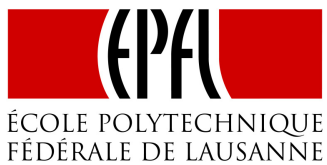
Graph signal processing for clustering

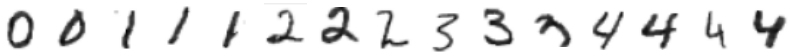
Nicolas Tremblay

PANAMA Team, INRIA Rennes with Rémi Gribonval,
Signal Processing Laboratory 2, EPFL, Lausanne with Pierre Vandergheynst.

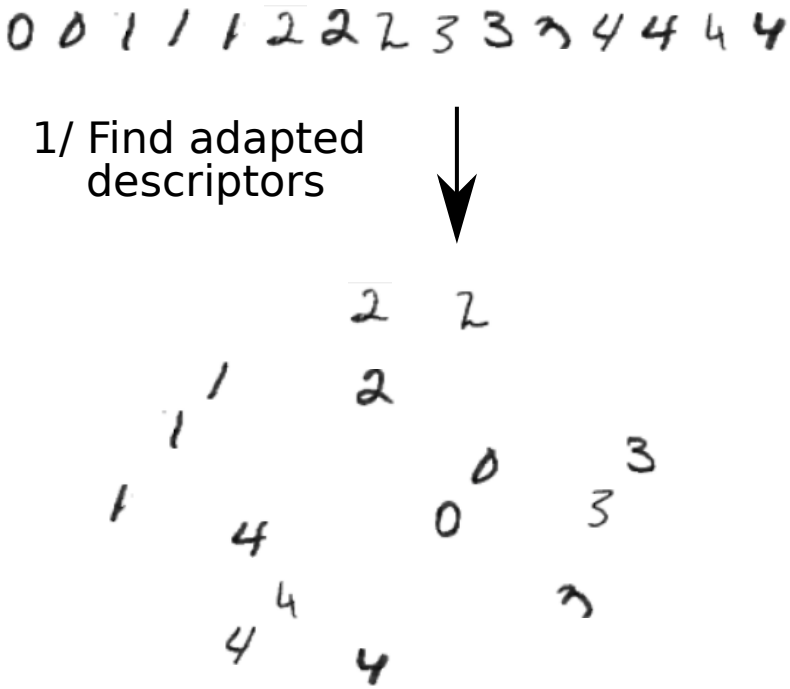


What's clustering ?

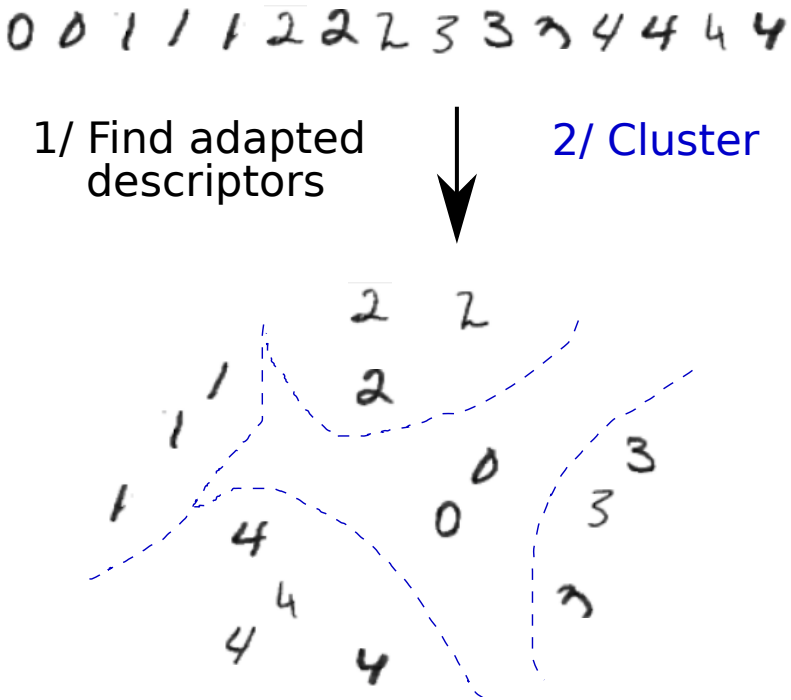
Given a series of N objects :



Given a series of N objects :



Given a series of N objects :



After step 1, one has :

- N vectors in d dimensions (descriptor dimension) :

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \in \mathbb{R}^d$$

After step 1, one has :

- N vectors in d dimensions (descriptor dimension) :

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \in \mathbb{R}^d$$

- and their *distance matrix* $\boldsymbol{D} \in \mathbb{R}^{N \times N}$.

After step 1, one has :

- N vectors in d dimensions (descriptor dimension) :

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \in \mathbb{R}^d$$

- and their *distance matrix* $\boldsymbol{D} \in \mathbb{R}^{N \times N}$.

The *goal* of clustering is to assign a label $c(i) = 1, \cdots, k$ to each object i in order to organize / simplify / analyze the data.

After step 1, one has :

- N vectors in d dimensions (descriptor dimension) :

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \in \mathbb{R}^d$$

- and their *distance matrix* $\boldsymbol{D} \in \mathbb{R}^{N \times N}$.

The *goal* of clustering is to assign a label $c(i) = 1, \cdots, k$ to each object i in order to organize / simplify / analyze the data.

There exists two different general types of methods :

- methods directly based on the $\boldsymbol{x}_i$ and/or $\boldsymbol{D}$ like k-means or hierarchical clustering.

After step 1, one has :

- N vectors in d dimensions (descriptor dimension) :

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \in \mathbb{R}^d$$

- and their *distance matrix* $\boldsymbol{D} \in \mathbb{R}^{N \times N}$.

The *goal* of clustering is to assign a label $c(i) = 1, \cdots, k$ to each object i in order to organize / simplify / analyze the data.

There exists two different general types of methods :

- methods directly based on the $\boldsymbol{x}_i$ and/or $\boldsymbol{D}$ like k-means or hierarchical clustering.
- graph-based methods.

Graph construction from the distance matrix D

Create a graph $\mathcal{G} = (V, E)$:

Graph construction from the distance matrix D

Create a graph $\mathcal{G} = (V, E)$:

- each node in V is one of the N objects

Graph construction from the distance matrix D

Create a graph $\mathcal{G} = (V, E)$:

- each node in V is one of the N objects
- each pair of nodes (i, j) is connected if the associated distance $D(i, j)$ is small enough.

Graph construction from the distance matrix D

Create a graph $\mathcal{G} = (V, E)$:

- each node in V is one of the N objects
- each pair of nodes (i, j) is connected if the associated distance $D(i, j)$ is small enough.

