

Spectral Clustering

The C++ implementation for spectral clustering

Generated by Doxygen 1.8.11

Contents

1	Spectral Clustering	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	7
4.1	DataSet Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Member Data Documentation	7
4.1.2.1	dataMatrix	7
4.1.2.2	dataName	7
4.1.2.3	dataVec	8
4.1.2.4	dimension	8
4.1.2.5	fullName	8
4.1.2.6	maxElements	8
4.1.2.7	strName	8
4.1.2.8	vertexCount	8
4.2	Ensemble Struct Reference	8
4.2.1	Detailed Description	8
4.2.2	Member Data Documentation	9
4.2.2.1	element	9
4.2.2.2	size	9

4.3	Evrot Class Reference	9
4.3.1	Detailed Description	10
4.3.2	Constructor & Destructor Documentation	10
4.3.2.1	Evrot(const Eigen::MatrixXf &X, int method)	10
4.3.2.2	~Evrot()	10
4.3.3	Member Function Documentation	10
4.3.3.1	build_Uab(const Eigen::VectorXf &theta, const int &a, const int &b)	10
4.3.3.2	cluster_assign()	11
4.3.3.3	evqual(const Eigen::MatrixXf &X)	11
4.3.3.4	evqualitygrad(const Eigen::VectorXf &theta, const int &angle_index)	12
4.3.3.5	evrot()	12
4.3.3.6	getClusters()	13
4.3.3.7	getQuality()	13
4.3.3.8	getRotatedEigenVectors()	13
4.3.3.9	gradU(const Eigen::VectorXf &theta, const int &k)	14
4.3.3.10	rotate_givens(const Eigen::VectorXf &theta)	14
4.3.4	Member Data Documentation	14
4.3.4.1	ik	14
4.3.4.2	jk	14
4.3.4.3	mClusters	14
4.3.4.4	mMethod	14
4.3.4.5	mNumAngles	14
4.3.4.6	mNumData	15
4.3.4.7	mNumDims	15
4.3.4.8	mQuality	15
4.3.4.9	mX	15
4.3.4.10	mXrot	15
4.4	Para Struct Reference	15
4.4.1	Detailed Description	15
4.4.2	Member Data Documentation	16

4.4.2.1	extractOption	16
4.4.2.2	isDistSorted	16
4.4.2.3	LaplacianOption	16
4.4.2.4	mMethod	16
4.4.2.5	postProcessing	16
4.4.2.6	sampled	16
4.5	SpectralClustering Class Reference	16
4.5.1	Detailed Description	18
4.5.2	Constructor & Destructor Documentation	18
4.5.2.1	SpectralClustering()	18
4.5.2.2	SpectralClustering(const int &argc, char **argv, const Para &p, bool &automatic)	18
4.5.2.3	~SpectralClustering()	18
4.5.3	Member Function Documentation	18
4.5.3.1	clusterByNorm(const int &norm)	18
4.5.3.2	extractFeatures(const std::vector< int > &storage, const std::vector< std::vector< int > > &neighborVec, const Eigen::MatrixXf &centroid)	20
4.5.3.3	getAdjacencyMatrix(Eigen::MatrixXf &adjacencyMatrix)	21
4.5.3.4	getDegreeMatrix(const Eigen::MatrixXf &adjacencyMatrix, Eigen::DiagonalMatrix< float, Dynamic > &degreeMatrix)	22
4.5.3.5	getEigenClustering(const Eigen::MatrixXf &laplacianMatrix, const int &norm)	22
4.5.3.6	getEigvecRotation(std::vector< int > &storage, std::vector< std::vector< int > > &neighborVec, Eigen::MatrixXf &clusterCenter, const Eigen::MatrixXf &X)	23
4.5.3.7	getEntropyRatio(const std::vector< int > &storage, float &EntropyRatio)	24
4.5.3.8	getLaplacianMatrix(const Eigen::MatrixXf &adjacencyMatrix, Eigen::DiagonalMatrix< float, Dynamic > &degreeMatrix, Eigen::MatrixXf &laplacianMatrix)	24
4.5.3.9	getParameterUserInput()	25
4.5.3.10	getSignalList()	26
4.5.3.11	normalizeEigvec(Eigen::MatrixXf &eigenVec)	27
4.5.3.12	performClustering()	27
4.5.3.13	performKMeans(const Eigen::MatrixXf &eigenVec, std::vector< int > &storage, std::vector< std::vector< int > > &neighborVec)	28
4.5.3.14	recordOptimalResult(const int &normOption, const int &clusNum)	29
4.5.3.15	recordPreset(const int &number)	30

4.5.3.16	<code>setDataset(const int &argc, char **argv)</code>	30
4.5.3.17	<code>setLabel(vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid)</code>	30
4.5.3.18	<code>setParameterAutomatic(const Para &p)</code>	31
4.5.4	Member Data Documentation	32
4.5.4.1	<code>activityList</code>	32
4.5.4.2	<code>distRange</code>	32
4.5.4.3	<code>ds</code>	32
4.5.4.4	<code>extractOption</code>	32
4.5.4.5	<code>group</code>	32
4.5.4.6	<code>initializationOption</code>	32
4.5.4.7	<code>isDistSorted</code>	32
4.5.4.8	<code>isOptimal</code>	32
4.5.4.9	<code>isPathlines</code>	32
4.5.4.10	<code>LaplacianOption</code>	32
4.5.4.11	<code>mMaxQuality</code>	33
4.5.4.12	<code>mMethod</code>	33
4.5.4.13	<code>normOption</code>	33
4.5.4.14	<code>numberOfClusters</code>	33
4.5.4.15	<code>object</code>	33
4.5.4.16	<code>postProcessing</code>	33
4.5.4.17	<code>presetNumber</code>	33
4.5.4.18	<code>readCluster</code>	33
4.5.4.19	<code>SCALING</code>	33
4.5.4.20	<code>sigmaVec</code>	33
4.5.4.21	<code>timeList</code>	33

5 File Documentation	35
5.1 Evrot.cpp File Reference	35
5.2 Evrot.h File Reference	35
5.2.1 Macro Definition Documentation	36
5.2.1.1 EPS	36
5.3 main.cpp File Reference	36
5.3.1 Function Documentation	37
5.3.1.1 main(int argc, char **argv)	37
5.3.1.2 setPara(Para &p)	37
5.4 Predefined.cpp File Reference	38
5.4.1 Function Documentation	38
5.4.1.1 deleteVecElements(std::vector< T > &original, const T &first, const T &second)	38
5.5 Predefined.h File Reference	39
5.5.1 Function Documentation	40
5.5.1.1 deleteVecElements(std::vector< T > &original, const T &first, const T &second)	40
5.6 README.md File Reference	40
5.7 SpectralClustering.cpp File Reference	40
5.7.1 Function Documentation	40
5.7.1.1 getMatrixPow(Eigen::DiagonalMatrix< float, Dynamic > &matrix, const float &powNumber)	40
5.8 SpectralClustering.h File Reference	41
5.8.1 Macro Definition Documentation	42
5.8.1.1 GradientStep	42
5.8.2 Function Documentation	42
5.8.2.1 getMatrixPow(Eigen::DiagonalMatrix< float, Dynamic > &matrix, const float &powNumber)	42
Index	43

Chapter 1

Spectral Clustering

Spectral clustering is a popular clustering technique that builds the normalized cut minimization for the input distance matrix. It includes two popular versions in flow visualization

- Spectral clustering (SC) with eigenrotation minimization (SC-eigen)
 - It can find the optimal number of clusters given the distance matrix and a preset bound k
 - It is very time consuming with complicated eigenrotation minimization inside the range
- k-means (SC k-means)
 - It finds the natural clusters with user input parameters after the generation of embedding space

Possible in the future we will implement a third version of spectral clustering, k-way normalized cut which has been found in flow visualization literature.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DataSet	7
Ensemble	8
Evrot	9
Para	15
SpectralClustering	16

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Evrot.cpp	35
Evrot.h	35
main.cpp	36
Predefined.cpp	38
Predefined.h	39
SpectralClustering.cpp	40
SpectralClustering.h	41

Chapter 4

Class Documentation

4.1 DataSet Struct Reference

```
#include <Predefined.h>
```

Public Attributes

- vector< vector< float > > [dataVec](#)
- Eigen::MatrixXf [dataMatrix](#)
- int [maxElements](#) = -1
- int [vertexCount](#) = -1
- int [dimension](#) = -1
- string [strName](#)
- string [fullName](#)
- string [dataName](#)

4.1.1 Detailed Description

Definition at line 20 of file Predefined.h.

4.1.2 Member Data Documentation

4.1.2.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 23 of file Predefined.h.

4.1.2.2 string DataSet::dataName

Definition at line 30 of file Predefined.h.

4.1.2.3 `vector<vector<float>> DataSet::dataVec`

Definition at line 22 of file `Predefined.h`.

4.1.2.4 `int DataSet::dimension = -1`

Definition at line 26 of file `Predefined.h`.

4.1.2.5 `string DataSet::fullName`

Definition at line 29 of file `Predefined.h`.

4.1.2.6 `int DataSet::maxElements = -1`

Definition at line 24 of file `Predefined.h`.

4.1.2.7 `string DataSet::strName`

Definition at line 28 of file `Predefined.h`.

4.1.2.8 `int DataSet::vertexCount = -1`

Definition at line 25 of file `Predefined.h`.

The documentation for this struct was generated from the following file:

- [Predefined.h](#)

4.2 Ensemble Struct Reference

```
#include <Predefined.h>
```

Public Attributes

- `int` [size](#)
- `std::vector< int >` [element](#)

4.2.1 Detailed Description

Definition at line 38 of file `Predefined.h`.

4.2.2 Member Data Documentation

4.2.2.1 `std::vector<int> Ensemble::element`

Definition at line 41 of file `Predefined.h`.

4.2.2.2 `int Ensemble::size`

Definition at line 40 of file `Predefined.h`.

