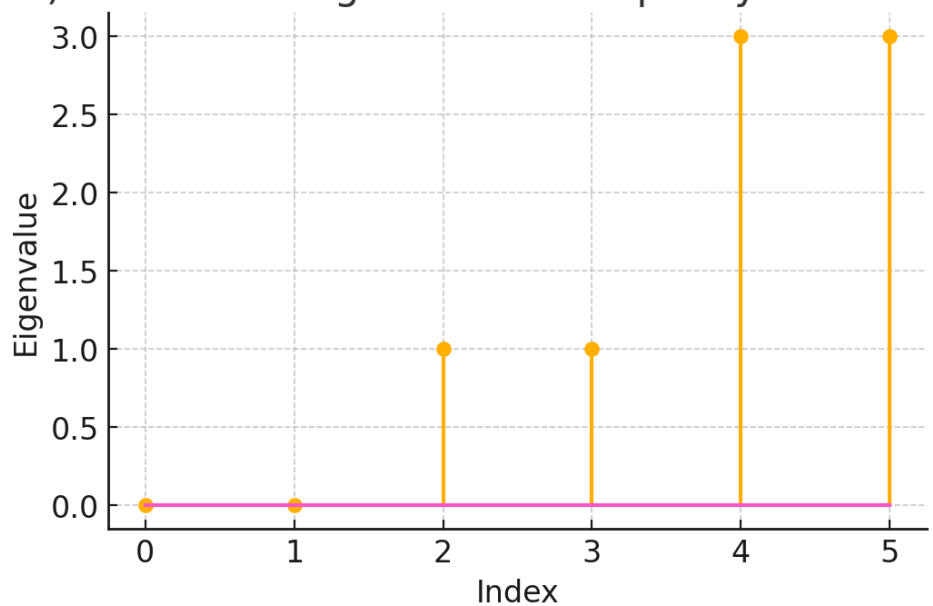


Spectrum (Connected)
Zero eigenvalue multiplicity = 1



Disconnected Graph (2 Components of 3 nodes)

Spectrum (Disconnected)
Zero eigenvalue multiplicity = 2



Rayleigh Quotient and Variational Characterisation

For symmetric L, the Rayleigh quotient of a nonzero vector $\mathbf{x}$ is:

$$R(\mathbf{x}) := \frac{\mathbf{x}^\top L\mathbf{x}}{\mathbf{x}^\top \mathbf{x}}.$$

- Expanding for $L = D - A$:

$$\mathbf{x}^\top L\mathbf{x} = \sum_{i,j} A_{ij}(x_i - x_j)^2/2.$$

- By the **Courant-Fischer theorem:**

$$\lambda_2 = \min_{\mathbf{x}\perp\mathbf{1}} \frac{\mathbf{x}^\top L\mathbf{x}}{\mathbf{x}^\top \mathbf{x}}.$$

- The minimum is attained by $\mathbf{x} = \mathbf{v}_2$ (Fiedler vector).
- Constraint $\mathbf{x} \perp \mathbf{1}$ (orthogonal to $\mathbf{1}$) ensures we skip the trivial eigenvector.
- Thus λ_2 is the smallest non-zero eigenvalue, solving $\min \mathbf{x}^\top L\mathbf{x}$ s.t. $\mathbf{x}^\top \mathbf{x} = 1, \mathbf{x}^\top \mathbf{1} = 0$.

Cheeger's Inequality

- **Conductance (Φ):** For a subset $S \subset V$,

$$\Phi(S) = \frac{\text{cut}(S, \bar{S})}{\min(\text{vol}(S), \text{vol}(\bar{S}))},$$

  where $\text{vol}(S) = \sum_{i \in S} \deg(i)$ and **cut size:** $\text{cut}(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} A_{ij}$ (Total weight of edges crossing the partition S vs complement).

- **Cheeger's inequality:** Relates λ_2 to the best conductance Φ of any cut:

$$\frac{\lambda_2}{2} \leq \Phi \leq \sqrt{2\lambda_2}.$$

- **Interpretation:** A small λ_2 implies existence of a cut with small conductance (a good balanced partition). Conversely, a strong spectral gap (large λ_2) indicates the graph is well-connected (no very sparse cut).

Cheeger's Inequality

- **Conductance (Φ):** For a subset $S \subset V$,

$$\Phi(S) = \frac{\text{cut}(S, \bar{S})}{\min(\text{vol}(S), \text{vol}(\bar{S}))},$$

  where $\text{vol}(S) = \sum_{i \in S} \deg(i)$ and **cut size:** $\text{cut}(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} A_{ij}$ (Total weight of edges crossing the partition S vs complement).
- **Cheeger's inequality:** Relates λ_2 to the best conductance Φ of any cut:

$$\frac{\lambda_2}{2} \leq \Phi \leq \sqrt{2\lambda_2}.$$

- **Interpretation:** A small λ_2 implies existence of a cut with small conductance (a good balanced partition). Conversely, a strong spectral gap (large λ_2) indicates the graph is well-connected (no very sparse cut).

Cheeger's Inequality

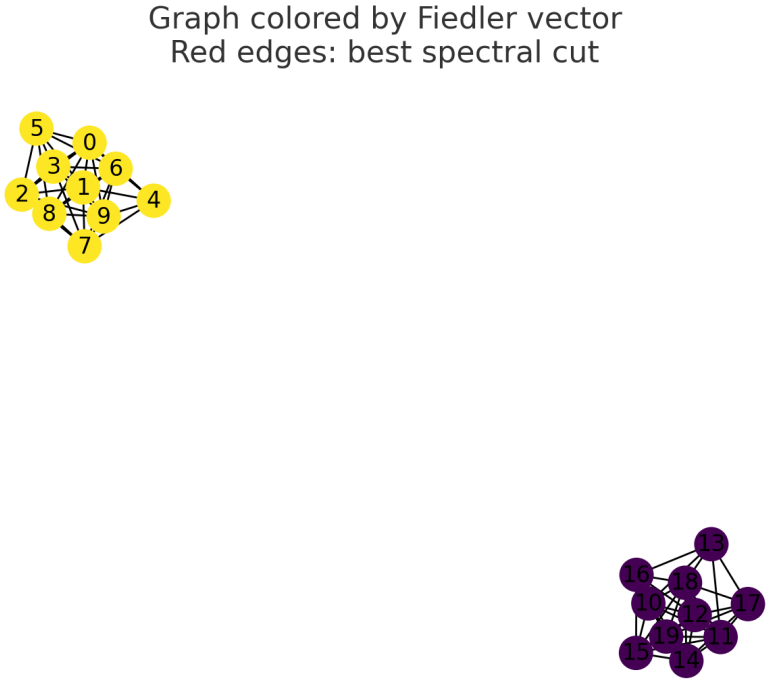
- **Conductance (Φ):** For a subset $S \subset V$,

$$\Phi(S) = \frac{\text{cut}(S, \bar{S})}{\min(\text{vol}(S), \text{vol}(\bar{S}))},$$

  where $\text{vol}(S) = \sum_{i \in S} \deg(i)$ and **cut size:** $\text{cut}(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} A_{ij}$ (Total weight of edges crossing the partition S vs complement).
- **Cheeger's inequality:** Relates λ_2 to the best conductance Φ of any cut:

$$\frac{\lambda_2}{2} \leq \Phi \leq \sqrt{2\lambda_2}.$$

- **Interpretation:** A small λ_2 implies existence of a cut with small conductance (a good balanced partition). Conversely, a strong spectral gap (large λ_2) indicates the graph is well-connected (no very sparse cut).


Graph colored by Fiedler vector
Red edges: best spectral cut
5
0
3
6
2
1
4
8
9
7
13
16
18
10
17
12
19
11
15
14
Sweep cut conductance vs. spectral bounds
1.0
0.8
0.6
0.4
0.2
0.0
Conductance Φ(S)
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Threshold k (size of S)
λ2/2
√2λ2
Φ*

Spectral Clustering

Graph Clustering and Cut Objectives

Goal: Partition graph into k clusters $(V_1, \ldots, V_k)$ such that:

- Many edges inside clusters, few edges between clusters.

Simple objective: minimise $\text{cut}(V_1, \bar{V}_1)$ for a bisection.

Problem: trivial solution can isolate a single node (very small cut but unbalanced clusters).

Graph Clustering and Cut Objectives

Goal: Partition graph into k clusters $(V_1, \ldots, V_k)$ such that:

- Many edges inside clusters, few edges between clusters.

Simple objective: minimise $\text{cut}(V_1, \bar{V}_1)$ for a bisection.

Problem: trivial solution can isolate a single node (very small cut but unbalanced clusters).