For example, two connectivity possibilities :

- Gaussian kernel :
  1. all pairs of nodes are connected with links of weights $\exp(-D(i, j)/\sigma)$
  2. remove all links of weight inferior to ϵ

Graph construction from the distance matrix D

Create a graph $\mathcal{G} = (V, E)$:

- each node in V is one of the N objects
- each pair of nodes (i, j) is connected if the associated distance $D(i, j)$ is small enough.

For example, two connectivity possibilities :

- Gaussian kernel :
  1. all pairs of nodes are connected with links of weights $\exp(-D(i, j)/\sigma)$
  2. remove all links of weight inferior to ϵ
- k nearest neighbors : connect each node to its k nearest neighbors.

The problem now states :

Given the graph $\mathcal{G}$ representing the similarity between the N objects, find a partition of all nodes in k clusters.

The problem now states :

Given the graph $\mathcal{G}$ representing the similarity between the N objects, find a partition of all nodes in k clusters.

Many methods exist [Fortunato '10] :

- Modularity (or other cost-function) optimisation methods [Newman '04]
- Random walk methods [Delvenne '10]
- Methods inspired from statistical physics [Krzakala '12], information theory [Rosvall '07]...
- spectral methods
- ...

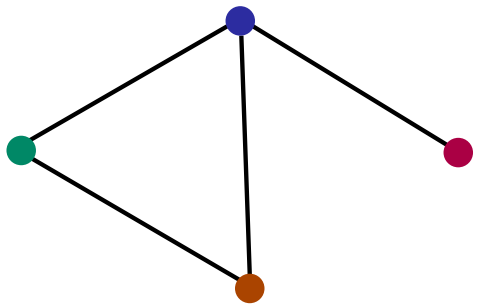
Three useful matrices

The adjacency matrix :

$$W = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The degree matrix :

$$S = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



The Laplacian matrix :

$$L = S - W = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

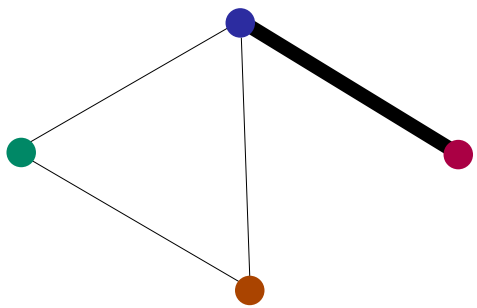
Three useful matrices

The adjacency matrix :

$$W = \begin{bmatrix} 0 & \mathbf{.5} & \mathbf{.5} & 0 \\ \mathbf{.5} & 0 & \mathbf{.5} & \mathbf{4} \\ \mathbf{.5} & \mathbf{.5} & 0 & 0 \\ 0 & \mathbf{4} & 0 & 0 \end{bmatrix}$$

The degree matrix :

$$S = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{5} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{4} \end{bmatrix}$$



The Laplacian matrix :

$$L = S - W = \begin{bmatrix} \mathbf{1} & \mathbf{-.5} & \mathbf{-.5} & 0 \\ \mathbf{-.5} & \mathbf{5} & \mathbf{-.5} & \mathbf{-4} \\ \mathbf{-.5} & \mathbf{-.5} & \mathbf{1} & 0 \\ 0 & \mathbf{-4} & 0 & \mathbf{4} \end{bmatrix}$$

The classical spectral clustering algorithm [Von Luxburg '06] :

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Compute :

$$\mathrm{U}_k = (\boldsymbol{u}_1 | \boldsymbol{u}_2 | \cdots | \boldsymbol{u}_k)$$

the first k eigenvectors of $\mathrm{L} = \mathrm{S} - \mathrm{W}$.

The classical spectral clustering algorithm [Von Luxburg '06] :

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Compute :
$$\mathrm{U}_k = (\boldsymbol{u}_1|\boldsymbol{u}_2|\cdots|\boldsymbol{u}_k)$$
the first k eigenvectors of $\mathrm{L} = \mathrm{S} - \mathrm{W}$.
2. Consider each node i as a point in $\mathbb{R}^k$:
$$\boldsymbol{f}_i = \mathrm{U}_k^\top \boldsymbol{\delta}_i.$$

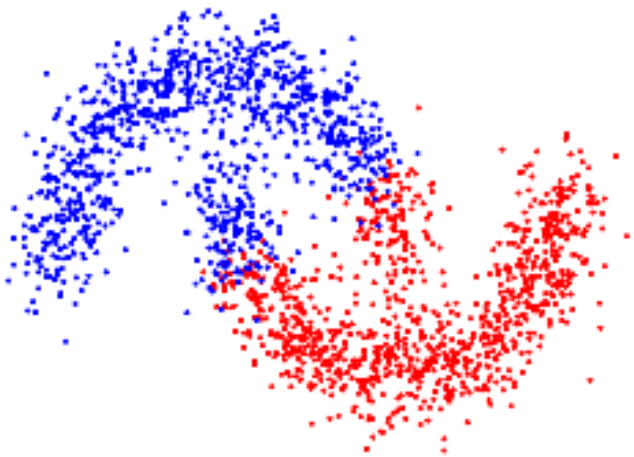
The classical spectral clustering algorithm [Von Luxburg '06] :

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

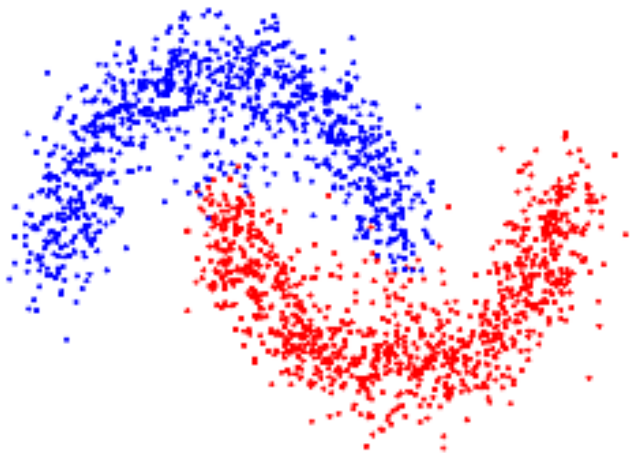
1. Compute :
$$\mathrm{U}_k = (\boldsymbol{u}_1|\boldsymbol{u}_2|\cdots|\boldsymbol{u}_k)$$
the first k eigenvectors of $\mathrm{L} = \mathrm{S} - \mathrm{W}$.
2. Consider each node i as a point in $\mathbb{R}^k$:
$$\boldsymbol{f}_i = \mathrm{U}_k^\top \boldsymbol{\delta}_i.$$
3. Run k-means with the Euclidean distance : $D_{ij} = ||\boldsymbol{f}_i - \boldsymbol{f}_j||$ and obtain k clusters.

