Common Folder

The C++ implementation for preparation for the clustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

Common Folder Description

It includes relevant functions

- File I/O operations
- Sampling for streamlines/pathlines
- k-means initialization (from samples, from random coordinates, or k-means++)
- The hierarchical L-method for finding optimal number of clusters
- Different similarity measures for streamlines/pathlines
- The functions to calculate the clustering evaluation metrics, silhouette, the Gamma statics, DB index and normalized validity measurement

Special notice

Distance Matrix

The distance matrix **distanceMatrix** is pre-stored as a 'float∗∗∗' so that every time when calculating the similarity measure between two selected curves, the 'distanceMatrix' will be checked to be NULL or not. If 'distanceMatrix' is NULL, then the similarity measure function will be called otherwise the cached value is called.

MetricPreparation

It is created before calculating the **MetricPreparation** due to the fact that for some similarity measures, e.g., the d_G (2), d_S(14) and d_P(15), either the segmentation on the streamlines/pathlines or the signature histograms should be calculated. In order to avoid repeated calculation of those signatures, we use a cache to pre-calculate the signatures for each line and store them for further pairwise distance value calculation.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 CompareFunc Class Reference

`#include <PreComputing.h>`

Public Member Functions

- bool operator() (const CurvatureObject &first, const CurvatureObject &second)

4.1.1 Detailed Description

Definition at line 62 of file PreComputing.h.

4.1.2 Member Function Documentation

4.1.2.1 bool CompareFunc::operator() (const CurvatureObject & *first*, const CurvatureObject & *second*) [inline]

Definition at line 65 of file PreComputing.h.

```
66     {
67         return first.curvature < second.curvature;
68     }
```

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

- PreComputing.h

4.2 CurvatureObject Struct Reference

`#include <PreComputing.h>`

Public Member Functions

- CurvatureObject (const float &curvature, const int &i)
- CurvatureObject ()

Public Attributes

- float curvature
- int index

4.2.1 Detailed Description

Definition at line 46 of file PreComputing.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CurvatureObject::CurvatureObject (const float & *curvature*, const int & *i*) [inline]

Definition at line 51 of file PreComputing.h.

```
51                                                      : curvature(
      curvature), index(i)
52      {}
```

4.2.2.2 CurvatureObject::CurvatureObject ( ) [inline]

Definition at line 54 of file PreComputing.h.

```
55      {}
```

4.2.3 Member Data Documentation

4.2.3.1 float CurvatureObject::curvature

Definition at line 48 of file PreComputing.h.

4.2.3.2 int CurvatureObject::index

Definition at line 49 of file PreComputing.h.

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

- PreComputing.h

4.3 cyl_bessel_j_integral_rep< value_type > Class Template Reference

`#include <Metric.h>`

Public Member Functions

- cyl_bessel_j_integral_rep (const value_type &a, const value_type &b, const value_type &c)
- value_type operator() (const value_type &t) const

Private Attributes

- const value_type a
- const value_type b
- const value_type c

4.3.1 Detailed Description

template<typename value_type>
class cyl_bessel_j_integral_rep< value_type >

Definition at line 348 of file Metric.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<typename value_type > cyl_bessel_j_integral_rep< value_type >::cyl_bessel_j_integral_rep (const value_type & *a*, const value_type & *b*, const value_type & *c*) [inline]

Definition at line 351 of file Metric.h.

```
351                                                                                 :
      a(a), b(b), c(c)
352     {}
```

4.3.3 Member Function Documentation

4.3.3.1 template<typename value_type > value_type cyl_bessel_j_integral_rep< value_type >::operator() (const value_type & *t*) const [inline]

Definition at line 354 of file Metric.h.

```
355     {
356         // pi * Jn(x) = Int_0^pi [cos(x * sin(t) - n*t) dt]
357         // return cos(x * sin(t) - (n * t));
358         return sqrt(a+2.0*b*t+c*t*t);
359     }
```

4.3.4 Member Data Documentation

4.3.4.1 template<typename value_type > const value_type cyl_bessel_j_integral_rep< value_type >::a [private]

Definition at line 362 of file Metric.h.