The documentation for this struct was generated from the following file:

- [Predefined.h](#)

4.3 Evrot Class Reference

```
#include <Evrot.h>
```

Public Member Functions

- [Evrot](#) (const Eigen::MatrixXf &X, int method)
- virtual [~Evrot](#) ()
- float [getQuality](#) ()
- `std::vector< std::vector< int > >` [getClusters](#) ()
- Eigen::MatrixXf & [getRotatedEigenVectors](#) ()

Protected Member Functions

- void [evrot](#) ()
- void [cluster_assign](#) ()
- float [evqual](#) (const Eigen::MatrixXf &X)
- float [evqualitygrad](#) (const Eigen::VectorXf &theta, const int &angle_index)
- Eigen::MatrixXf [rotate_givens](#) (const Eigen::VectorXf &theta)
- Eigen::MatrixXf [build_Uab](#) (const Eigen::VectorXf &theta, const int &a, const int &b)
- Eigen::MatrixXf [gradU](#) (const Eigen::VectorXf &theta, const int &k)

Protected Attributes

- int [mMethod](#)
- const int [mNumDims](#)
- const int [mNumData](#)
- int [mNumAngles](#)
- Eigen::VectorXi [ik](#)
- Eigen::VectorXi [jk](#)
- Eigen::MatrixXf [mX](#)
- Eigen::MatrixXf [mXrot](#)
- float [mQuality](#)
- `std::vector< std::vector< int > >` [mClusters](#)

4.3.1 Detailed Description

Definition at line 30 of file Evrot.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Evrot::Evrot (const Eigen::MatrixXf & X, int method)

Definition at line 12 of file Evrot.cpp.

```

12                                     :
13     mMethod(method),
14     mNumDims(X.cols()),
15     mNumData(X.rows()),
16     mNumAngles((int)(mNumDims*(mNumDims-1)/2)), // get the number of angles
17     ik(Eigen::VectorXi(mNumAngles)),
18     jk(Eigen::VectorXi(mNumAngles)),
19     mX(X),
20     mClusters(std::vector<std::vector<int> >(mNumDims)) //allocate clusters vector
21 {
22     // build index mapping (to index upper triangle)
23     int k = 0;
24     for( int i=0; i<mNumDims-1; i++ ){
25         for( int j=i+1; j<=mNumDims-1; j++ ){
26             ik[k] = i;
27             jk[k] = j;
28             k++;
29         }
30     }
31
32     evrot();
33 }
```

4.3.2.2 Evrot::~Evrot () [virtual]

Definition at line 37 of file Evrot.cpp.

```

38 {
39
40 }
```

4.3.3 Member Function Documentation

4.3.3.1 Eigen::MatrixXf Evrot::build_Uab (const Eigen::VectorXf & theta, const int & a, const int & b) [protected]

Definition at line 243 of file Evrot.cpp.

```

244 {
245     int k,i;
246     //set Uab to be an identity matrix
247     Eigen::MatrixXf Uab(mNumDims,mNumDims);
248     Uab.setZero();
249     Uab.setIdentity();
250
251     if( b < a ) {
252         return Uab;
253     }
254
255     float tt,u_ik;
256     for( k=a; k<=b; k++ ){
257         tt = theta[k];
258         #pragma omp parallel for schedule(static) num_threads(8)
259         for( i=0; i<mNumDims; i++ ) {
260             u_ik = Uab(i,ik[k]) * cos(tt) - Uab(i,jk[k]) * sin(tt);
261             Uab(i,jk[k]) = Uab(i,ik[k]) * sin(tt) + Uab(i,jk[k]) * cos(tt);
262             Uab(i,ik[k]) = u_ik;
263         }
264     }
265     return Uab;
266 }
```

4.3.3.2 void Evrot::cluster_assign () [protected]

Definition at line 127 of file Evrot.cpp.

```

128 {
129     // find max of each row
130     Eigen::VectorXi max_index_col(mNumData);
131     #pragma omp parallel for schedule(static) num_threads(8)
132     for (int i=0; i<mNumData; i++ )
133     {
134         int col=0;
135         float mValue = mXrot.row(i).cwiseAbs().maxCoeff(&col);
136         /*
137         int row, col;
138         mXrot.row(i).cwise().abs().maxCoeff(&row, &col);
139         */
140         max_index_col[i] = col;
141     }
142
143     // prepare cluster assignments
144     #pragma omp parallel for schedule(static) num_threads(8)
145     for(int j=0; j<mNumDims; j++ )
146     { // loop over all columns
147         for(int i=0; i<mNumData; i++ )
148         { // loop over all rows
149             if( max_index_col[i] == j ){
150                 mClusters[j].push_back(i);
151             }
152         }
153     }
154
155     /* delete cluster that has zero elements in case that vanishing vector won't create trouble */
156     std::vector<std::vector<int> > tempCluster;
157     for(int i=0; i<mClusters.size(); ++i)
158         if(!mClusters[i].empty())
159             tempCluster.push_back(mClusters[i]);
160     mClusters.clear();
161     mClusters = tempCluster;
162 }

```

4.3.3.3 float Evrot::evqual (const Eigen::MatrixXf & X) [protected]

Definition at line 165 of file Evrot.cpp.

```

166 {
167     // take the square of all entries and find max of each row
168     Eigen::MatrixXf X2(X.rows(), X.cols());
169     #pragma omp parallel for schedule(static) num_threads(8)
170     for(int i=0; i<X.rows(); ++i)
171     {
172         for(int j=0; j<X.cols(); ++j)
173         {
174             X2(i,j)=X(i,j)*X(i,j);
175         }
176     }
177
178     Eigen::VectorXf max_values(X.rows());
179
180     #pragma omp parallel for schedule(static) num_threads(8)
181     for(int i=0; i<X.rows(); ++i)
182         max_values(i)=X2.row(i).maxCoeff();
183
184     // compute cost
185     #pragma omp parallel for schedule(static) num_threads(8)
186     for (int i=0; i<mNumData; i++ )
187     {
188         X2.row(i) = X2.row(i) / max_values[i];
189     }
190
191     float J = 1.0 - (X2.sum()/mNumData -1.0)/mNumDims;
192
193     return J;
194 }

```

4.3.3.4 float Evrot::evqualitygrad (const Eigen::VectorXf & theta, const int & angle_index) [protected]

Definition at line 198 of file Evrot.cpp.

```

199 {
200     // build V,U,A
201     Eigen::MatrixXf V = gradU(theta, angle_index);
202
203     Eigen::MatrixXf U1 = build_Uab(theta, 0,angle_index-1);
204     Eigen::MatrixXf U2 = build_Uab(theta, angle_index+1,mNumAngles-1);
205
206     Eigen::MatrixXf A = mX*U1*V*U2;
207
208     // rotate vecs according to current angles
209     Eigen::MatrixXf Y = rotate_givens(theta);
210
211     // find max of each row
212     Eigen::VectorXf max_values(mNumData);
213     Eigen::VectorXi max_index_col(mNumData);
214     #pragma omp parallel for schedule(static) num_threads(8)
215     for (int i=0; i<mNumData; i++ ) {
216         int row, col;
217         Y.row(i).cwiseAbs().maxCoeff(&row, &col);
218         max_values[i] = Y(i,col);
219         max_index_col[i] = col;
220     }
221
222     // compute gradient
223     float dJ=0, tmp1, tmp2;
224     for( int j=0; j<mNumDims; j++ ){ // loop over all columns
225         for( int i=0; i<mNumData; i++ ){ // loop over all rows
226             tmp1 = A(i,j) * Y(i,j) / (max_values[i]*max_values[i]);
227             tmp2 = A(i,max_index_col[i]) * (Y(i,j)*Y(i,j)) / (max_values[i]*max_values[i]*max_values[i]);
228             dJ += tmp1-tmp2;
229         }
230     }
231     dJ = 2*dJ/mNumData/mNumDims;
232
233     return dJ;
234 }

```

4.3.3.5 void Evrot::evrot () [protected]

Definition at line 44 of file Evrot.cpp.

```

45 {
46
47     // definitions
48     int max_iter = 100;
49     float dQ,Q,Q_new,Q_old1,Q_old2,Q_up,Q_down;
50     float alpha;
51     int iter,d;
52
53     Eigen::VectorXf theta = Eigen::VectorXf::Zero(mNumAngles);
54     Eigen::VectorXf theta_new = Eigen::VectorXf::Zero(mNumAngles);
55
56     Q = evqual(mX); // initial quality
57
58     Q_old1 = Q;
59     Q_old2 = Q;
60     iter = 0;
61
62     while( iter < max_iter ){ // iterate to refine quality
63         iter++;
64         for( d = 0; d < mNumAngles; d++ ){
65             if( mMethod == 2 ){ // descend through numerical derivative
66                 alpha = 0.1;
67                 {
68                     // move up
69                     theta_new[d] = theta[d] + alpha;
70                     Eigen::MatrixXf Xrot = rotate_givens(theta_new);
71                     Q_up = evqual(Xrot);
72                 }
73                 {
74                     // move down
75                     theta_new[d] = theta[d] - alpha;

```

```

76         Eigen::MatrixXf Xrot = rotate_givens(theta_new);
77         Q_down = evqual(Xrot);
78     }
79
80     // update only if at least one of them is better
81     if( Q_up > Q || Q_down > Q){
82         if( Q_up > Q_down ){
83             theta[d] = theta[d] + alpha;
84             theta_new[d] = theta[d];
85             Q = Q_up;
86         } else {
87             theta[d] = theta[d] - alpha;
88             theta_new[d] = theta[d];
89             Q = Q_down;
90         }
91     }
92     } else { // descend through true derivative
93         alpha = 1.0;
94         dQ = evqualitygrad(theta, d);
95         theta_new[d] = theta[d] - alpha * dQ;
96         Eigen::MatrixXf Xrot = rotate_givens(theta_new);
97         Q_new = evqual(Xrot);
98         if( Q_new > Q){
99             theta[d] = theta_new[d];
100             Q = Q_new;
101         }
102         else{
103             theta_new[d] = theta[d];
104         }
105     }
106 }
107 // stopping criteria
108 if( iter > 2 ){
109     if( Q - Q_old2 < 1e-3 ){
110         break;
111     }
112 }
113 Q_old2 = Q_old1;
114 Q_old1 = Q;
115 }
116
117 mXrot = rotate_givens(theta_new);
118 cluster_assign();
119
120 //output
121 mQuality = Q;
122 }

```

4.3.3.6 `std::vector<std::vector<int>> Evrot::getClusters ()` [inline]

Definition at line 36 of file Evrot.h.

```
36 { return mClusters; }
```

4.3.3.7 `float Evrot::getQuality ()` [inline]

Definition at line 35 of file Evrot.h.

```
35 { return mQuality; }
```

4.3.3.8 `Eigen::MatrixXf& Evrot::getRotatedEigenVectors ()` [inline]

Definition at line 37 of file Evrot.h.

```
37 { return mXrot; }
```

4.3.3.9 Eigen::MatrixXf Evrot::gradU (const Eigen::VectorXf & *theta*, const int & *k*) [protected]

Definition at line 268 of file Evrot.cpp.

```

269 {
270     Eigen::MatrixXf V(mNumDims,mNumDims);
271     V.setZero();
272
273     V(ik[k],ik[k]) = -sin(theta[k]);
274     V(ik[k],jk[k]) = cos(theta[k]);
275     V(jk[k],ik[k]) = -cos(theta[k]);
276     V(jk[k],jk[k]) = -sin(theta[k]);
277
278     return V;
279 }
```

4.3.3.10 Eigen::MatrixXf Evrot::rotate_givens (const Eigen::VectorXf & *theta*) [protected]

Definition at line 236 of file Evrot.cpp.

```

237 {
238     Eigen::MatrixXf G = build_Uab(theta, 0, mNumAngles-1);
239     Eigen::MatrixXf Y = mX*G;
240     return Y;
241 }
```

4.3.4 Member Data Documentation

4.3.4.1 Eigen::VectorXi Evrot::ik [protected]

Definition at line 54 of file Evrot.h.