What's the point of using a graph ?

N points in $d = 2$ dimensions.
Result with k-means (k=2) :



After creating a graph from the N points' interdistances, and running the spectral clustering algorithm (with k=2) :



Computation bottlenecks of the spectral clustering algorithm

When N and/or k become too large, there are two main bottlenecks in the algorithm :

1. The partial eigendecomposition of the Laplacian.

Computation bottlenecks of the spectral clustering algorithm

When N and/or k become too large, there are two main bottlenecks in the algorithm :

1. The partial eigendecomposition of the Laplacian.
2. k-means.

Computation bottlenecks of the spectral clustering algorithm

When N and/or k become too large, there are two main bottlenecks in the algorithm :

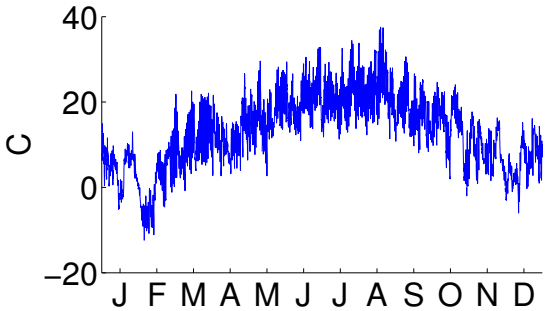
1. The partial eigendecomposition of the Laplacian.
2. k-means.

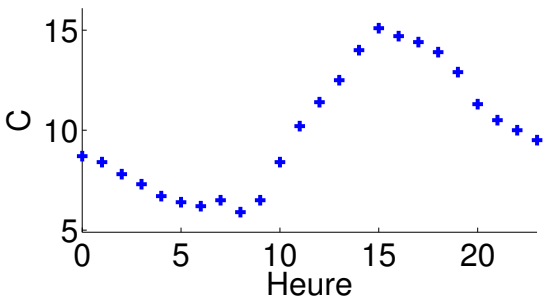
Our goal :

Circumvent both !

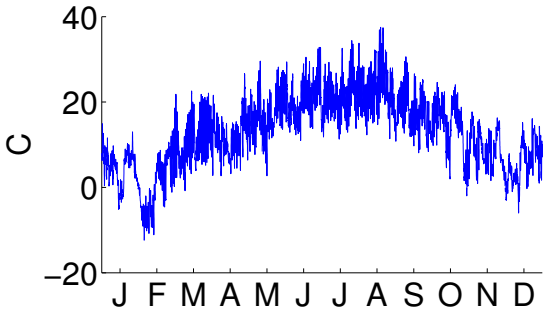
What's graph signal processing ?

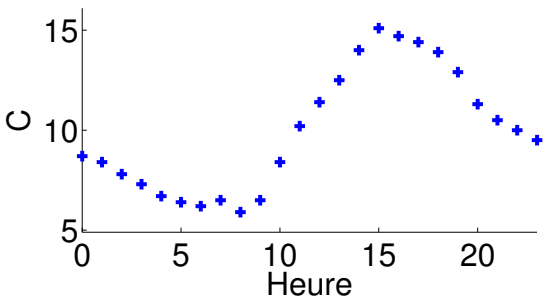
What's a graph signal ?

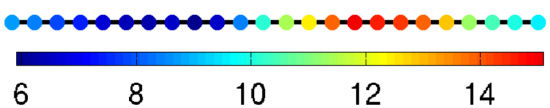




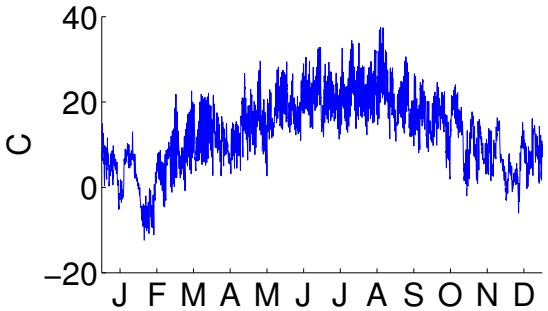
What's a graph signal ?

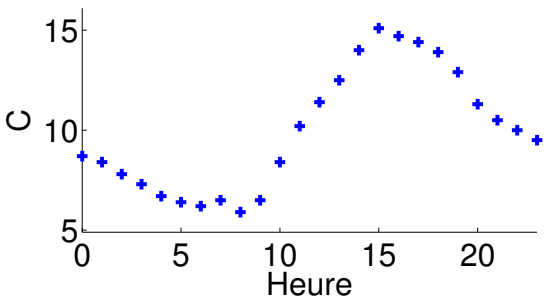


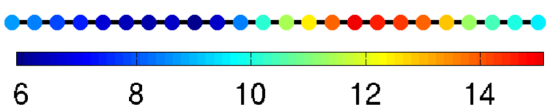




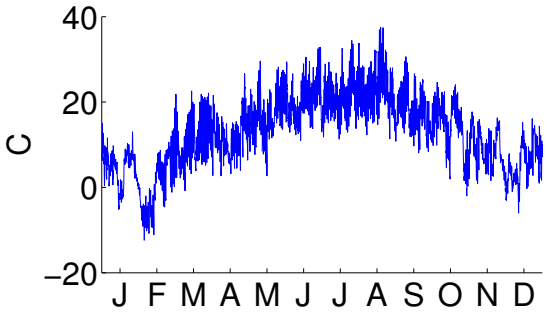
What's a graph signal ?

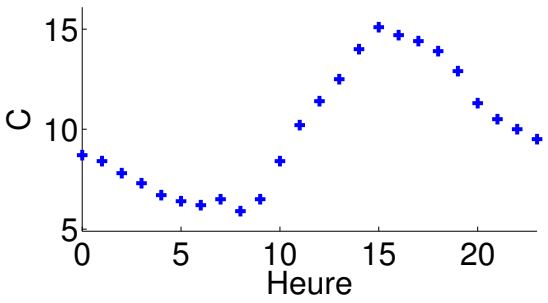


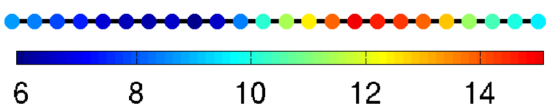


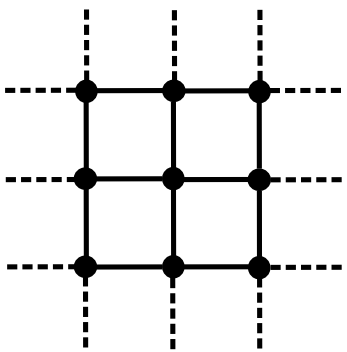


What's a graph signal ?