Graph Clustering and Cut Objectives

Goal: Partition graph into k clusters $(V_1, \ldots, V_k)$ such that:

- Many edges inside clusters, few edges between clusters.

Simple objective: minimise $\text{cut}(V_1, \bar{V}_1)$ for a bisection.

Problem: trivial solution can isolate a single node (very small cut but unbalanced clusters).

Graph Clustering and Cut Objectives

To encourage balanced partitions:

- **RatioCut:** for a k-partition,

$$\mathrm{Rcut}(V_1, \ldots, V_k) = \sum_{m=1}^{k} \frac{\mathrm{cut}(V_m, \bar{V}_m)}{|V_m|}.$$

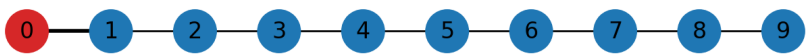
  (Penalise small clusters via $|V_m|$ in denominator.)

- **Normalised Cut (Ncut):**

$$\mathrm{Ncut}(V_1, \ldots, V_k) = \sum_{m=1}^{k} \frac{\mathrm{cut}(V_m, \bar{V}_m)}{\mathrm{vol}(V_m)}.$$

  (Denominator uses volume = sum of degrees.)

Degenerate



cut(S, S) = 1
RatioCut = 1.11
Normalized Cut = 1.06

Balanced

cut(S, S) = 1
RatioCut = 0.40
Normalized Cut = 0.22

Spectral Relaxation of RatioCut (Unnormalised)

RatioCut for $k = 2$:

$$\frac{\text{cut}(S, \bar{S})}{|S|} + \frac{\text{cut}(S, \bar{S})}{|\bar{S}|}$$

(with $S \cup \bar{S} = V$).

Represent a 2-partition by an indicator vector $\mathbf{y} \in 0, 1^n$ (or $\pm 1^n$): e.g. $y_i = 1$ if $i \in S$, 0 if $i \in \bar{S}$. Alternatively use $\mathbf{z} \in \pm 1^n$ with $z_i = \pm 1$ indicating two sides.

One can show:

$$\text{cut}(S, \bar{S}) = \frac{1}{4} \mathbf{z}^\top L \mathbf{z}$$

and $|S| = \frac{1}{2} n + \frac{1}{2} \sum_i z_i$. (For $\mathbf{z} \in \pm 1$, $\sum_i z_i = |S| - |\bar{S}|$.)

Spectral Relaxation of RatioCut (Unnormalised)

RatioCut for $k = 2$:

$$\frac{\text{cut}(S, \bar{S})}{|S|} + \frac{\text{cut}(S, \bar{S})}{|\bar{S}|}$$

(with $S \cup \bar{S} = V$).

Represent a 2-partition by an indicator vector $\mathbf{y} \in 0, 1^n$ (or $\pm 1^n$): e.g. $y_i = 1$ if $i \in S$, 0 if $i \in \bar{S}$. Alternatively use $\mathbf{z} \in \pm 1^n$ with $z_i = \pm 1$ indicating two sides.

One can show:

$$\text{cut}(S, \bar{S}) = \frac{1}{4} \mathbf{z}^\top L \mathbf{z}$$

and $|S| = \frac{1}{2} n + \frac{1}{2} \sum_i z_i$. (For $\mathbf{z} \in \pm 1$, $\sum_i z_i = |S| - |\bar{S}|$.)

Spectral Relaxation of RatioCut (Unnormalised)

RatioCut for $k = 2$:

$$\frac{\text{cut}(S, \bar{S})}{|S|} + \frac{\text{cut}(S, \bar{S})}{|\bar{S}|}$$

(with $S \cup \bar{S} = V$).

Represent a 2-partition by an indicator vector $\mathbf{y} \in 0, 1^n$ (or $\pm 1^n$): e.g. $y_i = 1$ if $i \in S$, 0 if $i \in \bar{S}$. Alternatively use $\mathbf{z} \in \pm 1^n$ with $z_i = \pm 1$ indicating two sides.

One can show:

$$\text{cut}(S, \bar{S}) = \frac{1}{4} \mathbf{z}^\top L \mathbf{z}$$

and $|S| = \frac{1}{2} n + \frac{1}{2} \sum_i z_i$. (For $\mathbf{z} \in \pm 1$, $\sum_i z_i = |S| - |\bar{S}|$.)

Spectral Relaxation of RatioCut (Unnormalised)

- The balanced condition $|S| = |\bar{S}|$ or generally treating $|S|$ as a constant yields a formulation: minimize $\mathbf{z}^\top L\mathbf{z}$ s.t. $z_i \in \pm 1$ and $\mathbf{z}^\top \mathbf{1} = 0$.
- Relax $\mathbf{z}$ to take real values: solve the minimisation $\mathbf{x}^\top L\mathbf{x}$ s.t. $\mathbf{x}^\top \mathbf{1} = 0, \mathbf{x}^\top \mathbf{x} = n$ (some normalisation).
- **Solution:** $\mathbf{x} = \mathbf{v}_2$ (Fiedler vector).
- So the relaxed optimal partition: $S = i : v_{2,i} > 0$ and $\bar{S} = i : v_{2,i} < 0$ (or threshold by median).

Spectral Relaxation of RatioCut (Unnormalised)

- The balanced condition $|S| = |\bar{S}|$ or generally treating $|S|$ as a constant yields a formulation: minimize $\mathbf{z}^\top L\mathbf{z}$ s.t. $z_i \in \pm 1$ and $\mathbf{z}^\top \mathbf{1} = 0$.
- Relax $\mathbf{z}$ to take real values: solve the minimisation $\mathbf{x}^\top L\mathbf{x}$ s.t. $\mathbf{x}^\top \mathbf{1} = 0, \mathbf{x}^\top \mathbf{x} = n$ (some normalisation).
- **Solution:** $\mathbf{x} = \mathbf{v}_2$ (Fiedler vector).
- So the relaxed optimal partition: $S = i : v_{2,i} > 0$ and $\bar{S} = i : v_{2,i} < 0$ (or threshold by median).

Spectral Relaxation of RatioCut (Unnormalised)

- The balanced condition $|S| = |\bar{S}|$ or generally treating $|S|$ as a constant yields a formulation: minimize $\mathbf{z}^\top L\mathbf{z}$ s.t. $z_i \in \pm 1$ and $\mathbf{z}^\top \mathbf{1} = 0$.
- Relax $\mathbf{z}$ to take real values: solve the minimisation $\mathbf{x}^\top L\mathbf{x}$ s.t. $\mathbf{x}^\top \mathbf{1} = 0, \mathbf{x}^\top \mathbf{x} = n$ (some normalisation).
- **Solution:** $\mathbf{x} = \mathbf{v}_2$ (Fiedler vector).
- So the relaxed optimal partition: $S = i : v_{2,i} > 0$ and $\bar{S} = i : v_{2,i} < 0$ (or threshold by median).

Spectral Relaxation of RatioCut (Unnormalised)

- The balanced condition $|S| = |\bar{S}|$ or generally treating $|S|$ as a constant yields a formulation: minimize $\mathbf{z}^\top L\mathbf{z}$ s.t. $z_i \in \pm 1$ and $\mathbf{z}^\top \mathbf{1} = 0$.
- Relax $\mathbf{z}$ to take real values: solve the minimisation $\mathbf{x}^\top L\mathbf{x}$ s.t. $\mathbf{x}^\top \mathbf{1} = 0, \mathbf{x}^\top \mathbf{x} = n$ (some normalisation).
- **Solution:** $\mathbf{x} = \mathbf{v}_2$ (Fiedler vector).
- So the relaxed optimal partition: $S = i : v_{2,i} > 0$ and $\bar{S} = i : v_{2,i} < 0$ (or threshold by median).

Normalised Spectral Clustering (Shi and Malik 2000)

For **Normalised Cut**, we use the normalised Laplacian. The relaxation leads to solving:

$$L_{\text{rw}}\mathbf{x} = \lambda\mathbf{x},$$

with $\mathbf{x}$ as the indicator (generalized eigenproblem $L\mathbf{x} = \lambda D\mathbf{x}$).

Algorithm (Normalised Spectral Clustering for k clusters):

1. Compute k eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ of Lsym (or Lrw) corresponding to the k smallest eigenvalues. ($\mathbf{v}_1 = \mathbf{1}/\sqrt{n}$ is trivial and often discarded for clustering).
2. Form matrix $U \in \mathbb{R}^{n \times k}$ with $U_{i,j} = (\mathbf{v}_j)i$. Each row $U_{i,*}$ is the k-dimensional embedding of node i.
3. (Optionally normalise rows if using L_{sym} to get unit length vectors.)
4. Cluster the points $\{U_{i,*}\}_{i=1}^{n}$ in $\mathbb{R}^k$ using k-means (or another clustering in Euclidean space).
5. Assign nodes to clusters according to the k-means output.