4.3.4.2 Eigen::VectorXi Evrot::jk [protected]

Definition at line 55 of file Evrot.h.

4.3.4.3 std::vector<std::vector<int> > Evrot::mClusters [protected]

Definition at line 62 of file Evrot.h.

4.3.4.4 int Evrot::mMethod [protected]

Definition at line 50 of file Evrot.h.

4.3.4.5 int Evrot::mNumAngles [protected]

Definition at line 53 of file Evrot.h.

4.3.4.6 `const int Evrot::mNumData` [protected]

Definition at line 52 of file Evrot.h.

4.3.4.7 `const int Evrot::mNumDims` [protected]

Definition at line 51 of file Evrot.h.

4.3.4.8 `float Evrot::mQuality` [protected]

Definition at line 60 of file Evrot.h.

4.3.4.9 `Eigen::MatrixXf Evrot::mX` [protected]

Definition at line 58 of file Evrot.h.

4.3.4.10 `Eigen::MatrixXf Evrot::mXrot` [protected]

Definition at line 59 of file Evrot.h.

The documentation for this class was generated from the following files:

- [Evrot.h](#)
- [Evrot.cpp](#)

4.4 Para Struct Reference

```
#include <SpectralClustering.h>
```

Public Attributes

- int [sampled](#)
- int [LaplacianOption](#)
- bool [isDistSorted](#)
- int [postProcessing](#)
- int [mMethod](#)
- int [extractOption](#)

4.4.1 Detailed Description

Definition at line 38 of file SpectralClustering.h.

4.4.2 Member Data Documentation

4.4.2.1 int Para::extractOption

Definition at line 69 of file SpectralClustering.h.

4.4.2.2 bool Para::isDistSorted

Definition at line 54 of file SpectralClustering.h.

4.4.2.3 int Para::LaplacianOption

Definition at line 49 of file SpectralClustering.h.

4.4.2.4 int Para::mMethod

Definition at line 64 of file SpectralClustering.h.

4.4.2.5 int Para::postProcessing

Definition at line 59 of file SpectralClustering.h.

4.4.2.6 int Para::sampled

Definition at line 44 of file SpectralClustering.h.

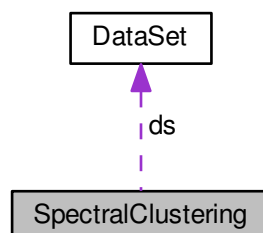
The documentation for this struct was generated from the following file:

- [SpectralClustering.h](#)

4.5 SpectralClustering Class Reference

```
#include <SpectralClustering.h>
```

Collaboration diagram for SpectralClustering:



Public Member Functions

- [SpectralClustering](#) ()
- [SpectralClustering](#) (const int &argc, char **argv, const [Para](#) &p, bool &automatic)
- [~SpectralClustering](#) ()
- void [performClustering](#) ()

Private Member Functions

- void [extractFeatures](#) (const std::vector< int > &storage, const std::vector< std::vector< int > > &neighborVec, const Eigen::MatrixXf ¢roid)
- void [setDataset](#) (const int &argc, char **argv)
- void [getParameterUserInput](#) ()
- void [setParameterAutomatic](#) (const [Para](#) &p)
- void [clusterByNorm](#) (const int &norm)
- void [setLabel](#) (vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf ¢roid)
- void [getAdjacencyMatrix](#) (Eigen::MatrixXf &adjacencyMatrix)
- void [getDegreeMatrix](#) (const Eigen::MatrixXf &adjacencyMatrix, Eigen::DiagonalMatrix< float, Dynamic > °reeMatrix)
- void [getLaplacianMatrix](#) (const Eigen::MatrixXf &adjacencyMatrix, Eigen::DiagonalMatrix< float, Dynamic > °reeMatrix, Eigen::MatrixXf &laplacianMatrix)
- void [getEigenClustering](#) (const Eigen::MatrixXf &laplacianMatrix, const int &norm)
- void [getSigmaList](#) ()
- void [getEntropyRatio](#) (const std::vector< int > &storage, float &EntropyRatio)
- void [recordPreset](#) (const int &number)
- void [recordOptimalResult](#) (const int &normOption, const int &clusNum)
- void [normalizeEigenvec](#) (Eigen::MatrixXf &eigenVec)
- void [performKMeans](#) (const Eigen::MatrixXf &eigenVec, std::vector< int > &storage, std::vector< std::vector< int > > &neighborVec)
- void [getEigvecRotation](#) (std::vector< int > &storage, std::vector< std::vector< int > > &neighborVec, Eigen::MatrixXf &clusterCenter, const Eigen::MatrixXf &X)

Private Attributes

- MetricPreparation [object](#)
- int [normOption](#) = -1
- std::vector< int > [group](#)
- std::vector< string > [activityList](#)
- std::vector< string > [timeList](#)
- [DataSet](#) [ds](#)
- int [numberOfClusters](#) = -1
- int [initializationOption](#) = -1
- std::vector< float > [distRange](#)
- std::vector< float > [sigmaVec](#)
- int [LaplacianOption](#) = -1
- bool [isDistSorted](#) = -1
- int [postProcessing](#) = -1
- int [extractOption](#) = -1
- int [SCALING](#)
- bool [isOptimal](#)
- int [presetNumber](#)
- bool [readCluster](#)
- bool [isPathlines](#)
- float [mMaxQuality](#) = 0
- int [mMethod](#) = -1

4.5.1 Detailed Description

Definition at line 76 of file SpectralClustering.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 SpectralClustering::SpectralClustering ()

Definition at line 21 of file SpectralClustering.cpp.

```
22 {  
23  
24 }
```

4.5.2.2 SpectralClustering::SpectralClustering (const int & argc, char ** argv, const Para & p, bool & automatic)

Definition at line 35 of file SpectralClustering.cpp.

```
36 {  
37     setDataset(argc, argv);  
38  
39     if (automatic)  
40         setParameterAutomatic(p);  
41     else  
42         getParameterUserInput();  
43  
44 }
```

4.5.2.3 SpectralClustering::~SpectralClustering ()

Definition at line 50 of file SpectralClustering.cpp.

```
51 {  
52     deleteDistanceMatrix(ds.dataMatrix.rows());  
53 }
```

4.5.3 Member Function Documentation

4.5.3.1 void SpectralClustering::clusterByNorm (const int & norm) [private]

Definition at line 173 of file SpectralClustering.cpp.

```

174 {
175     normOption = norm;
176
177     /* very hard to decide whether needed to perform such pre-processing */
178     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
179     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
180         ds.dataMatrix.cols(), normOption);
181
182     /* would store distance matrix instead because it would save massive time */
183     struct timeval start, end;
184     double timeTemp;
185     gettimeofday(&start, NULL);
186
187     deleteDistanceMatrix(ds.dataMatrix.rows());
188
189     std::ifstream distFile("../dataset/"+to_string(normOption)).c_str(), ios::in);
190     if(distFile.fail())
191     {
192         distFile.close();
193         getDistanceMatrix(ds.dataMatrix, normOption, object);
194         std::ofstream distFileOut("../dataset/"+to_string(normOption)).c_str(), ios::out);
195         for(int i=0; i<ds.dataMatrix.rows(); ++i)
196         {
197             for(int j=0; j<ds.dataMatrix.rows(); ++j)
198             {
199                 distFileOut << distanceMatrix[i][j] << " ";
200             }
201             distFileOut << std::endl;
202         }
203         distFileOut.close();
204     }
205     else
206     {
207         std::cout << "read distance matrix..." << std::endl;
208
209         distanceMatrix = new float*[ds.dataMatrix.rows()];
210         #pragma omp parallel for schedule(static) num_threads(8)
211         for (int i = 0; i < ds.dataMatrix.rows(); ++i)
212         {
213             distanceMatrix[i] = new float[ds.dataMatrix.rows()];
214         }
215         int i=0, j;
216         string line;
217         stringstream ss;
218         while(getline(distFile, line))
219         {
220             j=0;
221             ss.str(line);
222             while(ss>>line)
223             {
224                 if(i==j)
225                     distanceMatrix[i][j]=0;
226                 else
227                     distanceMatrix[i][j] = std::atof(line.c_str());
228                 ++j;
229             }
230             ++i;
231             ss.str("");
232             ss.clear();
233         }
234         distFile.close();
235     }
236
237     gettimeofday(&end, NULL);
238     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
239         + end.tv_usec - start.tv_usec) / 1.e6;
240     activityList.push_back("Distance matrix computing for norm "+to_string(norm)+" takes: ");
241     timeList.push_back(to_string(timeTemp)+" s");
242
243     getSigmaList();
244
245     Eigen::MatrixXf adjacencyMatrix, laplacianMatrix;
246     Eigen::DiagonalMatrix<float, Dynamic> degreeMatrix;
247
248     /* get weighted adjacency matrix by Gaussian kernel */
249     getAdjacencyMatrix(adjacencyMatrix);
250
251     /* get degree matrix */
252     getDegreeMatrix(adjacencyMatrix, degreeMatrix);
253
254     /* get Laplacian matrix */
255     getLaplacianMatrix(adjacencyMatrix, degreeMatrix, laplacianMatrix);
256
257     getEigenClustering(laplacianMatrix, norm);
258 }
259

```

4.5.3.2 `void SpectralClustering::extractFeatures (const std::vector< int > & storage, const std::vector< std::vector< int > > & neighborVec, const Eigen::MatrixXf & centroid) [private]`