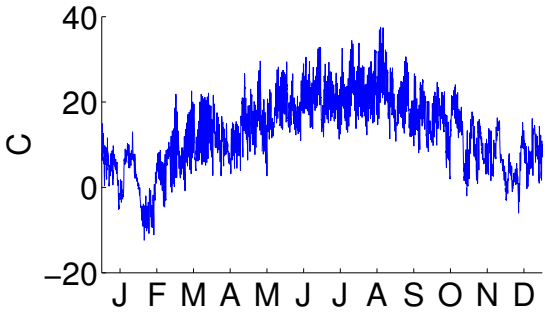


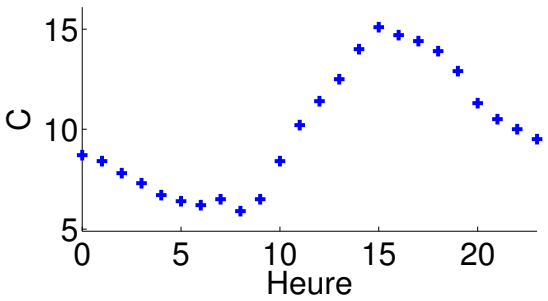


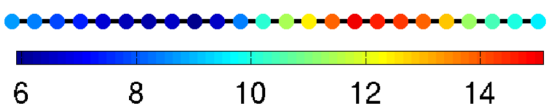


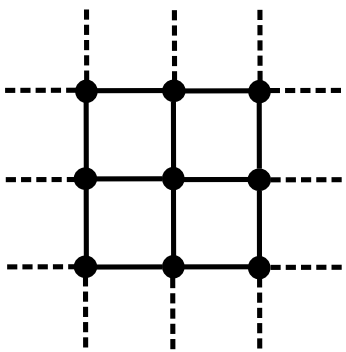


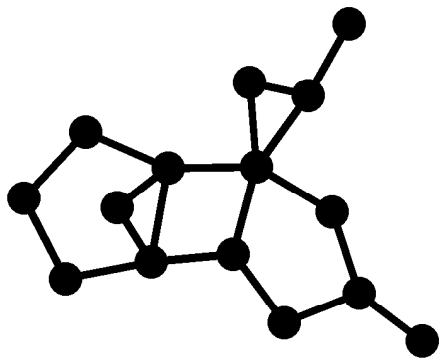
What's a graph signal ?



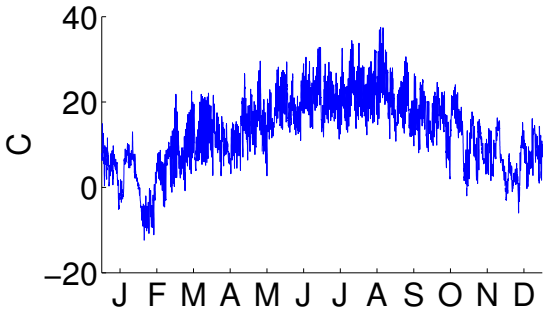


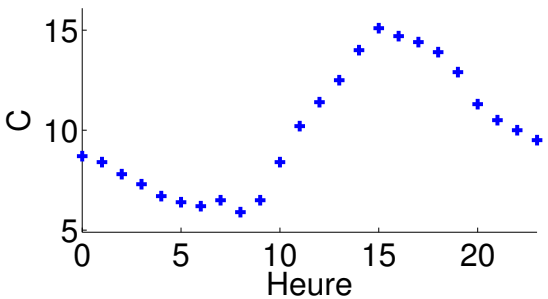


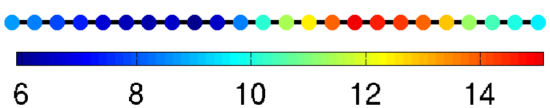


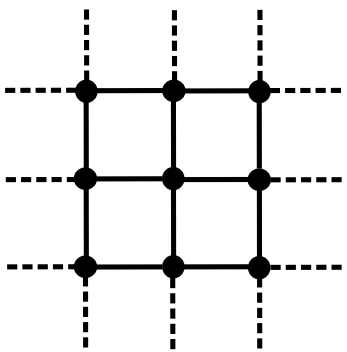


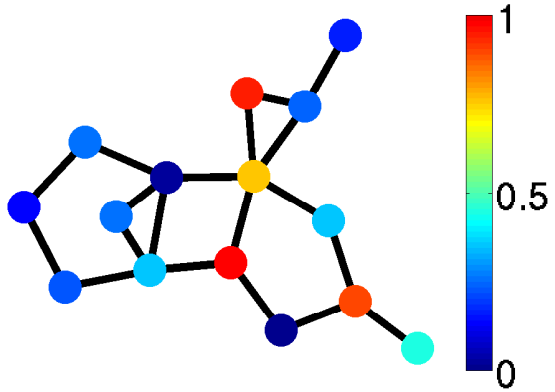
What's a graph signal ?

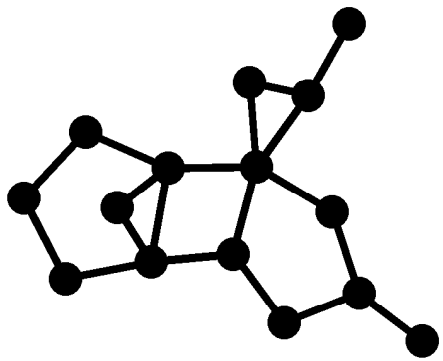








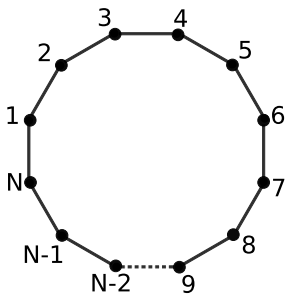




What's the graph Fourier matrix ?

[Hammond '11]

THE "CLASSICAL" GRAPH :



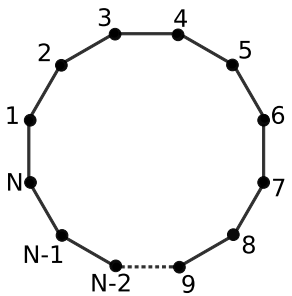
$$L_{cl} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ -1 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix}$$

All classical Fourier modes are the eigenvectors of L_{cl}

What's the graph Fourier matrix ?

[Hammond '11]

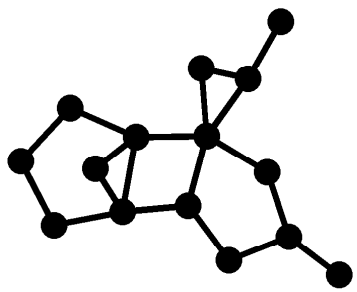
The "classical" graph :



$$L_{cl} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ -1 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix}$$

All classical Fourier modes are the eigenvectors of L_{cl}

Any graph :



L

By analogy, any graph's Fourier modes are the eigenvectors of its Laplacian matrix L.