For $k = 2$, this reduces to thresholding $\mathbf{v}2$ as previous slide.

Normalised Spectral Clustering (Shi and Malik 2000)

For **Normalised Cut**, we use the normalised Laplacian. The relaxation leads to solving:

$$L_{\text{rw}}\mathbf{x} = \lambda\mathbf{x},$$

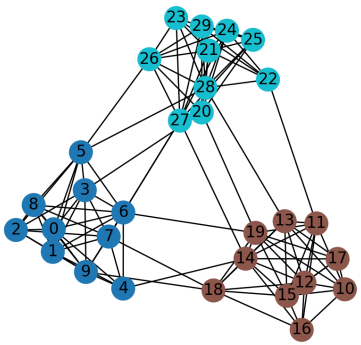
with $\mathbf{x}$ as the indicator (generalized eigenproblem $L\mathbf{x} = \lambda D\mathbf{x}$).

Algorithm (Normalised Spectral Clustering for k clusters):

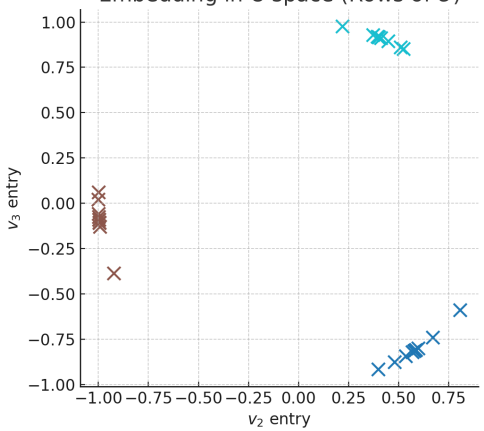
1. Compute k eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ of Lsym (or Lrw) corresponding to the k smallest eigenvalues. ($\mathbf{v}_1 = \mathbf{1}/\sqrt{n}$ is trivial and often discarded for clustering).
2. Form matrix $U \in \mathbb{R}^{n\times k}$ with $U_{i,j} = (\mathbf{v}_j)i$. Each row $U_{i,*}$ is the k-dimensional embedding of node i.
3. (Optionally normalise rows if using L_{sym} to get unit length vectors.)
4. Cluster the points $\{U_{i,*}\}_{i=1}^{n}$ in $\mathbb{R}^k$ using k-means (or another clustering in Euclidean space).
5. Assign nodes to clusters according to the k-means output.

For $k = 2$, this reduces to thresholding $\mathbf{v}2$ as previous slide.

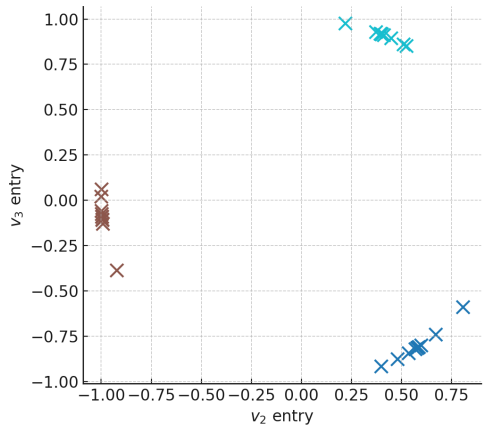
Original Graph (True Clusters)



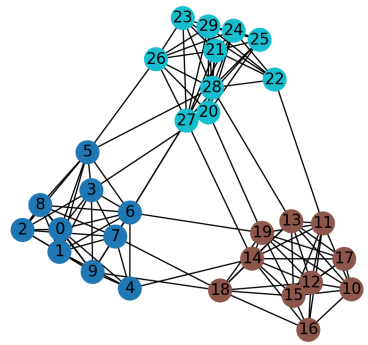
Embedding in U space (Rows of U)



k-means on Rows of U



Graph Colored by Spectral Clustering



Choosing Number of Clusters and Practical Notes

- **How many clusters (k)?** One heuristic: look for an "eigengap" - a large gap in the magnitude of eigenvalues λ_k vs λ_{k+1}. A big jump suggests k meaningful clusters.
- The eigenvectors can be sensitive to graph structure; noise or nearly-equal eigenvalues can cause instability. Using k-means on multiple eigenvectors tends to be more stable for $k > 2$.
- **Normalised vs unnormalised:** Normalised spectral clustering often performs better on imbalanced degree graphs, ensuring each cluster has fair volume. Unnormalized is simpler but may bias toward cutting off small degree nodes.
- **Complexity:** Computing eigenvectors can be expensive ($O(n^3)$ in worst case). For large graphs, use sparse methods or approximate eigen-solvers.

Choosing Number of Clusters and Practical Notes

- **How many clusters (k)?** One heuristic: look for an "eigengap" - a large gap in the magnitude of eigenvalues λ_k vs λ_{k+1}. A big jump suggests k meaningful clusters.
- The eigenvectors can be sensitive to graph structure; noise or nearly-equal eigenvalues can cause instability. Using k-means on multiple eigenvectors tends to be more stable for $k > 2$.
- **Normalised vs unnormalised:** Normalised spectral clustering often performs better on imbalanced degree graphs, ensuring each cluster has fair volume. Unnormalized is simpler but may bias toward cutting off small degree nodes.
- **Complexity:** Computing eigenvectors can be expensive ($O(n^3)$ in worst case). For large graphs, use sparse methods or approximate eigen-solvers.

Choosing Number of Clusters and Practical Notes

- **How many clusters (k)?** One heuristic: look for an "eigengap" - a large gap in the magnitude of eigenvalues λ_k vs λ_{k+1}. A big jump suggests k meaningful clusters.
- The eigenvectors can be sensitive to graph structure; noise or nearly-equal eigenvalues can cause instability. Using k-means on multiple eigenvectors tends to be more stable for $k > 2$.
- **Normalised vs unnormalised:** Normalised spectral clustering often performs better on imbalanced degree graphs, ensuring each cluster has fair volume. Unnormalized is simpler but may bias toward cutting off small degree nodes.
- **Complexity:** Computing eigenvectors can be expensive ($O(n^3)$ in worst case). For large graphs, use sparse methods or approximate eigen-solvers.

Choosing Number of Clusters and Practical Notes

- **How many clusters (k)?** One heuristic: look for an "eigengap" - a large gap in the magnitude of eigenvalues λ_k vs λ_{k+1}. A big jump suggests k meaningful clusters.
- The eigenvectors can be sensitive to graph structure; noise or nearly-equal eigenvalues can cause instability. Using k-means on multiple eigenvectors tends to be more stable for $k > 2$.
- **Normalised vs unnormalised:** Normalised spectral clustering often performs better on imbalanced degree graphs, ensuring each cluster has fair volume. Unnormalized is simpler but may bias toward cutting off small degree nodes.
- **Complexity:** Computing eigenvectors can be expensive ($O(n^3)$ in worst case). For large graphs, use sparse methods or approximate eigen-solvers.

Example: 2-Cluster Spectral Partitioning (Karate Club)

Zachary's Karate Club social network (34 nodes, 78 edges). A known split occurred (two factions after a conflict).

Graph:

- Zachary's Karate Club network. Colors indicate the actual split of the club. This graph will be partitioned via spectral clustering.
- Compute the Fiedler vector ($\mathbf{v}_2$ of L). Then cluster by its sign (unnormalized spectral bi-partition).

Result: Spectral clustering perfectly splits the two factions (except possibly one node) - the Fiedler vector's sign corresponds closely to the true division (correlation 0.86 between $\mathbf{v}_2$ and ground-truth split).

Example: 2-Cluster Spectral Partitioning (Karate Club)

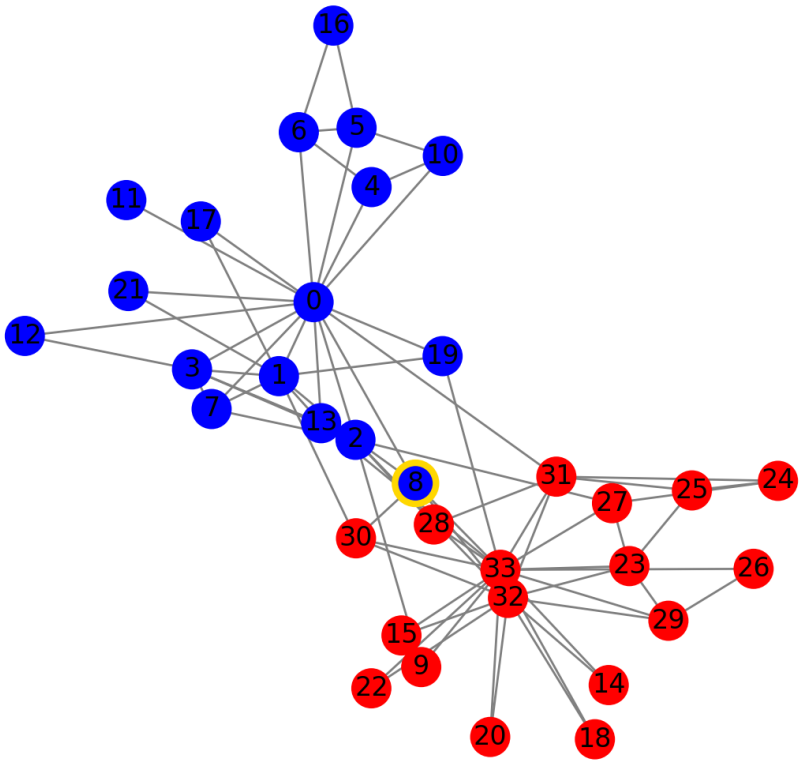
Zachary's Karate Club social network (34 nodes, 78 edges). A known split occurred (two factions after a conflict).