Definition at line 314 of file SpectralClustering.cpp.

```

316 {
317     const int& Row = ds.dataMatrix.rows();
318     const int& Column = ds.dataMatrix.cols();
319
320     std::cout << "Final group number information: " << std::endl;
321     for (int i = 0; i < storage.size(); ++i)
322     {
323         std::cout << storage[i] << " ";
324     }
325     std::cout << std::endl;
326
327     string pprocessing;
328     switch(postProcessing)
329     {
330     case 1:
331         pprocessing="Kmeans";
332         break;
333     case 2:
334         pprocessing="EigenRot";
335         break;
336     }
337
338     float EntropyRatio;
339     getEntropyRatio(storage, EntropyRatio);
340
341     /* record labeling information */
342     // IOHandler::generateGroups(neighborVec);
343
344
345     IOHandler::printClusters(ds.dataVec, group, storage, "SC_"+pprocessing+"_norm"+to_string(
normOption), ds.fullName, ds.dimension);
346
347     struct timeval start, end;
348     double timeTemp;
349
350     /* compute the centroid coordinates of each clustered group */
351
352     gettimeofday(&start, NULL);
353
354     vector<vector<float>> > closest(numberOfClusters);
355     vector<vector<float>> > furthest(numberOfClusters);
356
357     /* extract the closest and furthest streamlines to centroid */
358
359     #pragma omp parallel for schedule(static) num_threads(8)
360     for (int i=0; i<numberOfClusters; ++i)
361     {
362         float minDist = FLT_MAX;
363         float maxDist = -10;
364         int minIndex = -1, maxIndex = -1;
365         const std::vector<int>& groupRow = neighborVec[i];
366         const Eigen::VectorXf& eachCentroid = centroid.row(i);
367         for (int j = 0; j < groupRow.size(); ++j)
368         {
369             float distance = getDisimilarity(eachCentroid, ds.dataMatrix, groupRow[j],
normOption, object);
370             if(minDist>distance)
371             {
372                 minDist = distance;
373                 minIndex = groupRow[j];
374             }
375             if(maxDist<distance)
376             {
377                 maxDist = distance;
378                 maxIndex = groupRow[j];
379             }
380         }
381         closest[i] = ds.dataVec[minIndex];
382         furthest[i] = ds.dataVec[maxIndex];
383     }
384
385     std::vector<std::vector<float>> > center_vec(numberOfClusters, vector<float>(Column));
386     #pragma omp parallel for schedule(static) num_threads(8)
387     for (int i = 0; i < center_vec.size(); ++i)
388     {
389         for (int j = 0; j < Column; ++j)
390         {

```

```

392         center_vec[i][j] = centroid(i,j);
393     }
394 }
395
396 gettimeofday(&end, NULL);
397 timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
398             + end.tv_usec - start.tv_usec) / 1.e6;
399 activityList.push_back("Feature extraction takes: ");
400 timeList.push_back(to_string(timeTemp)+" s");
401
402 ValidityMeasurement vm;
403 vm.computeValue(normOption, ds.dataMatrix, group, object, false);
404 activityList.push_back("SC Validity measure is: ");
405 stringstream fc_ss;
406 fc_ss << vm.f_c;
407 timeList.push_back(fc_ss.str());
408
409 gettimeofday(&start, NULL);
410 Silhouette sil;
411 sil.computeValue(normOption, ds.dataMatrix, ds.
dataMatrix.rows(), ds.dataMatrix.cols(), group, object,
412                 numberOfClusters, false, neighborVec);
413 gettimeofday(&end, NULL);
414 timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
415 activityList.push_back("Silhouette calculation takes: ");
416 timeList.push_back(to_string(timeTemp)+" s");
417
418 std::cout << "Finishing extracting features!" << std::endl;
419
420 stringstream ss;
421 ss << "norm_" << normOption;
422
423 std::vector<float> closestRotation, furthestRotation;
424 const float& closestAverage = getRotation(closest, closestRotation);
425 const float& furthestAverage = getRotation(furthest, furthestRotation);
426
427 /* save closest, furthest and centroid representative streamlines */
428 IOHandler::printFeature(ds.dataName+"_SC"+pprocessing+"_closest_"+ss.str()+".vtk", closest,
sil.sCluster,
429                         closestRotation, ds.dimension);
430 IOHandler::printFeature(ds.dataName+"_SC"+pprocessing+"_furthest_"+ss.str()+".vtk", furthest,
sil.sCluster,
431                         furthestRotation, ds.dimension);
432 IOHandler::printFeature(ds.dataName+"_SC"+pprocessing+"_centroid_"+ss.str()+".vtk",
center_vec, sil.sCluster, ds.dimension);
433
434 IOHandler::printToFull(ds.dataVec, sil.sData, "SC"+pprocessing+"_SValueLine_"+ss.str(),
ds.fullName, ds.dimension);
435 IOHandler::printToFull(ds.dataVec, group, sil.sCluster, "SC"+pprocessing+"_SValueCluster_
"+ss.str(), ds.fullName, ds.dimension);
436
437 activityList.push_back("numCluster is: ");
438 timeList.push_back(to_string(numberOfClusters));
439
440 activityList.push_back("Norm option is: ");
441 timeList.push_back(to_string(normOption));
442
443 activityList.push_back("SC post-processing is: ");
444 switch(postProcessing)
445 {
446 case 1:
447     timeList.push_back("k-means");
448     break;
449
450 case 2:
451     timeList.push_back("vector rotation");
452     break;
453 }
454
455 activityList.push_back("Average rotation of closest is: ");
456 timeList.push_back(to_string(closestAverage));
457
458 activityList.push_back("Average rotation of furthest is: ");
459 timeList.push_back(to_string(furthestAverage));
460
461 IOHandler::generateReadme(activityList, timeList);
462
463 IOHandler::writeReadme(EntropyRatio, sil, "For norm "+to_string(normOption));
464 }

```

4.5.3.3 void SpectralClustering::getAdjacencyMatrix (Eigen::MatrixXf & adjacencyMatrix) [private]

Definition at line 579 of file SpectralClustering.cpp.

```

580 {
581     //in case of diagonal matrix element is not assigned
582     adjacencyMatrix = Eigen::MatrixXf::Zero(ds.dataMatrix.rows(), ds.
dataMatrix.rows());
583 #pragma omp parallel for schedule(static) num_threads(8)
584     for(int i=0;i<adjacencyMatrix.rows();++i)
585     {
586         for(int j=0;j<adjacencyMatrix.cols();++j)
587         {
588             float dist_ij;
589             if(i==j)
590                 continue;
591             else if(distanceMatrix)
592             {
593                 dist_ij = distanceMatrix[i][j];
594             }
595             else
596                 dist_ij = getDisimilarity(ds.dataMatrix, i, j,
normOption, object);
597             adjacencyMatrix(i,j)=exp(-dist_ij*dist_ij/sigmaVec[i]/
sigmaVec[j]);
598         }
599     }
600
601     std::cout << "Finish computing adjacency matrix!" << std::endl;
602 }

```

4.5.3.4 void SpectralClustering::getDegreeMatrix (const Eigen::MatrixXf & *adjacencyMatrix*, Eigen::DiagonalMatrix< float, Dynamic > & *degreeMatrix*) [private]

Definition at line 611 of file SpectralClustering.cpp.

```

612 {
613     degreeMatrix = Eigen::DiagonalMatrix<float,Dynamic>(ds.dataMatrix.rows());
614     Eigen::VectorXf v = VectorXf::Zero(ds.dataMatrix.rows());
615 #pragma omp parallel for schedule(static) num_threads(8)
616     for(int i=0;i<v.size();++i)
617     {
618         float summation = 0;
619         for(int j=0;j<adjacencyMatrix.cols();++j)
620         {
621             summation+=adjacencyMatrix(i,j);
622         }
623         v(i) = summation;
624     }
625
626     degreeMatrix.diagonal() = v;
627
628     std::cout << "Fnish computing degree matrix!" << std::endl;
629 }

```

4.5.3.5 void SpectralClustering::getEigenClustering (const Eigen::MatrixXf & *laplacianMatrix*, const int & *norm*) [private]

Definition at line 669 of file SpectralClustering.cpp.

```

670 {
671     struct timeval start, end;
672     gettimeofday(&start, NULL);
673
674     /* eigen decomposition for Hermite matrix (real and symmetric matrix) */
675     std::cout << "Eigen decomposition starts!..." << std::endl;
676     SelfAdjointEigenSolver<MatrixXf> eigensolver(laplacianMatrix);
677     std::cout << "Eigen decomposition ends!..." << std::endl;
678
679     gettimeofday(&end, NULL);
680     float timeTemp = ((end.tv_sec-start.tv_sec)*1000000u+end.tv_usec-start.tv_usec)/1.e6;
681     activityList.push_back("Eigen decomposition takes: ");
682     timeList.push_back(to_string(timeTemp)+" s");
683
684     const int& eigenRows = presetNumber;

```

```

685     std::cout << "Eigen rows are: " << eigenRows << std::endl;
686     //const int& eigenRows = 5;
687
688     Eigen::MatrixXf eigenVec(eigenRows, ds.dataMatrix.rows());
689
690     const int& Row = laplacianMatrix.rows();
691
692     /* from paper we know it should get largest eigenvalues, and from eigen library we know it's latter */
693     for(int i=Row-1;i>Row-eigenRows-1;--i)
694         eigenVec.row(Row-1-i) = eigensolver.eigenvectors().col(i).transpose();
695     eigenVec.transposeInPlace();
696
697     /* how many elements in each cluster */
698     std::vector<int> storage;
699
700     /* which elements stored in each cluster */
701     std::vector<std::vector<int> > neighborVec;
702
703     /* centroid cluster */
704     Eigen::MatrixXf clusterCenter;
705
706     /* k-means as a post-processing */
707     if(postProcessing==1)
708     {
709         normalizeEigenVec(eigenVec);
710
711         performKMeans(eigenVec,storage,neighborVec);
712
713         setLabel(neighborVec, storage, clusterCenter);
714
715         extractFeatures(storage,neighborVec,clusterCenter);
716     }
717     /* eigenvector rotation */
718     else if(postProcessing==2)
719     {
720         getEigvecRotation(storage,neighborVec,clusterCenter,eigenVec);
721
722         if(neighborVec.empty())
723             return;
724
725         setLabel(neighborVec, storage, clusterCenter);
726
727         if(isOptimal)
728             recordOptimalResult(norm, neighborVec.size());
729         else
730             extractFeatures(storage,neighborVec,clusterCenter);
731     }
732 }

```

4.5.3.6 void SpectralClustering::getEigvecRotation (std::vector< int > & storage, std::vector< std::vector< int > > & neighborVec, Eigen::MatrixXf & clusterCenter, const Eigen::MatrixXf & X) [private]

Definition at line 888 of file SpectralClustering.cpp.

```

890 {
891     mMaxQuality = 0;
892     Eigen::MatrixXf vecRot;
893     Eigen::MatrixXf vecIn = X.block(0,0,X.rows(),2);
894     Evrot *e = NULL;
895
896     struct timeval start, end;
897     gettimeofday(&start, NULL);
898
899     const int& xCols = X.cols();
900
901     std::cout << "Eigenvector rotation starts within " << xCols << " columns..." << std::endl;
902     for (int g=2; g <= xCols; g++)
903     {
904         // make it incremental (used already aligned vectors)
905         std::cout << "column " << g << ":";
906         if( g > 2 )
907         {
908             vecIn.resize(X.rows(),g);
909             vecIn.block(0,0,vecIn.rows(),g-1) = e->getRotatedEigenVectors();
910             vecIn.block(0,g-1,X.rows(),1) = X.block(0,g-1,X.rows(),1);
911             delete e;
912         }
913         //perform the rotation for the current number of dimensions
914         e = new Evrot(vecIn, mMethod);