4.3.4.2 template<typename value_type > const value_type cyl_bessel_j_integral_rep< value_type >::b [private]

Definition at line 362 of file Metric.h.

4.3.4.3 template<typename value_type > const value_type cyl_bessel_j_integral_rep< value_type >::c [private]

Definition at line 362 of file Metric.h.

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

- Metric.h

4.4 DetermClusterNum Class Reference

`#include <DetermClusterNum.h>`

Public Member Functions

- DetermClusterNum ()
- virtual ~DetermClusterNum ()
- const int & getFinalNumOfClusters ()
- void iterativeRefinement (std::map< int, float > &eval_graph)
- void recordLMethodResult (const int &normOption)

Private Member Functions

- const int LMethod (const std::map< int, float > &eval_graph, const int &cutoff)
- void removeExtreme (std::map< int, float > &eval_graph)

Private Attributes

- int finalNumOfClusters

4.4.1 Detailed Description

Definition at line 30 of file DetermClusterNum.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 DetermClusterNum::DetermClusterNum ( )

Definition at line 14 of file DetermClusterNum.cpp.

```
14                                  {
15     // TODO Auto-generated constructor stub
16
17 }
```

4.4.2.2 DetermClusterNum::~DetermClusterNum ( ) [virtual]

Definition at line 23 of file DetermClusterNum.cpp.

```
23                                   {
24     // TODO Auto-generated destructor stub
25 }
```

4.4.3 Member Function Documentation

4.4.3.1 const int& DetermClusterNum::getFinalNumOfClusters ( ) [inline]

Definition at line 48 of file DetermClusterNum.h.

```
49     {
50         return finalNumOfClusters;
51     }
```

4.4.3.2 void DetermClusterNum::iterativeRefinement (std::map< int, float > & *eval_graph*)

Definition at line 32 of file DetermClusterNum.cpp.

```
33 {
34     // some necessary pre-processing to remove irregular shapes for the L-method
35     removeExtreme(eval_graph);
36
37     // start from the first to search the point with knee
38     int cutoff, lastKnee;
39     int currentKnee = eval_graph.rbegin()->first;
40     cutoff = currentKnee;
41     do  // an iterative refinement for the L-method
42     {
43         lastKnee = currentKnee;
44         currentKnee = LMethod(eval_graph, cutoff);
45         std::cout << "returned value is " << currentKnee <<", cutoff is " << cutoff << std::endl;
46         cutoff = currentKnee*2;
47     }while(currentKnee < lastKnee);
48
49     // get the optimal number of clusters
50     finalNumOfClusters = currentKnee;
51
52     std::cout << finalNumOfClusters << std::endl;
53 }
```

4.4.3.3 const int DetermClusterNum::LMethod (const std::map< int, float > & *eval_graph*, const int & *cutoff*) [private]