The graph Fourier matrix

$$\boxed{L = S - W}$$

Its eigenvectors :

$$U = (\boldsymbol{u}_1 | \boldsymbol{u}_2 | \cdots | \boldsymbol{u}_N)$$

form the graph Fourier orthonormal basis.

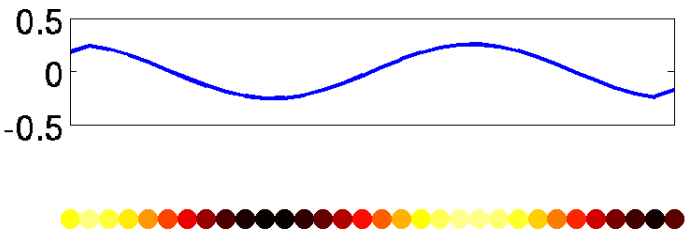
Its eigenvalues :

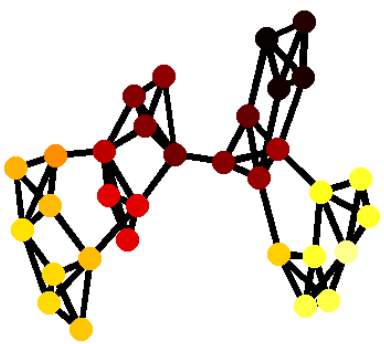
$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$$

represent the graph frequencies. λ_i is the squared frequency associated to the Fourier mode $\boldsymbol{u}_i$.

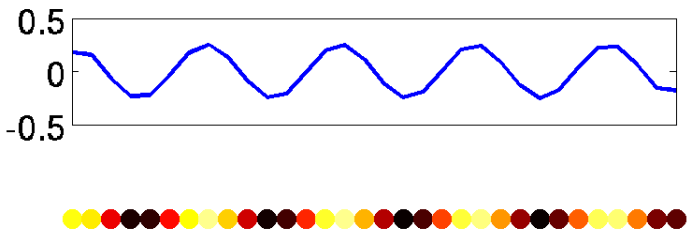
Illustration

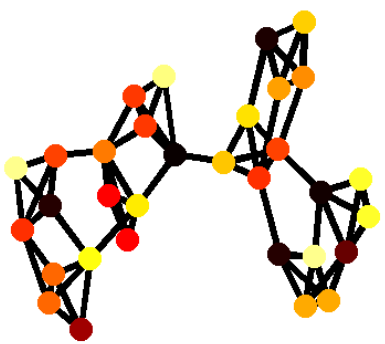
Low frequency :





High frequency :





The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size N.

The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size N.
- $\hat{f}$ is obtained by decomposing f on the eigenvectors $\boldsymbol{u}_i$:

$$\hat{f} = \begin{pmatrix} < \boldsymbol{u}_1, f > \\ < \boldsymbol{u}_2, f > \\ < \boldsymbol{u}_3, f > \\ \dots \\ < \boldsymbol{u}_N, f > \end{pmatrix}, \text{ i.e. } \boxed{\hat{f} = \mathrm{U}^\top f}$$

The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size N.
- $\hat{f}$ is obtained by decomposing f on the eigenvectors $\boldsymbol{u}_i$:

$$\hat{f} = \begin{pmatrix} < \boldsymbol{u}_1, f > \\ < \boldsymbol{u}_2, f > \\ < \boldsymbol{u}_3, f > \\ \dots \\ < \boldsymbol{u}_N, f > \end{pmatrix}, \text{ i.e. } \boxed{\hat{f} = \mathrm{U}^\top f}$$

- Inversely, the inverse Fourier transform reads :

$$\boxed{f = \mathrm{U}\,\hat{f}}$$

The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size N.
- $\hat{f}$ is obtained by decomposing f on the eigenvectors $\boldsymbol{u}_i$:

$$\hat{f} = \begin{pmatrix} < \boldsymbol{u}_1, f > \\ < \boldsymbol{u}_2, f > \\ < \boldsymbol{u}_3, f > \\ \dots \\ < \boldsymbol{u}_N, f > \end{pmatrix}, \text{ i.e. } \boxed{\hat{f} = \mathrm{U}^\top f}$$

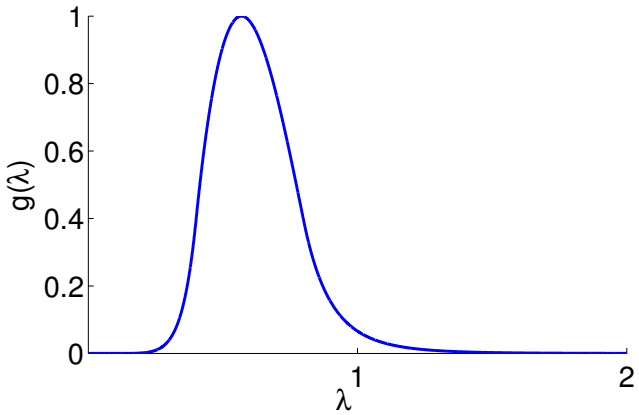
- Inversely, the inverse Fourier transform reads :

$$\boxed{f = \mathrm{U}\,\hat{f}}$$

- The Parseval theorem stays valid : $\forall (g, h) \; < g, h > = < \hat{g}, \hat{h} >$

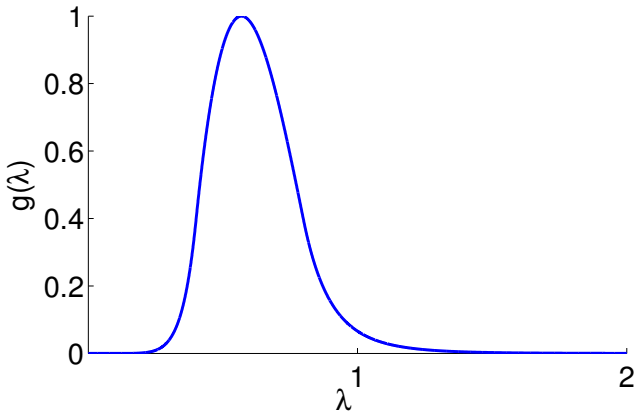
Filtering

Given a filter function g defined in the Fourier space.



Filtering

Given a filter function g defined in the Fourier space.