```

```

915
916     //save max quality
917     if (e->getQuality() > mMaxQuality)
918     {
919         mMaxQuality = e->getQuality();
920     }
921
922     if (isnan(e->getQuality()) || isinf(e->getQuality()))
923     {
924         std::cout << "Meet with nan or inf! Stop! " << std::endl;
925         return;
926     }
927
928     std::cout << " max quality is " << mMaxQuality << ", Evrot has quality " << e->
getQuality() << std::endl;
929     //save cluster data for max cluster or if we're near the max cluster (so prefer more clusters)
930     if ((e->getQuality() > mMaxQuality) || (mMaxQuality - e->
getQuality() <= 0.001))
931     {
932         neighborVec = e->getClusters();
933         vecRot = e->getRotatedEigenVectors();
934     }
935 }
936
937 gettimeofday(&end, NULL);
938 float timeTemp = ((end.tv_sec-start.tv_sec)*1000000u+end.tv_usec-start.tv_usec)/1.e6;
939 activityList.push_back("Eigenvector rotation takes: ");
940 timeList.push_back(to_string(timeTemp)+" s");
941
942 if(neighborVec.empty())
943     return;
944
945 clusterCenter = Eigen::MatrixXf::Zero(neighborVec.size(),vecRot.cols());
946 storage = std::vector<int>(neighborVec.size());
947
948 #pragma omp parallel for schedule(static) num_threads(8)
949 for (unsigned int i=0; i < neighborVec.size(); i++)
950 {
951     storage[i] = neighborVec[i].size();
952     for (unsigned int j=0; j < neighborVec[i].size(); j++)
953     {
954         //sum points within cluster
955         clusterCenter.row(i) += vecRot.row(neighborVec[i][j]);
956     }
957 }
958
959 #pragma omp parallel for schedule(static) num_threads(8)
960 for (unsigned int i=0; i < neighborVec.size(); i++) {
961     //find average point within cluster
962     clusterCenter.row(i) = clusterCenter.row(i) / neighborVec[i].size();
963 }
964
965 numberOfClusters = neighborVec.size();
966 }

```

4.5.3.7 void SpectralClustering::getEntropyRatio (const std::vector< int > & storage, float & EntropyRatio) [private]

Definition at line 561 of file SpectralClustering.cpp.

```

562 {
563     EntropyRatio = 0;
564     const int& Row = ds.dataMatrix.rows();
565     for (int i = 0; i < storage.size(); ++i)
566     {
567         float ratio = float(storage[i])/float(Row);
568         EntropyRatio-=ratio*log2f(ratio);
569     }
570     EntropyRatio/=log2f(storage.size());
571 }

```

4.5.3.8 void SpectralClustering::getLaplacianMatrix (const Eigen::MatrixXf & adjacencyMatrix, Eigen::DiagonalMatrix< float, Dynamic > & degreeMatrix, Eigen::MatrixXf & laplacianMatrix) [private]

Definition at line 639 of file SpectralClustering.cpp.


```

642 {
643     switch(LaplacianOption)
644     {
645     default:
646     case 1:
647         /* L = D^(-1)A */
648         getMatrixPow(degreeMatrix, -1.0);
649         laplacianMatrix=degreeMatrix*adjacencyMatrix;
650         break;
651     case 2:
652         Eigen::MatrixXf dMatrix = Eigen::MatrixXf(adjacencyMatrix.rows(),adjacencyMatrix.cols());
653         const Eigen::VectorXf& m_v = degreeMatrix.diagonal();
654         for(int i=0;i<dMatrix.rows();++i)
655             dMatrix(i,i) = m_v(i);
656         laplacianMatrix = dMatrix-adjacencyMatrix;
657         break;
658     }
659 }
660 }

```

4.5.3.9 void SpectralClustering::getParameterUserInput () [private]

Definition at line 1015 of file SpectralClustering.cpp.

```

1016 {
1017     std::cout << "It is a pathline data set? 1.Yes, 0.No." << std::endl;
1018     int pathlineOption;
1019     std::cin >> pathlineOption;
1020     assert(pathlineOption==1||pathlineOption==0);
1021     isPathlines = (pathlineOption==1);
1022
1023     int sampleOption;
1024
1025     if(isPathlines)
1026         sampleOption = 1;
1027     else
1028     {
1029         std::cout << "choose a sampling method for the dataset?" << std::endl
1030             << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
1031         std::cin >> sampleOption;
1032     }
1033     assert(sampleOption==1||sampleOption==2);
1034
1035     if(sampleOption==1)
1036         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
dimension,ds.maxElements);
1037     else if(sampleOption==2)
1038         IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.
dimension,ds.maxElements);
1039     else if(sampleOption==3)
1040         IOHandler::uniformArcSampling(ds.dataMatrix,ds.dataVec,
ds.dimension,ds.maxElements);
1041
1042     group = std::vector<int>(ds.dataMatrix.rows());
1043
1044     /* the default value for streamline clustering is 2 normalized Laplacian */
1045     std::cout << "-----" << std::endl;
1046     std::cout << "Laplacian option: 1.Normalized Laplacian, 2.Unsymmetric Laplacian" << std::endl;
1047     std::cout << "..And in streamline clustering people tend to choose 1.Normalized Laplacian!-----"
<< std::endl;
1048     std::cin >> LaplacianOption;
1049     assert(LaplacianOption==1||LaplacianOption==2);
1050
1051
1052     int sortedOption;
1053     std::cout << "Please choose whether local scaling by sorted distance: 1. yes, 2. no: " << std::endl;
1054     std::cin >> sortedOption;
1055     assert(sortedOption==1||sortedOption==2);
1056     if(sortedOption==1)
1057         sortedOption = true;
1058     else if(sortedOption==2)
1059         sortedOption = false;
1060
1061     std::cout << "-----" << std::endl;
1062     std::cout << "Input a desired cluster number among [1, " << ds.dataMatrix.rows() << "]: ";
1063     std::cin >> presetNumber;
1064     assert(presetNumber>1 && presetNumber<ds.dataMatrix.rows()/10);
1065
1066     std::cout << "-----" << std::endl;
1067     std::cout << "Input a post-processing method: 1.k-means, 2.eigenvector rotation: " << std::endl;

```

```

1068     std::cin >> postProcessing;
1069     assert(postProcessing==1||postProcessing==2);
1070
1071     if(postProcessing==2)
1072     {
1073         std::cout << "-----" << std::endl;
1074         std::cout << "Please input derivative method: 1.numerical derivative, 2.true derivative." <<
std::endl;
1075         std::cin >> mMethod;
1076         assert(mMethod==1 || mMethod==2);
1077     }
1078
1079     std::cout << "Please choose cluster number method, 0.user input, 1.read clustering: " << std::endl;
1080     int clusterInput;
1081     std::cin >> clusterInput;
1082     assert(clusterInput==0 || clusterInput==1);
1083     readCluster = (clusterInput==1);
1084
1085 }

```

4.5.3.10 void SpectralClustering::getSigmaList() [private]

Definition at line 498 of file SpectralClustering.cpp.

```

499 {
500     const int& Row = ds.dataMatrix.rows();
501     sigmaVec = std::vector<float>(Row);
502
503     if(isDistSorted)
504     {
505         /* get SCALING-th smallest dist */
506         #pragma omp parallel for schedule(static) num_threads(8)
507         for(int i=0;i<Row;++i)
508         {
509             /* instead we implement a n*logk priority_queue method for finding k-th smallest element */
510             std::priority_queue<float> limitQueue;
511             float tempDist;
512             for(int j=0;j<Row;++j)
513             {
514                 if(i==j)
515                     continue;
516                 if(distanceMatrix)
517                     tempDist = distanceMatrix[i][j];
518                 else
519                     tempDist = getDisimilarity(ds.dataMatrix, i, j,
normOption, object);
520                 // element is even larger than the biggest
521                 limitQueue.push(tempDist);
522                 if(limitQueue.size()>SCALING)
523                     limitQueue.pop();
524             }
525
526             sigmaVec[i] = limitQueue.top();
527         }
528     }
529     else
530     {
531         /* directly by index since in both papers only mention i-th neighboring point */
532         #pragma omp parallel for schedule(static) num_threads(8)
533         for(int i=0;i<Row;++i)
534         {
535             if(i<SCALING)
536             {
537                 if(distanceMatrix)
538                     sigmaVec[i]=distanceMatrix[i][SCALING];
539                 else
540                     sigmaVec[i]=getDisimilarity(ds.dataMatrix, i,
SCALING, normOption, object);
541             }
542             else
543             {
544                 if(distanceMatrix)
545                     sigmaVec[i]=distanceMatrix[i][SCALING-1];
546                 else
547                     sigmaVec[i]=getDisimilarity(ds.dataMatrix, i,
SCALING-1, normOption, object);
548             }
549         }
550     }
551     std::cout << "Finish local scaling..." << std::endl;
552 }

```

4.5.3.11 void SpectralClustering::normalizeEigenvec (Eigen::MatrixXf & *eigenVec*) [private]

Definition at line 755 of file SpectralClustering.cpp.

```

756 {
757     const int& rows = eigenVec.rows();
758     #pragma omp parallel for schedule(static) num_threads(8)
759     for(int i=0;i<rows;++i)
760     {
761         eigenVec.row(i)/=eigenVec.row(i).norm();
762     }
763 }
```

4.5.3.12 void SpectralClustering::performClustering ()