Definition at line 62 of file DetermClusterNum.cpp.

```
63 {
64     struct CompObj { float val; int index; };
65 // #pragma omp declare reduction(minimum : struct CompObj : omp_out = omp_in.val < omp_out.val ? omp_in :
       omp_out)
66     struct CompObj RMSE;
67     RMSE.val = FLT_MAX;
68     RMSE.index = -1;
69
70     const int& firstIndex = eval_graph.begin()->first;
71     /* find the minimal c that minimizes RMSE for the selected cutoff */
72 #pragma omp parallel num_threads(8)
73     {
74     #pragma omp nowait
75         for(int i=firstIndex;i<=cutoff;++i)
76         {
77             /* left segment linear least square fitting */
78             std::vector<float> index_vec;
79             std::vector<float> dist_vec;
80
81             // assign the vector for left segment
82             std::map<int, float>::const_iterator iter;
83             for(int j=firstIndex;j<=i;++j)
84             {
85                 iter = eval_graph.find(j);
86                 if(iter!=eval_graph.end())
87                 {
88                     index_vec.push_back(iter->first);
89                     dist_vec.push_back(iter->second);
90                 }
91             }
92             Eigen::MatrixXf A_sub(2, index_vec.size());
93             A_sub.row(0) = Eigen::VectorXf::Map(&(index_vec[0]), index_vec.size()).transpose();
94             A_sub.row(1) = Eigen::VectorXf::Constant(index_vec.size(), 1.0).transpose();
95             Eigen::VectorXf b_sub = Eigen::VectorXf::Map(&(dist_vec[0]), index_vec.size());
96             A_sub.transposeInPlace();
97             int firstRows = A_sub.rows();
98
99             // solve the least-square fitting problems
100             Eigen::VectorXf c = A_sub.colPivHouseholderQr().solve(b_sub);
101             Eigen::VectorXf error = b_sub-A_sub*c;
102             float rmse_l = error.transpose()*error;
103
104             /* right segment linear least square fitting */
105             index_vec.clear();
106             dist_vec.clear();
107
108             // assignment of the vector
109             for(int j=i+1;j<=cutoff;++j)
110             {
111                 iter = eval_graph.find(j);
112                 if(iter!=eval_graph.end())
113                 {
114                     index_vec.push_back(iter->first);
115                     dist_vec.push_back(iter->second);
116                 }
117             }
118             A_sub = Eigen::MatrixXf(2, index_vec.size());
119             A_sub.row(0) = Eigen::VectorXf::Map(&(index_vec[0]), index_vec.size()).transpose();
120             A_sub.row(1) = Eigen::VectorXf::Constant(index_vec.size(), 1.0).transpose();
121             b_sub = Eigen::VectorXf::Map(&(dist_vec[0]), index_vec.size());
122             A_sub.transposeInPlace();
123             int secondRows = A_sub.rows();
124
125             // least-square fitting problem
126             c = A_sub.colPivHouseholderQr().solve(b_sub);
127             error = b_sub-A_sub*c;
128             float rmse_r = error.transpose()*error;
129
130             /* compute the total weighted error */
131             float rmse = float(firstRows)/float(firstRows+secondRows)*rmse_l+
132                     float(secondRows)/float(firstRows+secondRows)*rmse_r;
133             // update the rmse value and index
134         #pragma omp critical
135             if(RMSE.val>rmse)
136             {
137                 RMSE.val=rmse;
138                 RMSE.index=i;
139             }
```

```
140         }
141     }
142
143     return RMSE.index;
144 }
```

4.4.3.4 void DetermClusterNum::recordLMethodResult (const int & *normOption*)

Definition at line 151 of file DetermClusterNum.cpp.

```
152 {
153     std::ofstream readme("../dataset/LMethod",ios::out | ios::app);
154     if(!readme)
155     {
156         std::cout << "Error creating readme!" << std::endl;
157         exit(1);
158     }
159     readme << "Optimal cluster number of norm " << normOption << " is " <<
      finalNumOfClusters << std::endl;
160     readme << std::endl;
161     readme.close();
162 }
```

4.4.3.5 void DetermClusterNum::removeExtreme (std::map< int, float > & *eval_graph*) [private]

Definition at line 169 of file DetermClusterNum.cpp.

```
170 {
171     // find the left index with the maximal distance
172     float maxDist = -1.0;
173     int leftIndex = -1;
174     for(auto iter:eval_graph)
175     {
176         if(maxDist<iter.second)
177         {
178             maxDist=iter.second;
179             leftIndex=iter.first;
180         }
181     }
182     auto iter_index = eval_graph.find(leftIndex);
183
184     // remove some irregular indices
185     for(auto iter=eval_graph.begin();iter!=iter_index;)
186     {
187         if(iter->first<leftIndex&&iter->second<maxDist)
188             eval_graph.erase(iter++);
189         else
190             ++iter;
191     }
192 }
```

4.4.4 Member Data Documentation

4.4.4.1 int DetermClusterNum::finalNumOfClusters [private]

Definition at line 72 of file DetermClusterNum.h.