Graph:

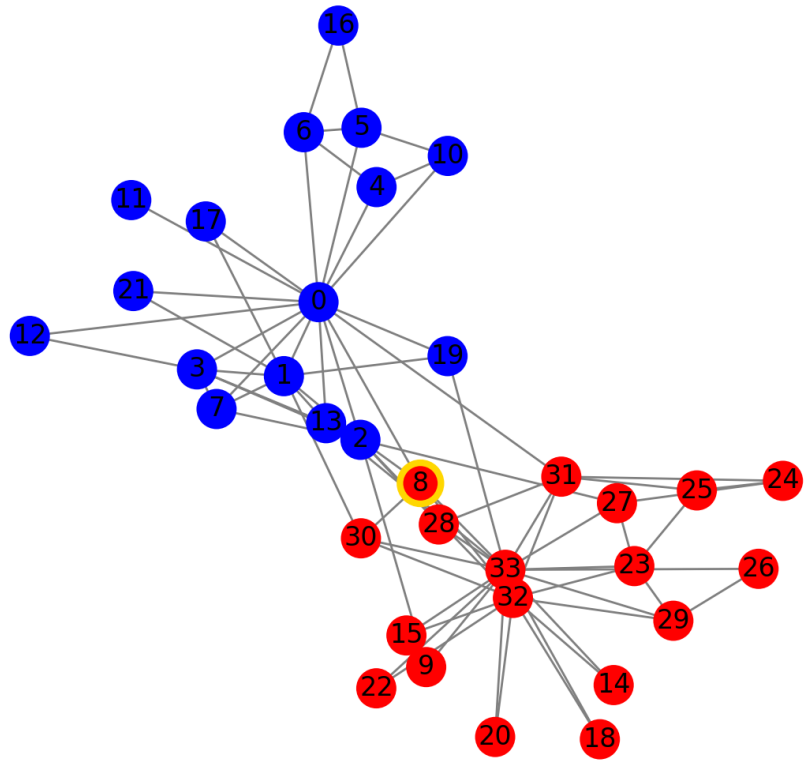
- Zachary's Karate Club network. Colors indicate the actual split of the club. This graph will be partitioned via spectral clustering.
- Compute the Fiedler vector ($\mathbf{v}_2$ of L). Then cluster by its sign (unnormalized spectral bi-partition).

Result: Spectral clustering perfectly splits the two factions (except possibly one node) - the Fiedler vector's sign corresponds closely to the true division (correlation 0.86 between $\mathbf{v}_2$ and ground-truth split).

Actual Factions (Zachary's Karate Club)



Spectral Bi-partition (Fiedler vector)



Spectral Embedding Visualisation (Karate Club)

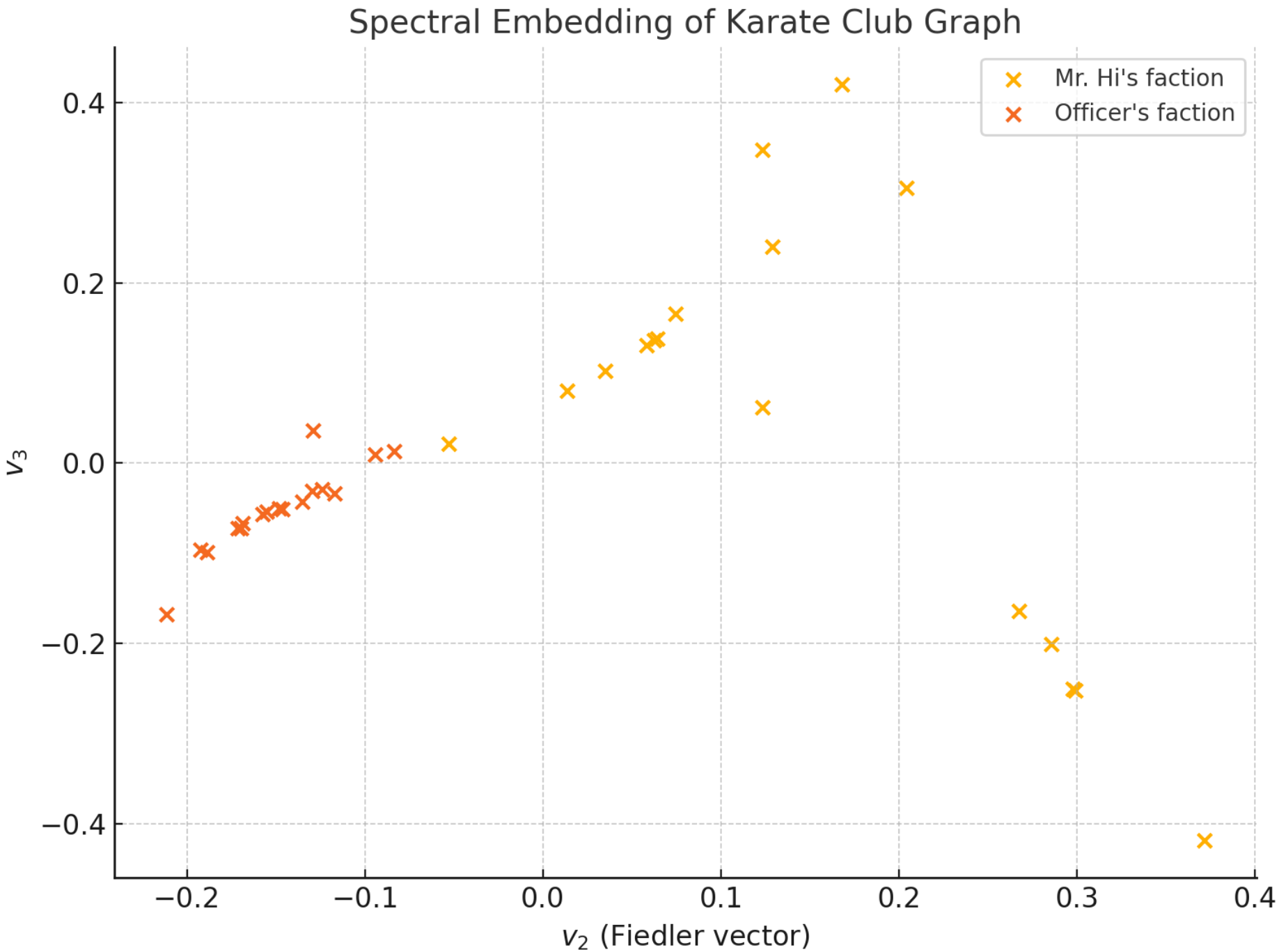
- We can embed each node in $\mathbb{R}^2$ using the first two nontrivial Laplacian eigenvectors $(\mathbf{v}_2, \mathbf{v}_3)$. Plotting these coordinates:
- Embedding of Karate Club nodes in 2D using $\mathbf{v}_2$ (horizontal) and $\mathbf{v}_3$ (vertical). Green "$\times$" = Mr. Hi's faction, Red "$\times$" = Officer's faction. The two clusters separate clearly along $\mathbf{v}_2$ (Fiedler axis). Some substructure in the green group is visible along $\mathbf{v}_3$. This spectral embedding clusters the nodes naturally.
- The second eigenvector (x-axis) clearly divides the two main clusters (red vs blue). The third eigenvector (y-axis) shows minor splits within one cluster (less significant). This illustrates how higher eigenvectors can capture finer structure beyond the first split.