Definition at line 59 of file SpectralClustering.cpp.

```

60 {
61     //distance metric type
62     /* 0: Euclidean Norm
63        1: Fraction Distance Metric
64        2: piece-wise angle average
65        3: Bhattacharyya metric for rotation
66        4: average rotation
67        5: signed-angle intersection
68        6: normal-direction multivariate distribution
69        7: Bhattacharyya metric with angle to a fixed direction
70        8: Piece-wise angle average \times standard deviation
71        9: normal-direction multivariate un-normalized distribution
72        10: x*y/|x||y| borrowed from machine learning
73        11: cosine similarity
74        12: Mean-of-closest point distance (MCP)
75        13: Hausdorff distance min_max(x_i,y_i)
76        14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
77        15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
78        16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
79        17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
S0097849318300128
80        for pathlines only
81     */
82     if(postProcessing==2)
83     {
84         std::cout << "Find optimal activated? 0. No, 1. Yes: " << std::endl;
85         int optimalOption;
86         std::cin >> optimalOption;
87         assert(optimalOption==0 || optimalOption==1);
88         isOptimal = (optimalOption==1);
89     }
90     std::cout << "Please input the preset number of clusters in [2, " << ds.
dataVec.size() << "]: " << std::endl;
91     std::cin >> presetNumber;
92     assert(presetNumber>=2 && presetNumber<=ds.dataVec.size());
93
94     /* record initial number of clusters of user input */
95     recordPreset(presetNumber);
96
97     readCluster = false;
98 }
99 else if(postProcessing==1)
100 {
101     std::cout << "Please choose cluster number method, 0.user input, 1.read clustering: " << std::endl;
102     int clusterInput;
103     std::cin >> clusterInput;
104     assert(clusterInput==0 || clusterInput==1);
105     readCluster = (clusterInput==1);
106 }
107
108 std::unordered_map<int,int> clusterMap;
109 if(readCluster)
110 {
111     IOHandler::readClusteringNumber(clusterMap, "cluster_number");
112 }
113
114
115 for(int i=0;i<=17;++i)
116 {
```

```

117         if(isPathlines)
118         {
119             /* don't want to deal with many too naive metrics */
120             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15 && i!=17)
121                 continue;
122         }
123         else
124         {
125             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
126                 continue;
127         }
128
129         if(postProcessing==1)
130         {
131             if(readCluster)
132                 presetNumber = clusterMap[i];
133             else
134             {
135                 std::cout << "Please input the preset number of clusters for norm " << i << " among [2, "
136                     << ds.dataVec.size() << "]: " << std::endl;
137                 std::cin >> presetNumber;
138             }
139             assert(presetNumber>=2 && presetNumber<=ds.dataVec.size());
140         }
141
142         std::cout << "-----" << std::endl;
143         std::cout << "Experiment on norm " << i << " starts!-----" << std::endl;
144
145         activityList.clear();
146         timeList.clear();
147
148         activityList.push_back("Preset numOfClusters for norm "+to_string(i) + " is: ");
149         timeList.push_back(to_string(presetNumber));
150
151         struct timeval start, end;
152         double timeTemp;
153         gettimeofday(&start, NULL);
154
155         clusterByNorm(i);
156
157         gettimeofday(&end, NULL);
158         timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
159             + end.tv_usec - start.tv_usec) / 1.e6;
160         activityList.push_back("SC for "+to_string(i) + " takes: ");
161         timeList.push_back(to_string(timeTemp)+"s");
162
163         std::cout << std::endl;
164     }
165 }

```

4.5.3.13 void SpectralClustering::performKMeans (const Eigen::MatrixXf & *eigenVec*, std::vector< int > & *storage*, std::vector< std::vector< int > > & *neighborVec*) [private]

Definition at line 773 of file SpectralClustering.cpp.

```

776 {
777
778     const int& Row = eigenVec.rows();
779     const int& Column = eigenVec.cols();
780
781     float moving=1000, tempMoving, before;
782
783     numberOfClusters = presetNumber;
784
785     storage = std::vector<int>(numberOfClusters);
786
787     /* centerTemp is temporary term for storing centroid position, clusterCenter is permanent */
788     MatrixXf centerTemp, clusterCenter;
789
790     /* chosen from sample for initialization of k-means */
791     Initialization::generateFromSamples(clusterCenter, Column, eigenVec,
792         numberOfClusters);
793
794     int tag = 0;
795
796     neighborVec=std::vector< std::vector<int> >(numberOfClusters);
797
798     float PCA_KMeans_delta, KMeans_delta;
799

```

```

799     std::cout << "...k-means started!" << std::endl;
800
801     struct timeval start, end;
802     gettimeofday(&start, NULL);
803
804     do
805     {
806         before = moving;
807         /* preset cluster number recorder */
808         std::fill(storage.begin(), storage.end(), 0);
809
810         centerTemp = MatrixXf::Zero(numberOfClusters, Column);
811
812         #pragma omp parallel for schedule(static) num_threads(8)
813         for (int i = 0; i < numberOfClusters; ++i)
814         {
815             neighborVec[i].clear();
816         }
817
818         #pragma omp parallel num_threads(8)
819         {
820             #pragma omp for nowait
821             for (int i = 0; i < Row; ++i)
822             {
823                 float dist = FLT_MAX;
824                 float temp;
825                 int clusTemp;
826                 for (int j = 0; j < numberOfClusters; ++j)
827                 {
828                     temp = (eigenVec.row(i)-clusterCenter.row(j)).norm();
829                     if(temp<dist)
830                     {
831                         dist = temp;
832                         clusTemp = j;
833                     }
834                 }
835
836                 #pragma omp critical
837                 {
838                     storage[clusTemp]++;
839                     neighborVec[clusTemp].push_back(i);
840                     group[i] = clusTemp;
841                     centerTemp.row(clusTemp)+=eigenVec.row(i);
842                 }
843             }
844         }
845
846         moving = FLT_MIN;
847
848         #pragma omp parallel for reduction(max:moving) num_threads(8)
849         for (int i = 0; i < numberOfClusters; ++i)
850         {
851             if(storage[i]>0)
852             {
853                 centerTemp.row(i)/=storage[i];
854                 tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
855                 clusterCenter.row(i) = centerTemp.row(i);
856                 if(moving<tempMoving)
857                     moving = tempMoving;
858             }
859         }
860         std::cout << "K-means iteration " << ++tag << " completed, and moving is "
861         << moving << "!" << std::endl;
862     }while(abs(moving-before)/before >= 1.0e-3 && tag < 50 && moving>0.01);
863
864     gettimeofday(&end, NULL);
865     float timeTemp = ((end.tv_sec-start.tv_sec)*1000000u+end.tv_usec-start.tv_usec)/1.e6;
866     activityList.push_back("K-means takes: ");
867     timeList.push_back(to_string(timeTemp)+" s");
868
869     for(auto iter=storage.begin(); iter!=storage.end(); )
870     {
871         if(*iter==0)
872             storage.erase(iter);
873         else
874             ++iter;
875     }
876     numberOfClusters = storage.size();
877 }

```

4.5.3.14 void SpectralClustering::recordOptimalResult (const int & normOption, const int & clusNum) [private]

Definition at line 1113 of file SpectralClustering.cpp.

```

1114 {
1115     std::ofstream readme("../dataset/optimal.txt",ios::out | ios::app);
1116     if(!readme)
1117     {
1118         std::cout << "Error creating readme!" << std::endl;
1119         exit(1);
1120     }
1121     readme << "Optimal number of cluster for norm " << normOption << " with sc eigen-rotation
minimization is "
1122         << clusNum << std::endl;
1123     readme << std::endl;
1124     readme.close();
1125 }

```

4.5.3.15 void SpectralClustering::recordPreset (const int & *number*) [private]

Definition at line 1093 of file SpectralClustering.cpp.

```

1094 {
1095     std::ofstream readme("../dataset/optimal.txt",ios::out | ios::app);
1096     if(!readme)
1097     {
1098         std::cout << "Error creating readme!" << std::endl;
1099         exit(1);
1100     }
1101     readme << "Preset cluster number is: " << number << std::endl;
1102     readme << std::endl;
1103     readme.close();
1104 }

```

4.5.3.16 void SpectralClustering::setDataset (const int & *argc*, char ** *argv*) [private]

Definition at line 473 of file SpectralClustering.cpp.

```

474 {
475     if(argc!=3)
476     {
477         std::cout << "Input argument should have 3!" << endl
478             << ". /cluster inputFile_name(in dataset folder) "
479             << "data_dimension(3)" << endl;
480         exit(1);
481     }
482     ds.strName = string("../dataset/") + string(argv[1]);
483     ds.dataName = string(argv[1]);
484     ds.dimension = atoi(argv[2]);
485
486     IOHandler::readFile(ds.strName,ds.dataVec,ds.vertexCount,
ds.dimension,ds.maxElements);
487
488     ds.fullName = ds.strName+"_full.vtk";
489     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
vertexCount, ds.dimension);
490
491     SCALING = 0.05*ds.dataVec.size();
492 }

```

4.5.3.17 void SpectralClustering::setLabel (vector< vector< int > > & *neighborVec*, vector< int > & *storage*, Eigen::MatrixXf & *centroid*) [private]

Definition at line 269 of file SpectralClustering.cpp.

```

270 {
271     std::vector<Ensemble> nodeVec(storage.size());
272     std::cout << "Cluster label setting begins with " << numberOfClusters << " clusters..."
    << std::endl;
273 #pragma omp parallel for schedule(static) num_threads(8)
274     for(int i=0;i<nodeVec.size();++i)
275     {
276         nodeVec[i].size = storage[i];
277         nodeVec[i].element = neighborVec[i];
278     }
279
280     /* sort group index by size of elements contained inside */
281     std::sort(nodeVec.begin(), nodeVec.end(), [](const Ensemble& first, const
    Ensemble& second)
282     {return first.size<second.size|| (first.size==second.size&&first.
    element[0]<second.element[0]);});
283
284     neighborVec = std::vector<std::vector<int> >(nodeVec.size());
285     storage = std::vector<int>(nodeVec.size());
286     centroid = Eigen::MatrixXf(nodeVec.size(), ds.dataMatrix.cols());
287
288 #pragma omp parallel for schedule(static) num_threads(8)
289     for(int i=0;i<nodeVec.size();++i)
290     {
291         neighborVec[i] = nodeVec[i].element;
292         storage[i] = nodeVec[i].size;
293         Eigen::VectorXf tempVec = Eigen::VectorXf::Zero(ds.dataMatrix.cols());
294         for(int j=0;j<storage[i];++j)
295         {
296             tempVec+=ds.dataMatrix.row(i).transpose();
297             /* don't forget to re-compute the group tag */
298             group[neighborVec[i][j]]=i;
299         }
300         centroid.row(i) = tempVec/storage[i];
301     }
302
303     std::cout << "Cluster label setting ends..." << std::endl;
304 }

```

4.5.3.18 void SpectralClustering::setParameterAutomatic (const Para & p) [private]

Definition at line 974 of file SpectralClustering.cpp.

```

975 {
976     std::cout << "It is a pathline data set? 1.Yes, 0.No." << std::endl;
977     int pathlineOption;
978     std::cin >> pathlineOption;
979     assert(pathlineOption==1||pathlineOption==0);
980     isPathlines = (pathlineOption==1);
981
982     if(isPathlines)
983     {
984         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
985     }
986     else
987     {
988         if(p.sampled==1)
989             IOHandler::expandArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
990         else if(p.sampled==2)
991             IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
992         else if(p.sampled==3)
993             IOHandler::uniformArcSampling(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
994     }
995
996     group = std::vector<int>(ds.dataMatrix.rows());
997
998     /* the default value for streamline clustering is 2 normalized Laplacian */
999     LaplacianOption = p.LaplacianOption;
1000
1001     isDistSorted = p.isDistSorted;
1002
1003     postProcessing = p.postProcessing;
1004
1005     mMethod = p.mMethod;
1006
1007     extractOption = p.extractOption;
1008
1009 }

```

4.5.4 Member Data Documentation

4.5.4.1 `std::vector<string> SpectralClustering::activityList` [private]

Definition at line 129 of file SpectralClustering.h.