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

- DetermClusterNum.h
- DetermClusterNum.cpp

4.5 ExtractedLine Struct Reference

`#include <IOHandler.h>`

Public Member Functions

- ExtractedLine (const int &pointIndex, const int &cluster)

Public Attributes

- int lineNum
- int cluster

4.5.1 Detailed Description

Definition at line 35 of file IOHandler.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ExtractedLine::ExtractedLine (const int & *pointIndex*, const int & *cluster*) [inline]

Definition at line 39 of file IOHandler.h.

```
41                  :lineNum(pointIndex),cluster(cluster)
42      {}
```

4.5.3 Member Data Documentation

4.5.3.1 int ExtractedLine::cluster

Definition at line 38 of file IOHandler.h.

4.5.3.2 int ExtractedLine::lineNum

Definition at line 37 of file IOHandler.h.

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

- IOHandler.h

4.6 FeatureLine Struct Reference

`#include <IOHandler.h>`

Public Member Functions

- FeatureLine ()
- FeatureLine (const std::vector< std::vector< float > > &dataVec)

Public Attributes

- std::vector< MeanLine > centerMass
- std::vector< ExtractedLine > closest
- std::vector< ExtractedLine > furthest
- std::vector< int > group
- std::vector< int > totalNum

4.6.1 Detailed Description

Definition at line 82 of file IOHandler.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 FeatureLine::FeatureLine ( ) [inline]

Definition at line 90 of file IOHandler.h.

```
91      {
92
93      }
```

4.6.2.2 FeatureLine::FeatureLine (const std::vector< std::vector< float > > & *dataVec*) [inline]

Definition at line 95 of file IOHandler.h.

```
96      {
97          group = std::vector<int>(dataVec.size());
98          totalNum = std::vector<int>(dataVec.size());
99      }
```

4.6.3 Member Data Documentation

4.6.3.1 std::vector<MeanLine> FeatureLine::centerMass

Definition at line 84 of file IOHandler.h.

4.6.3.2 std::vector<ExtractedLine> FeatureLine::closest

Definition at line 85 of file IOHandler.h.

4.6.3.3 std::vector<ExtractedLine> FeatureLine::furthest

Definition at line 86 of file IOHandler.h.

4.6.3.4 std::vector<int> FeatureLine::group

Definition at line 87 of file IOHandler.h.

4.6.3.5 std::vector<int> FeatureLine::totalNum

Definition at line 88 of file IOHandler.h.

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

- IOHandler.h

4.7 Initialization Class Reference

`#include <Initialization.h>`

Static Public Member Functions

- static void generateRandomPos (MatrixXf &clusterCenter, const int &column, const MatrixXf &cArray, const int &Cluster)
- static void generateFromSamples (MatrixXf &clusterCenter, const int &column, const MatrixXf &cArray, const int &Cluster)
- static void generateFarSamples (MatrixXf &clusterCenter, const int &column, const MatrixXf &cArray, const int &Cluster, const int &normOption, const MetricPreparation &object)

4.7.1 Detailed Description

Definition at line 24 of file Initialization.h.

4.7.2 Member Function Documentation

4.7.2.1 void Initialization::generateFarSamples (MatrixXf & *clusterCenter,* const int & *column,* const MatrixXf & *cArray,* const int & *Cluster,* const int & *normOption,* const MetricPreparation & *object*) [static]

Definition at line 108 of file Initialization.cpp.

```
114 {
115     assert(column==cArray.cols());
116     const int Total = cArray.rows();
117     clusterCenter = MatrixXf(Cluster,column);
118     int number[Cluster], selection;
119     srand(time(0));
120     const int& MaxNum = cArray.rows();
121     number[0] = rand()%MaxNum;
122     int chosen = 1;
123 
124     float percentage, nearest, toCentroid;
125     VectorXf distance(Total);
126     double squredSummation;
127     float left, right;
128     while(chosen<Cluster)
129     {
130         percentage = float(rand()/(double)RAND_MAX);
131         for (int i = 0; i < Total; ++i)
132         {
133             nearest = FLT_MAX;
134             for (int j = 0; j < chosen; ++j)
135             {
136                 toCentroid = getDisimilarity(cArray, i, number[j], normOption, object);
137                 if(nearest>toCentroid)
138                     nearest=toCentroid;
139             }
140             distance(i)=nearest*nearest;
141         }
142         squredSummation = distance.sum();
143         left = 0.0, right = 0.0;
144         for (int i = 0; i < Total; ++i)
145         {
146             left = right;
147             right += float((double)distance(i)/squredSummation);
148             if(left < percentage && percentage <= right)
149             {
150                 selection = i;
151                 break;
152             }
153         }
154         number[chosen] = selection;
155         chosen++;
156     }
157 
158 #pragma omp parallel for schedule(static) num_threads(8)
159     for (int i = 0; i < Cluster; ++i)
160     {
161         clusterCenter.row(i) = cArray.row(number[i]);
162     }
163 
164 }
```