Spectral Embedding Visualisation (Karate Club)

- We can embed each node in $\mathbb{R}^2$ using the first two nontrivial Laplacian eigenvectors $(\mathbf{v}_2, \mathbf{v}_3)$. Plotting these coordinates:
- Embedding of Karate Club nodes in 2D using $\mathbf{v}_2$ (horizontal) and $\mathbf{v}_3$ (vertical). Green "$\times$" = Mr. Hi's faction, Red "$\times$" = Officer's faction. The two clusters separate clearly along $\mathbf{v}_2$ (Fiedler axis). Some substructure in the green group is visible along $\mathbf{v}_3$. This spectral embedding clusters the nodes naturally.
- The second eigenvector (x-axis) clearly divides the two main clusters (red vs blue). The third eigenvector (y-axis) shows minor splits within one cluster (less significant). This illustrates how higher eigenvectors can capture finer structure beyond the first split.

Spectral Embedding Visualisation (Karate Club)

- We can embed each node in $\mathbb{R}^2$ using the first two nontrivial Laplacian eigenvectors $(\mathbf{v}_2, \mathbf{v}_3)$. Plotting these coordinates:
- Embedding of Karate Club nodes in 2D using $\mathbf{v}_2$ (horizontal) and $\mathbf{v}_3$ (vertical). Green "$\times$" = Mr. Hi's faction, Red "$\times$" = Officer's faction. The two clusters separate clearly along $\mathbf{v}_2$ (Fiedler axis). Some substructure in the green group is visible along $\mathbf{v}_3$. This spectral embedding clusters the nodes naturally.
- The second eigenvector (x-axis) clearly divides the two main clusters (red vs blue). The third eigenvector (y-axis) shows minor splits within one cluster (less significant). This illustrates how higher eigenvectors can capture finer structure beyond the first split.


Spectral Embedding of Karate Club Graph
Mr. Hi's faction
Officer's faction
0.4
0.2
0.0
−0.2
−0.4
v_3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
v_2 (Fiedler vector)

Laplacian Eigenmaps

Scenario:

- We have n data points (nodes), possibly lying on a low-dimensional manifold embedded in high-dimensional space.
- We know pairwise similarities (or build a nearest-neighbor graph).
- How to embed them in a lower-dimensional space while preserving local structure?

Laplacian Eigenmaps

Laplacian Eigenmaps (Belkin and Niyogi 2003):

- Construct a graph G of the data: nodes = data points, edges connect nearest neighbors (with weight w_{ij} reflecting similarity).
- **Goal:** find low-dim coordinates $\mathbf{y}_i \in \mathbb{R}^d$ for each node i such that if i and j are connected, then $\mathbf{y}_i$ and $\mathbf{y}_j$ are close. (Preserve local neighborhood distances.)
- Formulate as optimising quadratic form on the graph:

$$\text{minimise} \sum_{i,j} w_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$$

  subject to constraints preventing trivial solution (e.g. $\frac{1}{n}\sum_i \mathbf{y}_i = \mathbf{0}$ and $\frac{1}{n}\sum_i \mathbf{y}_i \mathbf{y}_i^\top = I_d$, or simpler: $Y^\top D Y = I$).
- This optimisation can be solved via the bottom $d+1$ eigenvectors of L (excluding the trivial all-ones direction). The solution $\mathbf{y}_i$ is given by these eigenvectors (just like clustering but using continuous embedding).

Laplacian Eigenmaps

Laplacian Eigenmaps (Belkin and Niyogi 2003):

- Construct a graph G of the data: nodes = data points, edges connect nearest neighbors (with weight w_{ij} reflecting similarity).
- **Goal:** find low-dim coordinates $\mathbf{y}_i \in \mathbb{R}^d$ for each node i such that if i and j are connected, then $\mathbf{y}_i$ and $\mathbf{y}_j$ are close. (Preserve local neighborhood distances.)
- Formulate as optimising quadratic form on the graph:

$$\text{minimise} \sum_{i,j} w_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$$

  subject to constraints preventing trivial solution (e.g. $\frac{1}{n}\sum_i \mathbf{y}_i = \mathbf{0}$ and $\frac{1}{n}\sum_i \mathbf{y}_i \mathbf{y}_i^\top = I_d$, or simpler: $Y^\top D Y = I$).
- This optimisation can be solved via the bottom $d+1$ eigenvectors of L (excluding the trivial all-ones direction). The solution $\mathbf{y}_i$ is given by these eigenvectors (just like clustering but using continuous embedding).

Laplacian Eigenmaps

Laplacian Eigenmaps (Belkin and Niyogi 2003):

- Construct a graph G of the data: nodes = data points, edges connect nearest neighbors (with weight w_{ij} reflecting similarity).
- **Goal:** find low-dim coordinates $\mathbf{y}_i \in \mathbb{R}^d$ for each node i such that if i and j are connected, then $\mathbf{y}_i$ and $\mathbf{y}_j$ are close. (Preserve local neighborhood distances.)
- Formulate as optimising quadratic form on the graph:

$$\text{minimise} \sum_{i,j} w_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$$

  subject to constraints preventing trivial solution (e.g. $\frac{1}{n}\sum_i \mathbf{y}_i = \mathbf{0}$ and $\frac{1}{n}\sum_i \mathbf{y}_i \mathbf{y}_i^\top = I_d$, or simpler: $Y^\top D Y = I$).
- This optimisation can be solved via the bottom $d+1$ eigenvectors of L (excluding the trivial all-ones direction). The solution $\mathbf{y}_i$ is given by these eigenvectors (just like clustering but using continuous embedding).

Laplacian Eigenmaps

Laplacian Eigenmaps (Belkin and Niyogi 2003):

- Construct a graph G of the data: nodes = data points, edges connect nearest neighbors (with weight w_{ij} reflecting similarity).
- **Goal:** find low-dim coordinates $\mathbf{y}_i \in \mathbb{R}^d$ for each node i such that if i and j are connected, then $\mathbf{y}_i$ and $\mathbf{y}_j$ are close. (Preserve local neighborhood distances.)
- Formulate as optimising quadratic form on the graph:

$$\text{minimise} \sum_{i,j} w_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$$

  subject to constraints preventing trivial solution (e.g. $\frac{1}{n}\sum_i \mathbf{y}_i = \mathbf{0}$ and $\frac{1}{n}\sum_i \mathbf{y}_i \mathbf{y}_i^\top = I_d$, or simpler: $Y^\top D Y = I$).
- This optimisation can be solved via the bottom $d+1$ eigenvectors of L (excluding the trivial all-ones direction). The solution $\mathbf{y}_i$ is given by these eigenvectors (just like clustering but using continuous embedding).

Laplacian Eigenmaps vs. PCA (Linear methods)

- **PCA (Principal Component Analysis)** finds a linear projection that best preserves variance (global structure) - it doesn't account for nonlinear manifolds. It treats distances between all points equally.
- **Laplacian Eigenmaps (LE)** focuses only on preserving local neighborhood relationships (it's a nonlinear method). Far apart points on the manifold can be projected far apart or even jumbled, as long as local structures remain.
- LE is one of several manifold learning techniques (others: Isomap, Locally Linear Embedding, Diffusion Maps). It is closely related to spectral clustering: instead of discrete clusters, we get continuous coordinates (which could also be used for clustering by further processing).

Laplacian Eigenmaps vs. PCA (Linear methods)

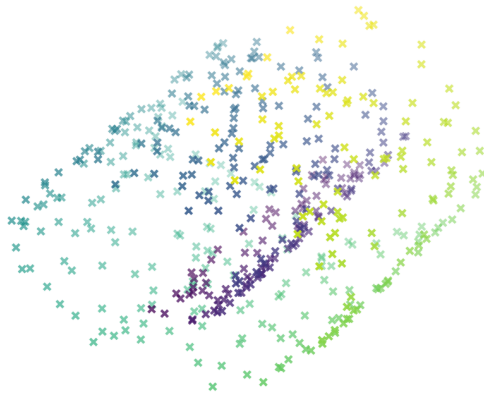
- **PCA (Principal Component Analysis)** finds a linear projection that best preserves variance (global structure) - it doesn't account for nonlinear manifolds. It treats distances between all points equally.
- **Laplacian Eigenmaps (LE)** focuses only on preserving local neighborhood relationships (it's a nonlinear method). Far apart points on the manifold can be projected far apart or even jumbled, as long as local structures remain.
- LE is one of several manifold learning techniques (others: Isomap, Locally Linear Embedding, Diffusion Maps). It is closely related to spectral clustering: instead of discrete clusters, we get continuous coordinates (which could also be used for clustering by further processing).