4.5.4.2 `std::vector<float> SpectralClustering::distRange` [private]

Definition at line 154 of file SpectralClustering.h.

4.5.4.3 `DataSet SpectralClustering::ds` [private]

Definition at line 139 of file SpectralClustering.h.

4.5.4.4 `int SpectralClustering::extractOption = -1` [private]

Definition at line 179 of file SpectralClustering.h.

4.5.4.5 `std::vector<int> SpectralClustering::group` [private]

Definition at line 124 of file SpectralClustering.h.

4.5.4.6 `int SpectralClustering::initializationOption = -1` [private]

Definition at line 149 of file SpectralClustering.h.

4.5.4.7 `bool SpectralClustering::isDistSorted = -1` [private]

Definition at line 169 of file SpectralClustering.h.

4.5.4.8 `bool SpectralClustering::isOptimal` [private]

Definition at line 189 of file SpectralClustering.h.

4.5.4.9 `bool SpectralClustering::isPathlines` [private]

Definition at line 204 of file SpectralClustering.h.

4.5.4.10 `int SpectralClustering::LaplacianOption = -1` [private]

Definition at line 164 of file SpectralClustering.h.

4.5.4.11 `float SpectralClustering::mMaxQuality = 0` [private]

Definition at line 355 of file SpectralClustering.h.

4.5.4.12 `int SpectralClustering::mMethod = -1` [private]

Definition at line 360 of file SpectralClustering.h.

4.5.4.13 `int SpectralClustering::normOption = -1` [private]

Definition at line 119 of file SpectralClustering.h.

4.5.4.14 `int SpectralClustering::numberOfClusters = -1` [private]

Definition at line 144 of file SpectralClustering.h.

4.5.4.15 `MetricPreparation SpectralClustering::object` [private]

Definition at line 114 of file SpectralClustering.h.

4.5.4.16 `int SpectralClustering::postProcessing = -1` [private]

Definition at line 174 of file SpectralClustering.h.

4.5.4.17 `int SpectralClustering::presetNumber` [private]

Definition at line 194 of file SpectralClustering.h.

4.5.4.18 `bool SpectralClustering::readCluster` [private]

Definition at line 199 of file SpectralClustering.h.

4.5.4.19 `int SpectralClustering::SCALING` [private]

Definition at line 184 of file SpectralClustering.h.

4.5.4.20 `std::vector<float> SpectralClustering::sigmaVec` [private]

Definition at line 159 of file SpectralClustering.h.

4.5.4.21 `std::vector<string> SpectralClustering::timeList` [private]

Definition at line 134 of file SpectralClustering.h.

The documentation for this class was generated from the following files:

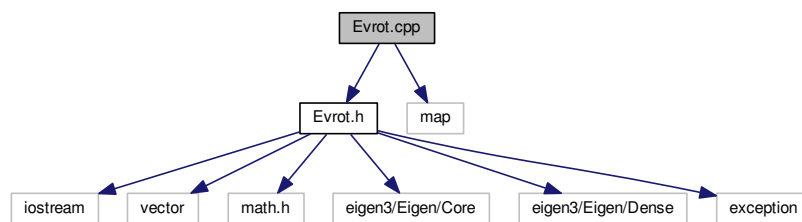
- [SpectralClustering.h](#)
- [SpectralClustering.cpp](#)

Chapter 5

File Documentation

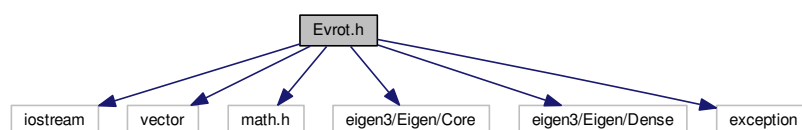
5.1 Evrot.cpp File Reference

```
#include "Evrot.h"  
#include <map>  
Include dependency graph for Evrot.cpp:
```

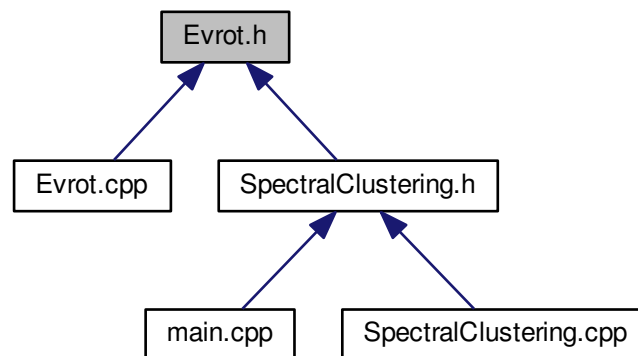


5.2 Evrot.h File Reference

```
#include <iostream>  
#include <vector>  
#include <math.h>  
#include <eigen3/Eigen/Core>  
#include <eigen3/Eigen/Dense>  
#include <exception>  
Include dependency graph for Evrot.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Evrot](#)

Macros

- `#define EPS 2.2204e-8`

5.2.1 Macro Definition Documentation

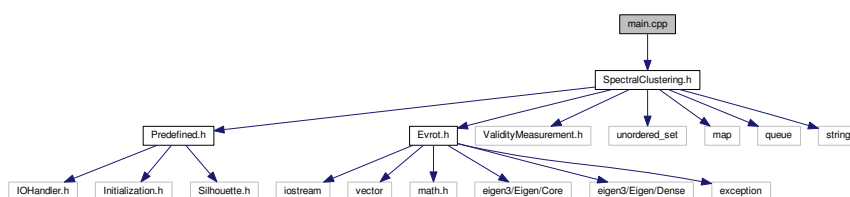
5.2.1.1 `#define EPS 2.2204e-8`

Definition at line 28 of file `Evrot.h`.

5.3 main.cpp File Reference

```
#include "SpectralClustering.h"
```

Include dependency graph for `main.cpp`:



Functions

- void `setPara` (`Para` &p)
- int `main` (int argc, char **argv)

5.3.1 Function Documentation

5.3.1.1 int main (int argc, char ** argv)

Definition at line 18 of file main.cpp.

```
19 {
20     Para p;
21
22     setPara(p);
23
24     /* enable automatic option */
25     bool automatic = true;
26
27     SpectralClustering spectClus(argc, argv, p, automatic);
28
29     spectClus.performClustering();
30
31     return 0;
32 }
```

5.3.1.2 void setPara (Para & p)

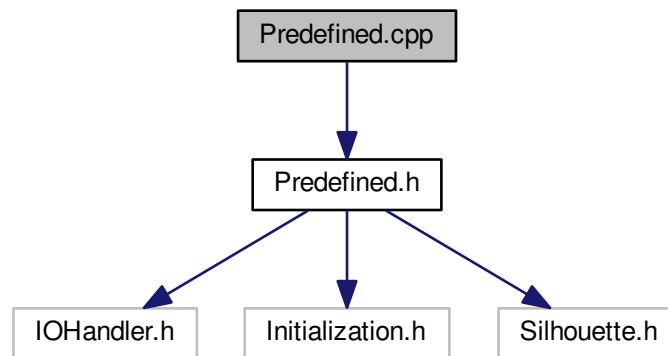
Definition at line 40 of file main.cpp.

```
41 {
42     /* 1.directly filling with last vertex; 2. uniform sampling, 3. equal-arc sampling */
43     p.sampled = 2;
44
45     /* Laplacian option: 1.Normalized Laplacian, 2.Unsymmetric Laplacian */
46     p.LaplacianOption = 1;
47
48     /* local scaling by sorted distance: true, false */
49     p.isDistSorted = true;
50
51     /* post-processing method: 1.k-means, 2.eigenvector rotation*/
52     std::cout << "Input the post-processing: 1.k-means, 2.eigenvector rotation: " << std::endl;
53     std::cin >> p.postProcessing;
54     assert(p.postProcessing==1 || p.postProcessing==2);
55
56     /* derivative method for eigen rotation: 1.numerical derivative, 2.true derivative */
57     p.mMethod = 2;
58
59     /* extraction option, 1. centroid, closest and furthest, 2. median, 3. statistical representation */
60     p.extractOption = 1;
61 }
```

5.4 Predefined.cpp File Reference

```
#include "Predefined.h"
```

Include dependency graph for Predefined.cpp:



Functions

- `template<class T >`
`void deleteVecElements (std::vector< T > &original, const T &first, const T &second)`

5.4.1 Function Documentation

5.4.1.1 `template<class T > void deleteVecElements (std::vector< T > & original, const T & first, const T & second)`

Definition at line 19 of file Predefined.cpp.

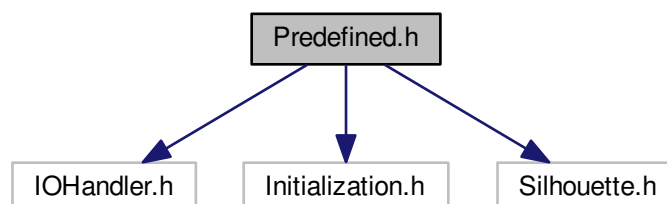
```

20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result (size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert (tag==size-2);
33     original = result;
34 }
```

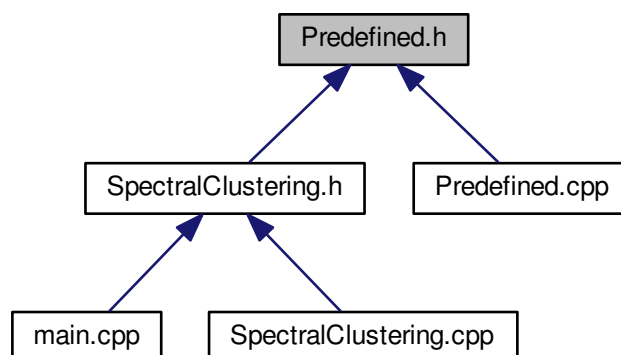
5.5 Predefined.h File Reference

```
#include "IOHandler.h"  
#include "Initialization.h"  
#include "Silhouette.h"
```

Include dependency graph for Predefined.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [DataSet](#)
- struct [Ensemble](#)

Functions

- template<class T >
void [deleteVecElements](#) (std::vector< T > &original, const T &first, const T &second)

5.5.1 Function Documentation

5.5.1.1 `template<class T> void deleteVecElements (std::vector< T> & original, const T & first, const T & second)`

Definition at line 19 of file `Predefined.cpp`.