4.7.2.2 void Initialization::generateFromSamples (MatrixXf & *clusterCenter,* const int & *column,* const MatrixXf & *cArray,* const int & *Cluster*) [static]

Definition at line 54 of file Initialization.cpp.

```
58 {
59     clusterCenter = MatrixXf(Cluster,column);
60     std::vector<int> number(Cluster);
61     srand(time(0));
62 
63     const int& MaxNum = cArray.rows();
64 
65     std::cout << MaxNum << std::endl;
66 
67     number[0] = rand()%MaxNum;
```

```
68      int randNum, chosen = 1;
69      bool found;
70      for (int i = 1; i < Cluster; ++i)
71      {
72          do
73          {
74              randNum = rand()%MaxNum;
75              found = false;
76              for(int j=0;j<chosen;j++)
77              {
78                  if(randNum==number[j])
79                  {
80                      found = true;
81                      break;
82                  }
83              }
84          }while(found!=false);
85          number[i] = randNum;
86          ++chosen;
87      }
88      assert(chosen==Cluster);
89      assert(column==cArray.cols());
90
91  #pragma omp parallel for schedule(static) num_threads(8)
92      for (int i = 0; i < Cluster; ++i)
93      {
94          clusterCenter.row(i) = cArray.row(number[i]);
95      }
96  }
```

4.7.2.3 void Initialization::generateRandomPos (MatrixXf & *clusterCenter,* const int & *column,* const MatrixXf & *cArray,* const int & *Cluster*) [static]

Definition at line 18 of file Initialization.cpp.

```
22  {
23      clusterCenter = MatrixXf::Random(Cluster, column);
24      MatrixXf range(2, column);
25      range.row(0) = cArray.colwise().maxCoeff();  //first row contains max
26      range.row(1) = cArray.colwise().minCoeff();  //second row contains min
27      VectorXf diffRange = range.row(0)-range.row(1);
28
29      MatrixXf diagonalRange = MatrixXf::Zero(column,column);
30
31  #pragma omp parallel for schedule(static) num_threads(8)
32      for (int i = 0; i < column; ++i)
33      {
34          diagonalRange(i,i) = diffRange(i);
35      }
36      clusterCenter = (clusterCenter+MatrixXf::Constant(Cluster,column,1.0))/2.0;
37
38  #pragma omp parallel for schedule(static) num_threads(8)
39      for (int i = 0; i < Cluster; ++i)
40      {
41          clusterCenter.row(i) = clusterCenter.row(i)*diagonalRange+range.row(1);
42      }
43  }
```