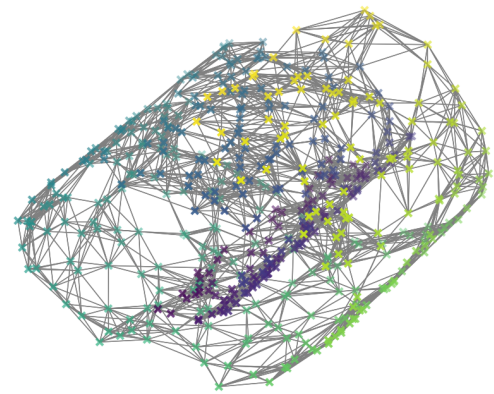
Laplacian Eigenmaps vs. PCA (Linear methods)

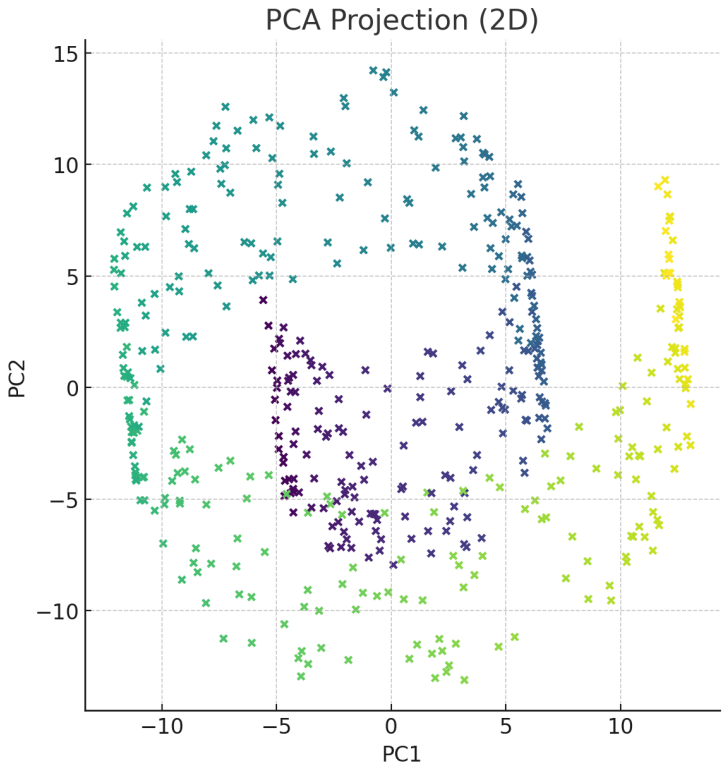
- **PCA (Principal Component Analysis)** finds a linear projection that best preserves variance (global structure) - it doesn't account for nonlinear manifolds. It treats distances between all points equally.
- **Laplacian Eigenmaps (LE)** focuses only on preserving local neighborhood relationships (it's a nonlinear method). Far apart points on the manifold can be projected far apart or even jumbled, as long as local structures remain.
- LE is one of several manifold learning techniques (others: Isomap, Locally Linear Embedding, Diffusion Maps). It is closely related to spectral clustering: instead of discrete clusters, we get continuous coordinates (which could also be used for clustering by further processing).

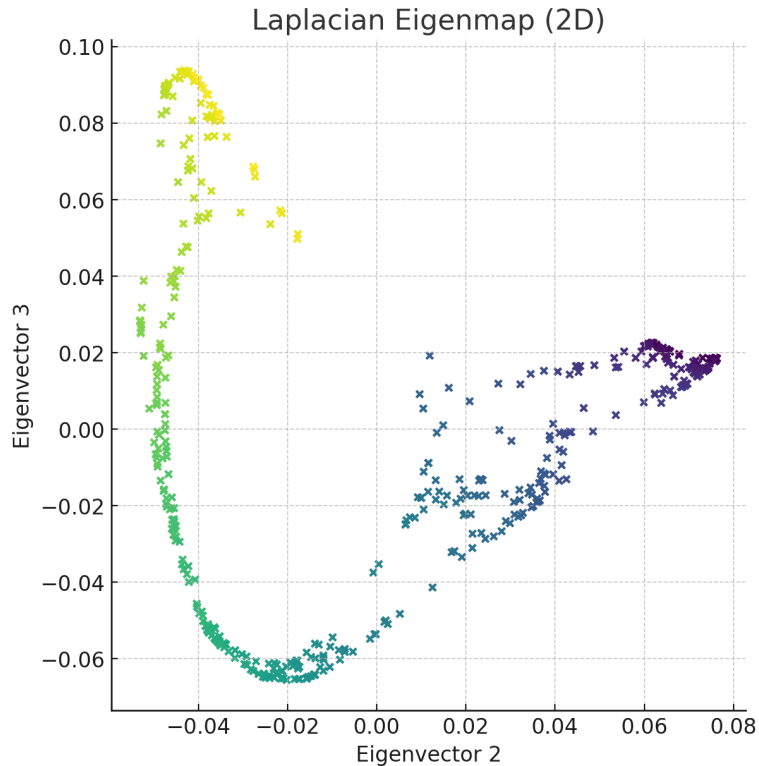
Laplacian Eigenmaps vs. PCA (Linear methods)

- **Example: Swiss roll manifold.** If data lies on a twisted 2D surface in 3D: PCA would fail to unroll it (since it's nonlinear). Laplacian Eigenmaps can "unroll" the manifold by using the graph of nearest neighbors - eigenvectors of L recover the underlying 2D parameterisation (up to distortion) (conceptually shown in figure).
- **Limitations:** Requires choosing a neighbourhood graph and weight scheme; sensitive to graph construction. Eigen-decomposition can be expensive for very large n.


Raw Swiss Roll Data


k-NN Graph on Data


PCA Projection (2D)
PC1
PC2
15
10
5
0
−5
−10
−10
−5
0
5
10


Laplacian Eigenmap (2D)
Eigenvector 2
Eigenvector 3
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08

Node Embedding Approaches

Graph Node Embeddings: Overview

Beyond clustering into communities, we often want to represent each node as a point in $\mathbb{R}^d$ (with $d \ll n$) for tasks like link prediction, visualisation, or as features for ML models.

Two broad families:

1. **Matrix Factorisation** (Spectral) methods: Define some matrix of node similarities (adjacency, Laplacian, or higher-order) and factorize it (via eigen-decomposition/SVD). E.g.:
   - Adjacency spectral embedding (ASE).
   - Laplacian eigenmaps (just covered).
   - Katz similarity embedding (HOPE algorithm).
   - Graph factorisation (explicit low-rank factorisation of adjacency).

Graph Node Embeddings: Overview

Beyond clustering into communities, we often want to represent each node as a point in $\mathbb{R}^d$ (with $d \ll n$) for tasks like link prediction, visualisation, or as features for ML models.

Two broad families:

1. **Matrix Factorisation** (Spectral) methods: Define some matrix of node similarities (adjacency, Laplacian, or higher-order) and factorize it (via eigen-decomposition/SVD). E.g.:
   - Adjacency spectral embedding (ASE).
   - Laplacian eigenmaps (just covered).
   - Katz similarity embedding (HOPE algorithm).
   - Graph factorisation (explicit low-rank factorisation of adjacency).

Graph Node Embeddings: Overview

2. **Random-walk based** (neural) methods: Generate random walks on the graph, treat them like "sentences" and use word embedding techniques (like Word2Vec) to learn node vectors. E.g.:
   - DeepWalk (Perozzi et al. 2014): uniform random walks + Skip-gram model.
   - node2vec (Grover and Leskovec 2016): biased random walks (with BFS/DFS flavor) + Skip-gram.
   - Others: LINE (first/second order proximity), Struc2vec, etc.

Key insight: Many of these methods are connected — random walk methods often implicitly factorise a matrix capturing node co-occurrences, meaning they have an underlying spectral interpretation.

Graph Node Embeddings: Overview

2. **Random-walk based** (neural) methods: Generate random walks on the graph, treat them like "sentences" and use word embedding techniques (like Word2Vec) to learn node vectors. E.g.:
   - DeepWalk (Perozzi et al. 2014): uniform random walks + Skip-gram model.
   - node2vec (Grover and Leskovec 2016): biased random walks (with BFS/DFS flavor) + Skip-gram.
   - Others: LINE (first/second order proximity), Struc2vec, etc.

Key insight: Many of these methods are connected — random walk methods often implicitly factorise a matrix capturing node co-occurrences, meaning they have an underlying spectral interpretation.

Matrix Factorisation: Adjacency and Beyond

- **Adjacency Spectral Embedding (ASE):** Use the top d eigenvectors of A (or singular vectors if graph is not symmetric) as embedding. For an undirected graph, $A = X\Sigma X^\top$ (spectral decomposition), take $X_d\Sigma_d^{1/2}$ as $n \times d$ embedding (this is akin to PCA on A).
- **Interpretation:** This gives the best rank-d approximation $A \approx \hat{A} = X_d\Sigma_d X_d^\top$. If graph has d well-defined communities (like a Stochastic Block Model), this can recover community structure (each eigenvector may correspond to a cluster).
- However, A's leading eigenvectors often pick up high-degree nodes or global structures (not necessarily best for clustering if degree distribution is skewed).