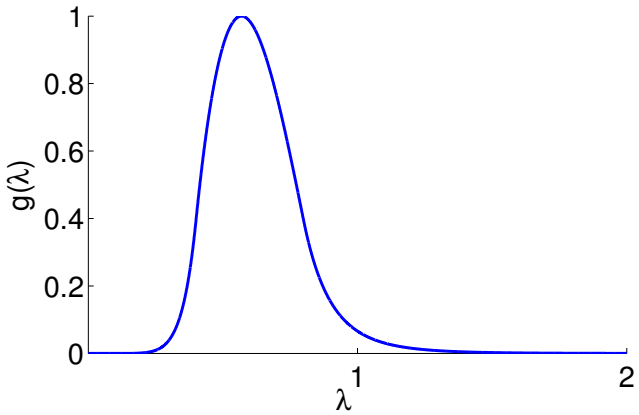


In the Fourier space, the signal filtered by g reads :

$$\hat{f}^g = \begin{pmatrix} \hat{f}(1)\, g(\lambda_1) \\ \hat{f}(2)\, g(\lambda_2) \\ \hat{f}(3)\, g(\lambda_3) \\ \dots \\ \hat{f}(N)\, g(\lambda_N) \end{pmatrix} = \hat{\mathrm{G}}\, \hat{f} \text{ with } \hat{\mathrm{G}} = \begin{pmatrix} g(\lambda_1) & 0 & 0 & \dots & 0 \\ 0 & g(\lambda_2) & 0 & \dots & 0 \\ 0 & 0 & g(\lambda_3) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & g(\lambda_N) \end{pmatrix}$$

Filtering

Given a filter function g defined in the Fourier space.



In the Fourier space, the signal filtered by g reads :

$$\hat{f}^g = \begin{pmatrix} \hat{f}(1)\, g(\lambda_1) \\ \hat{f}(2)\, g(\lambda_2) \\ \hat{f}(3)\, g(\lambda_3) \\ \dots \\ \hat{f}(N)\, g(\lambda_N) \end{pmatrix} = \hat{\mathrm{G}}\, \hat{f} \text{ with } \hat{\mathrm{G}} = \begin{pmatrix} g(\lambda_1) & 0 & 0 & \dots & 0 \\ 0 & g(\lambda_2) & 0 & \dots & 0 \\ 0 & 0 & g(\lambda_3) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & g(\lambda_N) \end{pmatrix}$$

In the node space, the filtered signal f^g reads therefore :

$$\boxed{f^g = \mathrm{U}\, \hat{\mathrm{G}}\, \mathrm{U}^\top f = \mathrm{G} f}$$

So where's the link ?

Remember : the classical spectral clustering algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Compute :
$$\mathrm{U}_k = (\boldsymbol{u}_1|\boldsymbol{u}_2|\cdots|\boldsymbol{u}_k)$$
the first k eigenvectors of $\mathrm{L} = \mathrm{S} - \mathrm{W}$.
2. Consider each node i as a point in $\mathbb{R}^k$:
$$\boldsymbol{f}_i = \mathrm{U}_k^\top \boldsymbol{\delta}_i.$$
3. Run k-means with the Euclidean distance : $D_{ij} = ||\boldsymbol{f}_i - \boldsymbol{f}_j||$ and obtain k clusters.

Remember : the classical spectral clustering algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

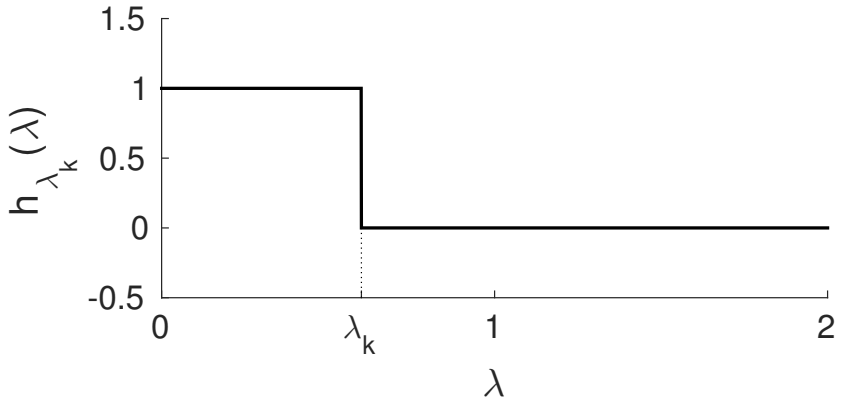
1. Compute :
$$\mathrm{U}_k = (\boldsymbol{u}_1|\boldsymbol{u}_2|\cdots|\boldsymbol{u}_k)$$
the first k eigenvectors of $\mathrm{L} = \mathrm{S} - \mathrm{W}$.
2. Consider each node i as a point in $\mathbb{R}^k$:
$$\boldsymbol{f}_i = \mathrm{U}_k^\top \boldsymbol{\delta}_i.$$
3. Run k-means with the Euclidean distance : $D_{ij} = ||\boldsymbol{f}_i - \boldsymbol{f}_j||$ and obtain k clusters.

Let's work on the first bottleneck : estimate D_{ij} without partially diagonalizing the Laplacian matrix.

Ideal low-pass filtering

1st step : assume we know U_k and λ_k

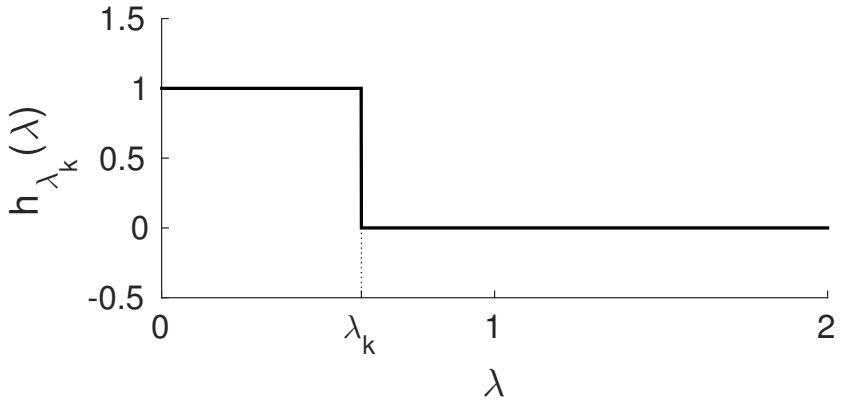
Given h_{λ_k} an ideal LP, $H_{\lambda_k} = U\hat{H}_{\lambda_k}U^\top = U_k U_k^\top$ is its filter matrix.



Ideal low-pass filtering

1st step : assume we know U_k and λ_k

Given h_{λ_k} an ideal LP, $H_{\lambda_k} = U\hat{H}_{\lambda_k}U^\top = U_k U_k^\top$ is its filter matrix.

