Quantum Transport Clustering

The package `quantum_transport_clustering` (written in `python-3.6`) 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

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

Graph Methods

```
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 Lapalcian (symmetrically normalized by default)
- Store the raw data as well as adjacency matrix and graph Laplacian

PARAMETERS	
`data_`	If `graph_embedded = True` , `data_` is a numpy array of shape $(n_{\text{feature}}, m_{\text{sample}})$, or m_{sample} points in $\mathbb{R}^{n_{\text{feature}}}$. If `graph_embedded = False` , `data_` is a numpy array

		of shape (m_{sample}, m_{sample}) representing the adjacency of a graph with m_{sample} nodes.
`graph_embedded`		`bool` , optional. If `True` , assume the graph is embedded in a Euclidean space. If `False` , assume the input data set is an adjacency matrix not *a priori* embedded in a Euclidean space.
`edt_tau`		`int` , $\tau > 0$, optional. If specified, it is the number of iterations of effective dissimilarity transformation (EDT). Neglected if `graph_embedded = False` .
`eps_quant`		`float` , in range $0 < \varepsilon < 100$, optional. The the quantile of distance distribution. If not specified, $\varepsilon = 1$. Neglected if `graph_embedded = False` .
`normed`		`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 $H = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.
`compute_lap`		`bool` , optional. If `True` , graph Laplacian will be computed upon initialization.

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

Example:

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

Spectral Clustering

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

Perform correct spectral clustering on undirected graph Laplacians. Requires `numpy >= 1.13` .

PARAMETERS

`n_clusters`	`int`, $n_{\text{cluster}} > 0$, the number of clusters.
`norm_method`	`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.
`is_exact`	`bool`. If `True`, exact eigenvalues and eigenvectors will be computed. If `False`, first (small) n_{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_{sample}). The class labels associated with each node.

Example:

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

Quantum Transport Clustering

```
quantum_transport_clustering.QuantumTransportClustering(n_clusters, Hamiltonian, s=1.0, is_exact=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.
`is_exact`	`bool`, optional. If `True`, exact eigenvalues and eigenvectors of H will be computed. If `False`, first `n_eigs` low energy states will be computed approximated.
`n_eigs`	`int`, $n_{\text{eigs}} > 0$, optional. If `n_eigs` not specified and `is_exact = False`, $n_{\text{eigs}} = 10 \times n_{\text{cluster}}$. If `n_eigs` is specified and `is_exact = True`, then first n_{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=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()` produces the array `Omega_` or the Ω-matrix which contains the raw class labels.
`Espresso()`	Perform "direct extraction method" on Ω. This method creates attribute `labels_` as the predicted class labels.
`Coldbrew()`	Compute "consensus matrix" C based on Ω. This method creates attribute `consensus_matrix_`.

RETURNS

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

Example:

```
shot = qtc.QuantumTransportClustering(n_clusters=3, Hamiltonian=Lap_) # initialization
Omg_ = shot.Grind() # generate raw class label
# One may extract the eigevalues by attribute shot.Heigval
shot.Espresso() # direct extraction method
class_labels_ = shot.labels_
shot.Coldbrew() # generate consensus matrix
C_matrix_ = shot.consensus_matrix_
```

More in-depth discussions about the spectral clustering and QTC algorithms, including the interpretations of the parameters and variables, can be found at [Quantum Transport Senses Community Structure in Networks].