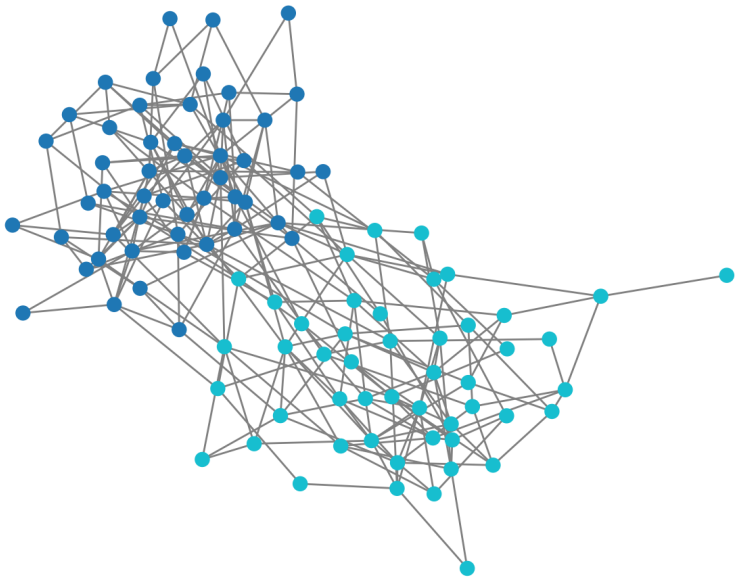
Matrix Factorisation: Adjacency and Beyond

- **Adjacency Spectral Embedding (ASE):** Use the top d eigenvectors of A (or singular vectors if graph is not symmetric) as embedding. For an undirected graph, $A = X\Sigma X^\top$ (spectral decomposition), take $X_d\Sigma_d^{1/2}$ as $n \times d$ embedding (this is akin to PCA on A).
- **Interpretation:** This gives the best rank-d approximation $A \approx \hat{A} = X_d\Sigma_d X_d^\top$. If graph has d well-defined communities (like a Stochastic Block Model), this can recover community structure (each eigenvector may correspond to a cluster).
- However, A's leading eigenvectors often pick up high-degree nodes or global structures (not necessarily best for clustering if degree distribution is skewed).

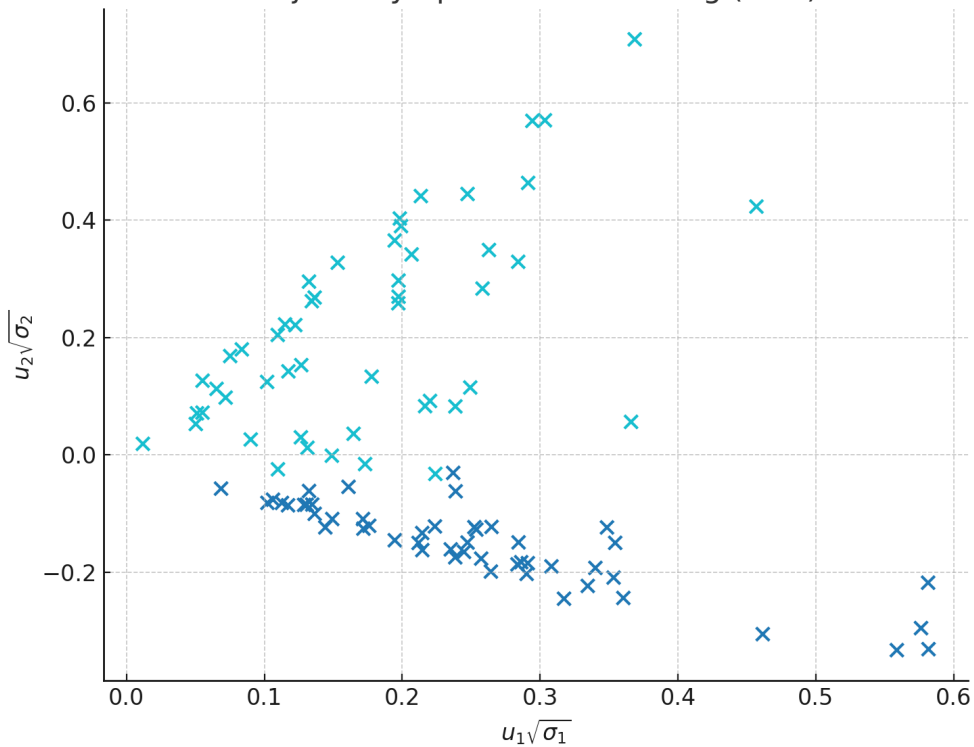
Matrix Factorisation: Adjacency and Beyond

- **Adjacency Spectral Embedding (ASE):** Use the top d eigenvectors of A (or singular vectors if graph is not symmetric) as embedding. For an undirected graph, $A = X\Sigma X^\top$ (spectral decomposition), take $X_d\Sigma_d^{1/2}$ as $n \times d$ embedding (this is akin to PCA on A).
- **Interpretation:** This gives the best rank-d approximation $A \approx \hat{A} = X_d\Sigma_d X_d^\top$. If graph has d well-defined communities (like a Stochastic Block Model), this can recover community structure (each eigenvector may correspond to a cluster).
- However, A's leading eigenvectors often pick up high-degree nodes or global structures (not necessarily best for clustering if degree distribution is skewed).

Original Graph (SBM Communities)



Adjacency Spectral Embedding (d=2)



Random-Walk Embeddings: DeepWalk

DeepWalk (Perozzi et al. 2014):

- For each node u, simulate many random walks of fixed length (e.g. 40). A walk is a sequence $u = v_0, v_1, \ldots, v_t$.
- Treat each random walk as a "sentence" of nodes. For a node v_i in the walk, consider nodes within a window (e.g. ± 5 steps) as its context (neighbors in the sentence).
- Use **Skip-gram with Negative Sampling** (SGNS) to learn embeddings: maximize probability of observing context nodes given the embedding of center node. This is exactly the Word2Vec algorithm applied to node sequences.
- **Result:** each node has a vector $\mathbf{h}_u \in \mathbb{R}^d$. Nodes that tend to co-occur on random walks get similar embeddings. Typically, this captures communities (since random walks stay within clusters with higher probability).

Random-Walk Embeddings: DeepWalk

DeepWalk (Perozzi et al. 2014):

- For each node u, simulate many random walks of fixed length (e.g. 40). A walk is a sequence $u = v_0, v_1, \ldots, v_t$.
- Treat each random walk as a "sentence" of nodes. For a node v_i in the walk, consider nodes within a window (e.g. ± 5 steps) as its context (neighbors in the sentence).
- Use **Skip-gram with Negative Sampling** (SGNS) to learn embeddings: maximize probability of observing context nodes given the embedding of center node. This is exactly the Word2Vec algorithm applied to node sequences.
- **Result:** each node has a vector $\mathbf{h}_u \in \mathbb{R}^d$. Nodes that tend to co-occur on random walks get similar embeddings. Typically, this captures communities (since random walks stay within clusters with higher probability).

Random-Walk Embeddings: DeepWalk

DeepWalk (Perozzi et al. 2014):

- For each node u, simulate many random walks of fixed length (e.g. 40). A walk is a sequence $u = v_0, v_1, \ldots, v_t$.
- Treat each random walk as a "sentence" of nodes. For a node v_i in the walk, consider nodes within a window (e.g. ± 5 steps) as its context (neighbors in the sentence).
- Use **Skip-gram with Negative Sampling** (SGNS) to learn embeddings: maximize probability of observing context nodes given the embedding of center node. This is exactly the Word2Vec algorithm applied to node sequences.
- **Result:** each node has a vector $\mathbf{h}_u \in \mathbb{R}^d$. Nodes that tend to co-occur on random walks get similar embeddings. Typically, this captures communities (since random walks stay within clusters with higher probability).

