

# Quantum Transport Clustering

The package `quantum_transport_clustering` contains three major class objects:

- `GraphMethods`: construct undirected graphs, and compute and encapsulate their graph Laplacians
- `SpectralClustering`: perform correct spectral clustering on undirected graph Laplacians
- `QuantumTransportClustering`: perform quantum transport clustering on undirected graph Laplacians

Usage example

```
1 import quantum_transport_clustering as qtc
2 graph_ = qtc.GraphMethods(data)
3 ...
4 spec = qtc.SpectralClustering(n_clusters=3, norm_method='row')
5 ...
6 shot = qtc.QuantumTransportClustering(n_clusters=3, Hamiltonian=Lap_)
```

## Graph Methods

```
1 quantum_transport_clustering.GraphMethods(data_, graph_embedded=True, edt_tau =
None, eps_quant = None, normed=True, compute_lap = True)
```

The Class `GraphMethods` is able to

- Generate Gaussian RBF adjacency matrix using Euclidean distances of the data distribution
- Compute Graph Laplacian (symmetrically normalized by default)
- Store the raw data as well as adjacency matrix and graph Laplacian

### Parameters

<code>data_</code>	If <code>graph_embedded = True</code> , <code>data_</code> is a numpy array of shape $(n_{\text{feature}}, m_{\text{sample}})$ , or $m_{\text{sample}}$ points in $\mathbb{R}^{n_{\text{feature}}}$ . If <code>graph_embedded = False</code> , <code>data_</code> is a numpy array of shape $(m_{\text{sample}}, m_{\text{sample}})$ representing the adjacency of a graph with $m_{\text{sample}}$ nodes.
<code>graph_embedded</code>	bool, optional. If <code>True</code> , assume the graph is embedded in a Euclidean space. If <code>False</code> , assume the input data set is an adjacency matrix not <i>a priori</i> embedded in a Euclidean space.
<code>edt_tau</code>	int, $\tau > 0$ , optional. If specified, it is the number of iterations of effective

dissimilarity transformation (EDT). Neglected if `graph_embedded = False`.

<code>eps_quant</code>	float, in range $0 < \epsilon < 100$ , optional. The the quantile of distance distribution. If not specified, $\epsilon = 1$ . Neglected if <code>graph_embedded = False</code> .
<code>normed</code>	bool, optional. If False, graph Laplacian is $L = D - A$ where $D$ is degree diagonal matrix, and $A$ the adjacency matrix. If True, graph Laplacian will be normalized $H = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ .
<code>compute_lap</code>	bool, optional. If True, graph Laplacian will be computed upon initialization.

### Returns

`Lap_` numpy array of shape  $(m_{\text{sample}}, m_{\text{sample}})$ . The graph Laplacian matrix  $L$  or  $H$ .

Example:

```
1 graph_ = qtc.GraphMethods(data)
2 laplacian_matrix_ = graph_.Lap_
```

## Spectral Clustering

```
1 quantum_transport_clustering.SpectralClustering(n_clusters, norm_method='row',
isExact=True)
```

Perform correct spectral clustering on undirected graph Laplacians.

### Parameters

<code>n_clusters</code>	int, $n_{\text{cluster}} > 0$ , the number of clusters.
<code>norm_method</code>	None, "row", or "deg". If None, the spectral embedding is not normalized. If "row", the spectral embedding is $L^2$ -normalized by row where each row represent a node. If "deg", the spectral embedding is normalized by degree vector.
<code>isExact</code>	bool. If True, exact eigenvalues and eigenvectors will be computed. If False, first (small) $n_{\text{cluster}}$ eigenvalues and eigenvectors will be computed.

### Methods

`fit(Lap_)` `Lap_` is the symmetric graph Laplacian. First, the eigenvalues and eigenstates are

computed. Next, perform spectral embedding and  $k$ -means.

## Returns

**labels\_** An integer-valued numpy array of shape  $(m_{\text{sample}})$ . The class labels associated with each node.

Example:

```
1 spec = qtc.SpectralClustering(n_clusters=3, norm_method='row')
2 spec.fit(laplacian_matrix)
3 spec_labels_ = spec.labels_
```

# Quantum Transport Clustering

```
1 quantum_transport_clustering.QuantumTransportClustering(n_clusters, Hamiltonian,
s=1.0, isExact=True, n_eigs = None)
```

Perform quantum transport clustering on undirected graph Laplacians.

## Parameters

**n\_clusters** int,  $n_{\text{cluster}} > 0$ , the number of clusters.

**Hamiltonian** numpy array of shape  $(m_{\text{sample}}, m_{\text{sample}})$ . The symmetric graph Laplacian matrix  $H$ .

**s** float,  $\tilde{s} > 0$ , optional. The actual  $s$ -parameter of Laplace transform will be  $s = \tilde{s} \times (E_{n_{\text{cluster}}-1} - E_0) / (n_{\text{cluster}} - 1)$ , where  $E_n$  are eigenvalues of  $H$ .

**isExact** bool, optional. If True, exact eigenvalues and eigenvectors of  $H$  will be computed. If False, first  $n_{\text{eigs}}$  low energy states will be computed approximated.

**n\_eigs** int,  $n_{\text{eigs}} > 0$ , optional. If  $n_{\text{eigs}}$  not specified and  $\text{isExact} = \text{False}$ ,  $n_{\text{eigs}} = 10 \times n_{\text{cluster}}$ . If  $n_{\text{eigs}}$  is specified and  $\text{isExact} = \text{True}$ , then first  $n_{\text{eigs}}$  low exact energy state will be used to perform quantum transport clustering. The latter case can be used to speed up the clustering processes.

## Methods

**Grind()** Grind( $s=\text{None}$ , grind='medium', method='diff', init\_nodes\_=None) Option grind can be "coarse", "medium", "fine", "micro", or custom. Option method can be diff or

kmeans corresponding to direct difference and  $k$ -means methods. If `grind="custom"`, then `init_nodes_` is the custom python list of initialization nodes. Method `Grind()` produce the array `Omg_` is the matrix  $\Omega$  which contains the raw class labels.

<code>Espresso()</code>	Perform "direct extraction method" on $\Omega$ . This method creates attribute <code>labels_</code> as the predicted class labels.
<code>Coldbrew()</code>	Compute "consensus matrix" $C$ based on $\Omega$ . This method creates attribute <code>consensus_matrix_</code> .

## Returns

<code>Omg_</code>	An integer-valued numpy array of shape $(m_{\text{sample}}, m_{\text{initialization}})$ . The raw class labels of $m_{\text{samples}}$ from quantum transport from $m_{\text{initialization}}$ nodes.
<code>labels_</code>	An integer-valued numpy array of shape $(m_{\text{sample}})$ . The final prediction by <code>Espresso()</code> .
<code>consensus_matrix_</code>	A float-valued numpy array of shape $(m_{\text{sample}}, m_{\text{sample}})$ . The consensus matrix computed by <code>Coldbrew()</code> .

Example:

```

1 shot = qtc.QuantumTransportClustering(n_clusters=3, Hamiltonian=Lap_)
2 shot.Grind()
3 shot.Espresso()
4 class_labels_ = shot.labels_
5 shot.Coldbrew()
6 C_matrix_ = shot.consensus_matrix_

```