```

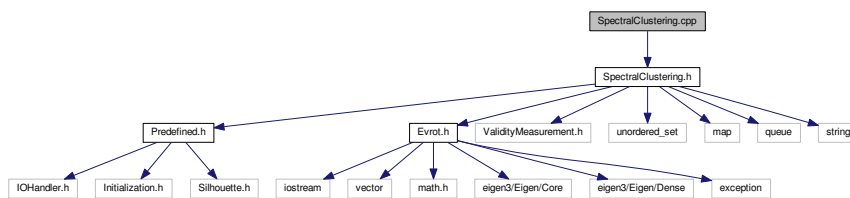
20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result(size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert(tag==size-2);
33     original = result;
34 }
```

5.6 README.md File Reference

5.7 SpectralClustering.cpp File Reference

```
#include "SpectralClustering.h"
```

Include dependency graph for `SpectralClustering.cpp`:



Functions

- void `getMatrixPow` (`Eigen::DiagonalMatrix< float, Dynamic > &matrix`, `const float &powNumber`)

5.7.1 Function Documentation

5.7.1.1 `void getMatrixPow (Eigen::DiagonalMatrix< float, Dynamic > & matrix, const float & powNumber)`

Definition at line 741 of file `SpectralClustering.cpp`.

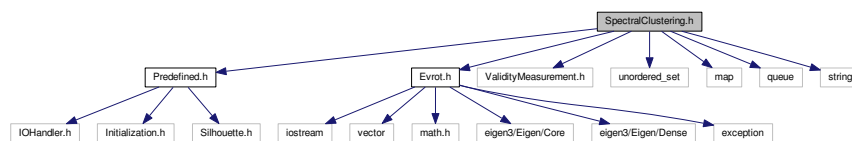
```

742 {
743     Eigen::VectorXf& m_v = matrix.diagonal();
744     #pragma omp parallel for schedule(static) num_threads(8)
745     for(int i=0;i<m_v.size();++i)
746         m_v(i) = pow(m_v(i), powNumber);
747 }
```

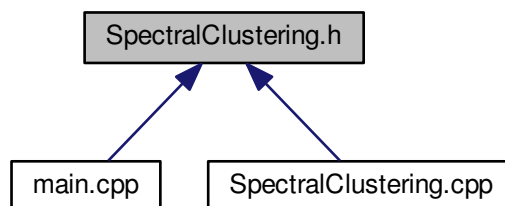

5.8 SpectralClustering.h File Reference

```
#include "Predefined.h"
#include "Evrot.h"
#include "ValidityMeasurement.h"
#include <unordered_set>
#include <map>
#include <queue>
#include <string>
```

Include dependency graph for SpectralClustering.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Para](#)
- class [SpectralClustering](#)

Macros

- `#define` [GradientStep](#) 0.3

Functions

- void [getMatrixPow](#) (Eigen::DiagonalMatrix< float, Dynamic > &matrix, const float &powNumber)

5.8.1 Macro Definition Documentation

5.8.1.1 `#define GradientStep 0.3`

Definition at line 31 of file SpectralClustering.h.

5.8.2 Function Documentation

5.8.2.1 `void getMatrixPow (Eigen::DiagonalMatrix< float, Dynamic > & matrix, const float & powNumber)`

Definition at line 741 of file SpectralClustering.cpp.

```
742 {  
743     Eigen::VectorXf& m_v = matrix.diagonal();  
744     #pragma omp parallel for schedule(static) num_threads(8)  
745     for(int i=0; i<m_v.size(); ++i)  
746         m_v(i) = pow(m_v(i), powNumber);  
747 }
```

Index

- ~Evrot
 - Evrot, [10](#)
- ~SpectralClustering
 - SpectralClustering, [18](#)
- activityList
 - SpectralClustering, [32](#)
- build_Uab
 - Evrot, [10](#)
- cluster_assign
 - Evrot, [10](#)
- clusterByNorm
 - SpectralClustering, [18](#)
- dataMatrix
 - DataSet, [7](#)
- dataName
 - DataSet, [7](#)
- DataSet, [7](#)
 - dataMatrix, [7](#)
 - dataName, [7](#)
 - dataVec, [7](#)
 - dimension, [8](#)
 - fullName, [8](#)
 - maxElements, [8](#)
 - strName, [8](#)
 - vertexCount, [8](#)
- dataVec
 - DataSet, [7](#)
- deleteVecElements
 - Predefined.cpp, [38](#)
 - Predefined.h, [40](#)
- dimension
 - DataSet, [8](#)
- distRange
 - SpectralClustering, [32](#)
- ds
 - SpectralClustering, [32](#)
- EPS
 - Evrot.h, [36](#)
- element
 - Ensemble, [9](#)
- Ensemble, [8](#)
 - element, [9](#)
 - size, [9](#)
- evqual
 - Evrot, [11](#)
- evqualitygrad
 - Evrot, [11](#)
- Evrot, [11](#)
- Evrot, [9](#)
 - ~Evrot, [10](#)
 - build_Uab, [10](#)
 - cluster_assign, [10](#)
 - evqual, [11](#)
 - evqualitygrad, [11](#)
 - Evrot, [10](#)
 - evrot, [12](#)
 - getClusters, [13](#)
 - getQuality, [13](#)
 - getRotatedEigenVectors, [13](#)
 - gradU, [13](#)
 - ik, [14](#)
 - jk, [14](#)
 - mClusters, [14](#)
 - mMethod, [14](#)
 - mNumAngles, [14](#)
 - mNumData, [14](#)
 - mNumDims, [15](#)
 - mQuality, [15](#)
 - mXrot, [15](#)
 - mX, [15](#)
 - rotate_givens, [14](#)
- evrot
 - Evrot, [12](#)
- Evrot.cpp, [35](#)
- Evrot.h, [35](#)
 - EPS, [36](#)
- extractFeatures
 - SpectralClustering, [20](#)
- extractOption
 - Para, [16](#)
 - SpectralClustering, [32](#)
- fullName
 - DataSet, [8](#)
- getAdjacencyMatrix
 - SpectralClustering, [21](#)
- getClusters
 - Evrot, [13](#)
- getDegreeMatrix
 - SpectralClustering, [22](#)
- getEigenClustering
 - SpectralClustering, [22](#)
- getEigvecRotation
 - SpectralClustering, [23](#)
- getEntropyRatio
 - SpectralClustering, [24](#)

- getLaplacianMatrix
 - SpectralClustering, 24
- getMatrixPow
 - SpectralClustering.cpp, 40
 - SpectralClustering.h, 42
- getParameterUserInput
 - SpectralClustering, 25
- getQuality
 - Evrot, 13
- getRotatedEigenVectors
 - Evrot, 13
- getSigmaList
 - SpectralClustering, 26
- GradientStep
 - SpectralClustering.h, 42
- gradU
 - Evrot, 13
- group
 - SpectralClustering, 32
- ik
 - Evrot, 14
- initializationOption
 - SpectralClustering, 32
- isDistSorted
 - Para, 16
 - SpectralClustering, 32
- isOptimal
 - SpectralClustering, 32
- isPathlines
 - SpectralClustering, 32
- jk
 - Evrot, 14
- LaplacianOption
 - Para, 16
 - SpectralClustering, 32
- mClusters
 - Evrot, 14
- mMaxQuality
 - SpectralClustering, 32
- mMethod
 - Evrot, 14
 - Para, 16
 - SpectralClustering, 33
- mNumAngles
 - Evrot, 14
- mNumData
 - Evrot, 14
- mNumDims
 - Evrot, 15
- mQuality
 - Evrot, 15
- mXrot
 - Evrot, 15
- main
 - main.cpp, 37
- main.cpp, 36
 - main, 37
 - setPara, 37
- maxElements
 - DataSet, 8
- mX
 - Evrot, 15
- normOption
 - SpectralClustering, 33
- normalizeEigenvec
 - SpectralClustering, 26
- numberOfClusters
 - SpectralClustering, 33
- object
 - SpectralClustering, 33
- Para, 15
 - extractOption, 16
 - isDistSorted, 16
 - LaplacianOption, 16
 - mMethod, 16
 - postProcessing, 16
 - sampled, 16
- performClustering
 - SpectralClustering, 27
- performKMeans
 - SpectralClustering, 28
- postProcessing
 - Para, 16
 - SpectralClustering, 33
- Predefined.cpp, 38
 - deleteVecElements, 38
- Predefined.h, 39
 - deleteVecElements, 40
- presetNumber
 - SpectralClustering, 33
- README.md, 40
- readCluster
 - SpectralClustering, 33
- recordOptimalResult
 - SpectralClustering, 29
- recordPreset
 - SpectralClustering, 30
- rotate_givens
 - Evrot, 14
- SCALING
 - SpectralClustering, 33
- sampled
 - Para, 16
- setDataset
 - SpectralClustering, 30
- setLabel
 - SpectralClustering, 30
- setPara
 - main.cpp, 37

- setParameterAutomatic
 - SpectralClustering, [31](#)
- sigmaVec
 - SpectralClustering, [33](#)
- size
 - Ensemble, [9](#)
- SpectralClustering, [16](#)
 - ~SpectralClustering, [18](#)
 - activityList, [32](#)
 - clusterByNorm, [18](#)
 - distRange, [32](#)
 - ds, [32](#)
 - extractFeatures, [20](#)
 - extractOption, [32](#)
 - getAdjacencyMatrix, [21](#)
 - getDegreeMatrix, [22](#)
 - getEigenClustering, [22](#)
 - getEigvecRotation, [23](#)
 - getEntropyRatio, [24](#)
 - getLaplacianMatrix, [24](#)
 - getParameterUserInput, [25](#)
 - getSigmaList, [26](#)
 - group, [32](#)
 - initializationOption, [32](#)
 - isDistSorted, [32](#)
 - isOptimal, [32](#)
 - isPathlines, [32](#)
 - LaplacianOption, [32](#)
 - mMaxQuality, [32](#)
 - mMethod, [33](#)
 - normOption, [33](#)
 - normalizeEigenvector, [26](#)
 - numberOfClusters, [33](#)
 - object, [33](#)
 - performClustering, [27](#)
 - performKMeans, [28](#)
 - postProcessing, [33](#)
 - presetNumber, [33](#)
 - readCluster, [33](#)
 - recordOptimalResult, [29](#)
 - recordPreset, [30](#)
 - SCALING, [33](#)
 - setDataset, [30](#)
 - setLabel, [30](#)
 - setParameterAutomatic, [31](#)
 - sigmaVec, [33](#)
 - SpectralClustering, [18](#)
 - timeList, [33](#)
- SpectralClustering.cpp, [40](#)
 - getMatrixPow, [40](#)
- SpectralClustering.h, [41](#)
 - getMatrixPow, [42](#)
 - GradientStep, [42](#)
- strName
 - DataSet, [8](#)
- timeList
 - SpectralClustering, [33](#)
- vertexCount
 - DataSet, [8](#)