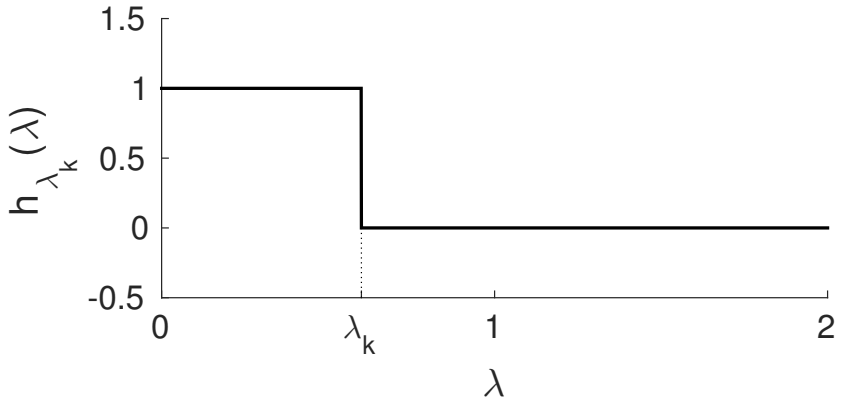


Let $R = (\boldsymbol{r}_1|\boldsymbol{r}_2|\cdots|\boldsymbol{r}_\eta) \in \mathbb{R}^{N\times\eta}$ be a random Gaussian matrix.
We define $\tilde{\boldsymbol{f}}_i = (H_{\lambda_k}R)^\top \boldsymbol{\delta}_i \in \mathbb{R}^\eta$ and $\tilde{D}_{ij} = ||\tilde{\boldsymbol{f}}_i - \tilde{\boldsymbol{f}}_j||$.

Ideal low-pass filtering

1st step : assume we know U_k and λ_k

Given h_{λ_k} an ideal LP, $H_{\lambda_k} = U\hat{H}_{\lambda_k}U^\top = U_k U_k^\top$ is its filter matrix.



Let $R = (\boldsymbol{r}_1|\boldsymbol{r}_2|\cdots|\boldsymbol{r}_\eta) \in \mathbb{R}^{N\times\eta}$ be a random Gaussian matrix.
We define $\tilde{\boldsymbol{f}}_i = (H_{\lambda_k}R)^\top \boldsymbol{\delta}_i \in \mathbb{R}^\eta$ and $\tilde{D}_{ij} = ||\tilde{\boldsymbol{f}}_i - \tilde{\boldsymbol{f}}_j||$.

Norm conservation theorem for ideal filter

Let $\epsilon > 0$, if $\eta > \eta_0 \sim \frac{\log N}{\epsilon^2}$, then, with proba $> 1 - 1/N$, we have :

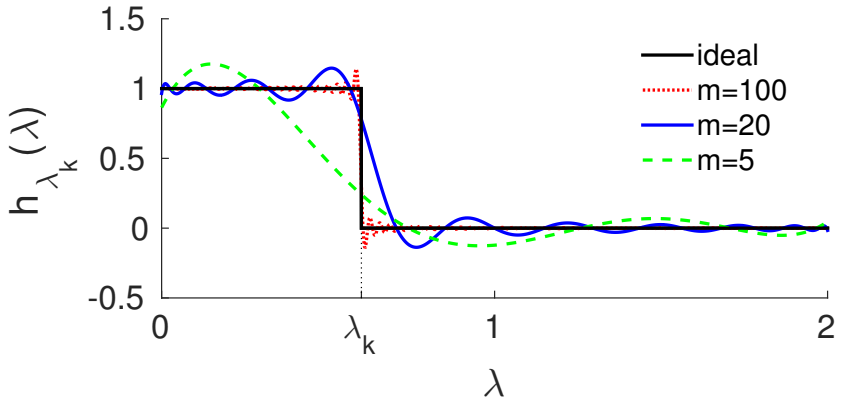
$$\forall (i,j) \in [1,N]^2 \qquad (1-\epsilon)D_{ij} \leq \tilde{D}_{ij} \leq (1+\epsilon)D_{ij}.$$

Non-ideal low-pass filtering

2nd step : assume all we know is λ_k

In practice, we use a poly approx of order m of h_{λ_k} :

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$



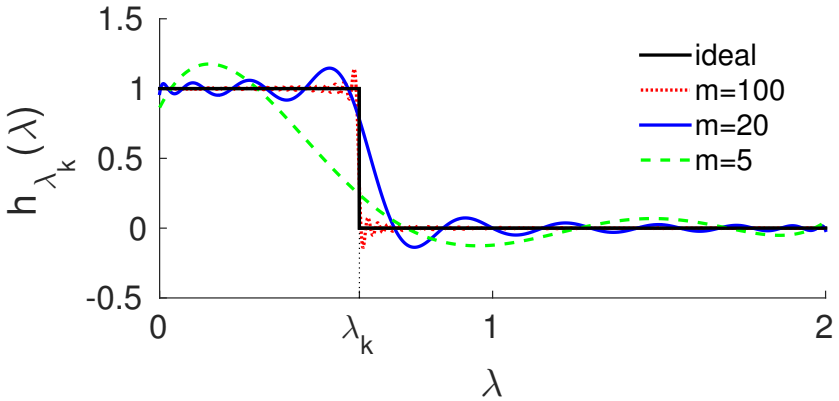
- Does not require the knowledge of U_k.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].

Non-ideal low-pass filtering

2nd step : assume all we know is λ_k

In practice, we use a poly approx of order m of h_{λ_k} :

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$



Indeed, in this case, filtering a vector $\boldsymbol{x}$ reads :

$$\tilde{\mathrm{H}}_{\lambda_k}\boldsymbol{x} = \mathrm{U}\tilde{h}_{\lambda_k}(\Lambda)\mathrm{U}^\top\boldsymbol{x} = \mathrm{U}\sum_{l=1}^{m}\alpha_l\Lambda^l\mathrm{U}^\top\boldsymbol{x} = \sum_{l=1}^{m}\alpha_l\mathrm{L}^l\boldsymbol{x}$$

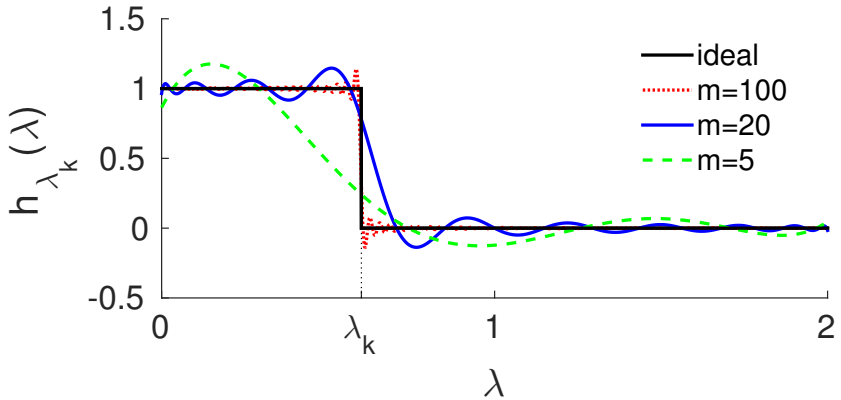
- Does not require the knowledge of U_k.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].