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

- Initialization.h
- Initialization.cpp

4.8 IOHandler Class Reference

`#include <IOHandler.h>`

Static Public Member Functions

- static void readFile (const string &fileName, std::vector< std::vector< float > > &dataVec, int &vertexCount, const int &dimension, int &maxElement)
- static void readFile (const string &fileName, std::vector< std::vector< float > > &dataVec, int &vertexCount, const int &dimension, const int &trajectoryNum, const int &Frame)
- static void printVTK (const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &vertexCount, const int &dimension, const std::vector< int > &clusterNumber, const std::vector< float > &sCluster)
- static void printVTK (const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &vertexCount, const int &dimension)
- static void printVTK (const string &fileName, const std::vector< MeanLine > &dataVec, const int &vertex↩ Count, const int &dimension, const std::vector< float > &sCluster)
- static void printToFull (const std::vector< std::vector< float > > &dataVec, const std::vector< int > &group, const std::vector< int > &totalNum, const string &groupName, const string &fullName, const int &dimension)
- static void printToFull (const std::vector< std::vector< float > > &dataVec, const std::vector< float > &sData, const string &groupName, const string &fullName, const int &dimension)
- static void printToFull (const std::vector< std::vector< float > > &origin, const std::vector< int > &group, const string &fullName, const string &groupName, const int &dimension)
- static void printToFull (const std::vector< std::vector< float > > &dataVec, const std::vector< int > &group, const std::vector< float > &sCluster, const string &groupName, const string &fullName, const int &dimension)
- static void writeReadme (const double &PCA_KMeans_delta, const double &KMeans_delta)
- static void writeReadme (const string &comment, const std::vector< float > &sAverage)
- static void writeReadme (const std::vector< string > &timeName, const std::vector< double > &timeDiff, const int &cluster)
- static void writeReadme (const std::vector< string > &timeName, const std::vector< string > &timeDiff, const int &cluster)
- static void writeReadme (const std::vector< ExtractedLine > &closest, const std::vector< ExtractedLine > &furthest, const int &normOption)
- static void writeReadme (const std::vector< ExtractedLine > &closest, const std::vector< ExtractedLine > &furthest)
- static void writeReadme (const float &closestAverage, const float &furthestAverage)
- static void writeReadme (const string &comments)
- static void writeReadme (const float &entropy, const Silhouette &sil, const string &norm_str)
- static void writeReadMe (const float &value, const string &dataSet, const string &clustering, const string &value_name)
- static void writeGroupSize (const std::vector< int > &storage)
- static void expandArray (MatrixXf &data, const std::vector< std::vector< float > > &dataVec, const int &dimension, const int &maxElements)
- static void expandArray (std::vector< std::vector< float > > &equalArray, const std::vector< std::vector< float > > &trajectories, const int &dimension, const int &maxRowNum)
- static void sampleArray (MatrixXf &data, const std::vector< std::vector< float > > &dataVec, const int &dimension, const int &maxElements)
- static void formArray (float ***data, const std::vector< std::vector< float > > &dataVec, const int &dimension)
- static void uniformArcSampling (MatrixXf &data, const std::vector< std::vector< float > > &dataVec, const int &dimension, const int &maxElements)
- static void deleteArray (float **data, const int &row)
- static void assignVec (std::vector< std::vector< float > > &closestStreamline, std::vector< int > &cluster, const std::vector< ExtractedLine > &closest, int &pointNumber, const std::vector< std::vector< float > > &dataVec)
- static void assignVec (std::vector< int > &cluster, const std::vector< MeanLine > &centerMass)
- static void writeGroup (const std::vector< int > &group, const std::vector< std::vector< float > > &dataVec)
- static void printQuery (const int &normOption, const int &order, const StringQuery &queryResult, const std↩ ::vector< std::vector< float > > &dataVec)
- static void printTXT (float **data, const int &Row, const int &Column)

- static void printFeature (const string &fileName, const std::vector< std::vector< float > > &array, const std::vector< float > &sCluster, const int &dimension)
- static void printFeature (const string &fileName, const std::vector< std::vector< float > > &array, const std::vector< float > &sCluster, const std::vector< float > &rotation, const int &dimension)
- static void printClusters (const std::vector< std::vector< float > > &dataVec, const std::vector< int > &group, const std::vector< int > &storage, const string &groupName, const string &fullName, const int &dimension)
- static void printClustersNoise (const std::vector< std::vector< float > > &dataVec, const std::vector< int > &group, const std::vector< int > &storage, const string &groupName, const string &fullName, const int &dimension)
- static void generateReadme (const std::vector< string > &activityList, const std::vector< double > &timeList, const int &normOption, const int &numClusters, const float &sValue, const float &threshold)
- static void generateReadme (const std::vector< string > &activityList, const std::vector< string > &timeList)
- static void generateGroups (const std::vector< std::vector< int > > &storage)
- static void generateGroups (const std::vector< std::vector< int > > &storage, const string &fileName)
- static void readClusteringNumber (std::unordered_map< int, int > &clusMap, const string &fileName)

4.8.1 Detailed Description

Definition at line 106 of file IOHandler.h.