Random-Walk Embeddings: DeepWalk

DeepWalk (Perozzi et al. 2014):

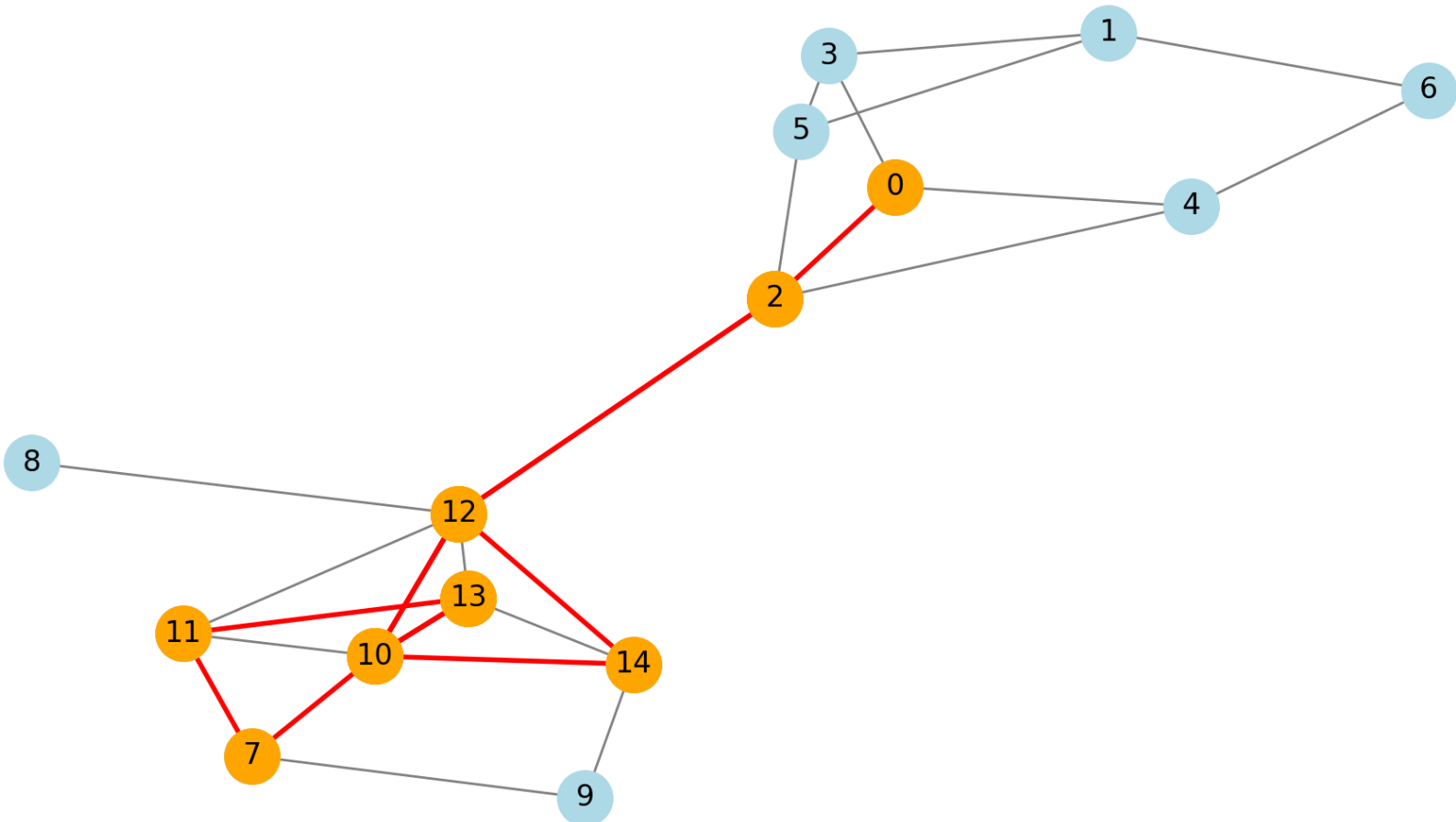
- For each node u, simulate many random walks of fixed length (e.g. 40). A walk is a sequence $u = v_0, v_1, \ldots, v_t$.
- Treat each random walk as a "sentence" of nodes. For a node v_i in the walk, consider nodes within a window (e.g. ± 5 steps) as its context (neighbors in the sentence).
- Use **Skip-gram with Negative Sampling** (SGNS) to learn embeddings: maximize probability of observing context nodes given the embedding of center node. This is exactly the Word2Vec algorithm applied to node sequences.
- **Result:** each node has a vector $\mathbf{h}_u \in \mathbb{R}^d$. Nodes that tend to co-occur on random walks get similar embeddings. Typically, this captures communities (since random walks stay within clusters with higher probability).

Random-Walk Embeddings: DeepWalk

DeepWalk (Perozzi et al. 2014):

- For each node u, simulate many random walks of fixed length (e.g. 40). A walk is a sequence $u = v_0, v_1, \ldots, v_t$.
- Treat each random walk as a "sentence" of nodes. For a node v_i in the walk, consider nodes within a window (e.g. ± 5 steps) as its context (neighbors in the sentence).
- Use **Skip-gram with Negative Sampling** (SGNS) to learn embeddings: maximize probability of observing context nodes given the embedding of center node. This is exactly the Word2Vec algorithm applied to node sequences.
- **Result:** each node has a vector $\mathbf{h}_u \in \mathbb{R}^d$. Nodes that tend to co-occur on random walks get similar embeddings. Typically, this captures communities (since random walks stay within clusters with higher probability).

Graph with Random Walk Highlighted



Random Walk as 'Sentence' with Context Window

0	2	12	10	7	11	13	10	12	14	10	13	10	12	2

Random-Walk Embeddings: node2vec and others

node2vec (Grover and Leskovec 2016):

- Extends DeepWalk by introducing parameters (p, q) to bias the random walk. After walking from t to v, the next step is chosen among neighbors of v with probabilities:
  - if going back to t (the previous node) $\propto 1/p$ (discouraged if $p > 1$),
  - if going to a neighbor of v that is not t: $\propto 1$ if that neighbor is "close" to t (distance 1), $\propto 1/q$ if it is farther (i.e., exploring outward).
- Effect: $q > 1$ favors BFS (stay close to t, good for homophily communities), $q < 1$ favors DFS (venture far, capturing structural equivalence).
- Use the same skip-gram training on these biased walks. node2vec can interpolate between embedding for community detection vs role discovery.

Unifying View: Embeddings as Low-Rank Approximation

A lot of graph embedding techniques can be viewed as finding a low-rank approximation to some matrix that encodes similarity between nodes. For example:

- **Spectral clustering:** low-rank approximation of Laplacian (use eigenvectors of L).
- **Laplacian Eigenmaps:** same as above (just using continuous embedding instead of clustering).
- **Adjacency SVD:** low-rank approximation of A.
- **DeepWalk/node2vec:** low-rank factorisation of PMI matrix built from $D^{-1}A$ powers.

Unifying View: Embeddings as Low-Rank Approximation

Thus, *"spectral methods"* broadly underpin these algorithms: in many cases, the optimal embedding could be obtained by an eigen-decomposition. The difference is often scalability and flexibility:

- DeepWalk/node2vec use SGD to avoid computing large matrices explicitly, but implicitly they are doing an eigen-like factorisation.
- Spectral clustering gives theoretical guarantees (e.g., eigen-gap and Cheeger bounds), while deep embeddings often give empirical improvements (and can incorporate nonlinearity or additional info).

Bottom line: Spectral methods provide the foundation, and modern embeddings refine them for large graphs and specific tasks.

Unifying View: Embeddings as Low-Rank Approximation

Thus, *"spectral methods"* broadly underpin these algorithms: in many cases, the optimal embedding could be obtained by an eigen-decomposition. The difference is often scalability and flexibility:

- DeepWalk/node2vec use SGD to avoid computing large matrices explicitly, but implicitly they are doing an eigen-like factorisation.
- Spectral clustering gives theoretical guarantees (e.g., eigen-gap and Cheeger bounds), while deep embeddings often give empirical improvements (and can incorporate nonlinearity or additional info).

Bottom line: Spectral methods provide the foundation, and modern embeddings refine them for large graphs and specific tasks.

Unifying View: Embeddings as Low-Rank Approximation

Thus, *"spectral methods"* broadly underpin these algorithms: in many cases, the optimal embedding could be obtained by an eigen-decomposition. The difference is often scalability and flexibility:

- DeepWalk/node2vec use SGD to avoid computing large matrices explicitly, but implicitly they are doing an eigen-like factorisation.
- Spectral clustering gives theoretical guarantees (e.g., eigen-gap and Cheeger bounds), while deep embeddings often give empirical improvements (and can incorporate nonlinearity or additional info).

Bottom line: Spectral methods provide the foundation, and modern embeddings refine them for large graphs and specific tasks.

Unifying View: Embeddings as Low-Rank Approximation

Thus, *"spectral methods"* broadly underpin these algorithms: in many cases, the optimal embedding could be obtained by an eigen-decomposition. The difference is often scalability and flexibility:

- DeepWalk/node2vec use SGD to avoid computing large matrices explicitly, but implicitly they are doing an eigen-like factorisation.
- Spectral clustering gives theoretical guarantees (e.g., eigen-gap and Cheeger bounds), while deep embeddings often give empirical improvements (and can incorporate nonlinearity or additional info).

Bottom line: Spectral methods provide the foundation, and modern embeddings refine them for large graphs and specific tasks.

Questions?