Non-ideal low-pass filtering

2nd step : assume all we know is λ_k

In practice, we use a poly approx of order m of h_{λ_k} :

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$



Indeed, in this case, filtering a vector $\boldsymbol{x}$ reads :

$$\tilde{\mathrm{H}}_{\lambda_k}\boldsymbol{x} = \mathrm{U}\tilde{h}_{\lambda_k}(\Lambda)\mathrm{U}^\top\boldsymbol{x} = \mathrm{U}\sum_{l=1}^{m}\alpha_l\Lambda^l\mathrm{U}^\top\boldsymbol{x} = \sum_{l=1}^{m}\alpha_l\mathrm{L}^l\boldsymbol{x}$$

- Does not require the knowledge of U_k.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].

The theorem stays (more or less) valid with this non-ideal filtering !

Last step : estimate λ_k

Goal : given L, estimate its k-th eigenvalue as fast as possible.

Last step : estimate λ_k

Goal : given L, estimate its k-th eigenvalue as fast as possible.

We use eigencount techniques (also based on polynomial filtering of random vectors !) :

- given the interval $[0, b]$, get an approximation of the number of enclosed eigenvalues.
- And find λ_k by dichotomy on b.

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.
2. Generate η random graph signals in matrix $\mathrm{R} \in \mathbb{R}^{N \times \eta}$.

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.
2. Generate η random graph signals in matrix $\mathrm{R} \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{\mathrm{H}}_{\lambda_k}$ and treat each node i as a point in $\mathbb{R}^{\eta}$:

$$\tilde{\boldsymbol{f}}_i^\top = \boldsymbol{\delta}_i^\top \tilde{\mathrm{H}}_{\lambda_k} \mathrm{R}.$$

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.
2. Generate η random graph signals in matrix $\mathrm{R} \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{\mathrm{H}}_{\lambda_k}$ and treat each node i as a point in $\mathbb{R}^{\eta}$:

$$\tilde{\boldsymbol{f}}_i^{\top} = \boldsymbol{\delta}_i^{\top} \tilde{\mathrm{H}}_{\lambda_k} \mathrm{R}.$$

4. Run k-means with the Euclidean distance : $\tilde{D}_{ij} = ||\tilde{\boldsymbol{f}}_i - \tilde{\boldsymbol{f}}_j||$ and obtain k clusters.

Accelerated spectral algorithm

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.
2. Generate η random graph signals in matrix $\mathrm{R} \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{\mathrm{H}}_{\lambda_k}$ and treat each node i as a point in $\mathbb{R}^{\eta}$:

$$\tilde{\boldsymbol{f}}_i^{\top} = \boldsymbol{\delta}_i^{\top} \tilde{\mathrm{H}}_{\lambda_k} \mathrm{R}.$$

4. Run k-means with the Euclidean distance : $\tilde{D}_{ij} = ||\tilde{\boldsymbol{f}}_i - \tilde{\boldsymbol{f}}_j||$ and obtain k clusters.

Let's work on the second bottleneck : avoid k-means in possibly very large dimension N (step 4).

Fast spectral algorithm ?

Given the N-node graph $\mathcal{G}$ of adjacency matrix W :

1. Estimate λ_k, the k-th eigenvalue of L.
2. Generate η random graph signals in matrix $\mathrm{R} \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{\mathrm{H}}_{\lambda_k}$ and treat each node i as a point in $\mathbb{R}^{\eta}$:

$$\tilde{\boldsymbol{f}}_i^{\top} = \boldsymbol{\delta}_i^{\top} \tilde{\mathrm{H}}_{\lambda_k} \mathrm{R}.$$

4. Sample randomly $\rho \simeq k \log k << N$ nodes out of N :

$$\tilde{\boldsymbol{f}}_i^r = \mathrm{M} \tilde{\boldsymbol{f}}_i = (\mathrm{M} \tilde{\mathrm{H}}_{\lambda_k} \mathrm{R})^{\top} \boldsymbol{\delta}_i^r.$$

5. Run k-means in this reduced space with the Euclidean distance : $\tilde{D}_{ij}^r = ||\tilde{\boldsymbol{f}}_i^r - \tilde{\boldsymbol{f}}_j^r||$ and obtain k clusters.
6. Interpolate cluster indicator functions $\boldsymbol{c}_l^r$ on the whole graph :

$$\tilde{\boldsymbol{c}}_l = \arg \min_{\boldsymbol{x} \in \mathbb{R}^N} ||\mathrm{M}\boldsymbol{x} - \boldsymbol{c}_l^r||^2 + \mu \boldsymbol{x} \mathrm{L}^{\top} \boldsymbol{x}.$$

Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $\mathcal{G}$;

Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $\mathcal{G}$;
2. subsample the set of nodes ;

Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $\mathcal{G}$;
2. subsample the set of nodes ;
3. cluster the reduced set of nodes ;

Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $\mathcal{G}$;
2. subsample the set of nodes ;
3. cluster the reduced set of nodes ;
4. interpolate the cluster indicator vectors back to the complete graph.

This work was done in collaboration with :

- Gilles PUY and Rémi GRIBONVAL from the PANAMA team (INRIA).
- Pierre VANDERGHEYNST from EPFL.

This work was done in collaboration with :

- Gilles PUY and Rémi GRIBONVAL from the PANAMA team (INRIA).
- Pierre VANDERGHEYNST from EPFL.

Part of this work has been published (or submitted) :

- Circumventing the first bottleneck has been accepted to ICASSP 2016
- Interpolation of k-bandlimited graph signals has been submitted to ACHA in November (an application of which helps us circumvent the second bottleneck).

Perspectives and difficult questions

Two difficult questions (among others) :

1. Given a semi-definite positive matrix, how to estimate as fast as possible its k-th eigenvalue, and only that one ?
2. How to subsample ρ nodes out of N while ensuring that clustering them in k classes is the result one would have obtained by clustering all N nodes ?

Perspectives and difficult questions

Two difficult questions (among others) :

1. Given a semi-definite positive matrix, how to estimate as fast as possible its k-th eigenvalue, and only that one ?
2. How to subsample ρ nodes out of N while ensuring that clustering them in k classes is the result one would have obtained by clustering all N nodes ?

Perspectives

1. How about if nodes are added one by one ?
2. Rational filters instead of polynomial filters ?
3. Approximating other spectral clustering algorithms ?