4.8.2 Member Function Documentation

4.8.2.1 void IOHandler::assignVec (std::vector< std::vector< float > > & *closestStreamline,* std::vector< int > & *cluster,* const std::vector< **ExtractedLine** > & *closest,* int & *pointNumber,* const std::vector< std::vector< float > > & *dataVec*) [static]

Definition at line 1048 of file IOHandler.cpp.

```
1053 {
1054     if(closest.empty())
1055         return;
1056     closestStreamline = std::vector<std::vector<float> >(closest.size(), std::vector<float>());
1057     cluster = std::vector<int>(closest.size());
1058     pointNumber = 0;
1059     for (int i = 0; i < closestStreamline.size(); ++i)
1060     {
1061         closestStreamline[i] = dataVec[closest[i].lineNum];
1062         pointNumber+=closestStreamline[i].size();
1063         cluster[i] = closest[i].cluster;
1064     }
1065 }
```

4.8.2.2 void IOHandler::assignVec (std::vector< int > & *cluster,* const std::vector< **MeanLine** > & *centerMass*) [static]

Definition at line 1074 of file IOHandler.cpp.

```
1076 {
1077     cluster = std::vector<int>(centerMass.size());
1078 #pragma omp parallel for schedule(static) num_threads(8)
1079     for (int i = 0; i < cluster.size(); ++i)
1080     {
1081         cluster[i] = centerMass[i].cluster;
1082     }
1083 }
```

4.8.2.3 void IOHandler::deleteArray (float ** *data*, const int & *row*) [static]

Definition at line 797 of file IOHandler.cpp.

```
799 {
800     if(data==NULL)
801         return;
802 #pragma omp parallel for schedule(static) num_threads(8)
803     for (int i = 0; i < row; ++i)
804     {
805         delete[] data[i];
806     }
807     delete[] data;
808 }
```

4.8.2.4 void IOHandler::expandArray (MatrixXf & *data*, const std::vector< std::vector< float > > & *dataVec*, const int & *dimension*, const int & *maxElements*) [static]

Definition at line 439 of file IOHandler.cpp.

```
443 {
444     data = Eigen::MatrixXf(dataVec.size(), maxElements);
445 #pragma omp parallel for schedule(static) num_threads(8)
446     for (int i = 0; i < dataVec.size(); ++i)
447     {
448         const std::vector<float>& eachVec = dataVec[i];
449         const int& vecSize = eachVec.size();
450         //data.row(i) = Eigen::VectorXf::Map(&(eachVec[0]), vecSize);
451         for (int j = 0; j<vecSize; j++)
452             data(i,j) = eachVec[j];
453
454         for (int j = vecSize; j < maxElements; j=j+dimension)
455         {
456             for (int k=0; k<dimension; k++)
457                 data(i,j+k) = eachVec[vecSize-dimension+k];
458         }
459     }
460 }
```

4.8.2.5 void IOHandler::expandArray (std::vector< std::vector< float > > & *equalArray*, const std::vector< std::vector< float > > & *trajectories*, const int & *dimension*, const int & *maxRowNum*) [static]

Definition at line 1254 of file IOHandler.cpp.

```
1258 {
1259     equalArray = std::vector<std::vector<float> >(trajectories.size(),
1260                  std::vector<float>(maxElement));
1261 #pragma omp parallel for schedule(static) num_threads(8)
1262     for (int i = 0; i < trajectories.size(); ++i)
1263     {
1264         std::vector<float>& tempRow = equalArray[i];
1265         const std::vector<float>& tempTraj = trajectories[i];
1266         const int& vecSize = tempTraj.size();
1267         memcpy(&(tempRow[0]), &(tempTraj[0]), vecSize*sizeof(float));
1268         for (int j = vecSize; j < maxElement; j=j+dimension)
1269         {
1270             memcpy(&(tempRow[j]), &(tempTraj[vecSize-dimension]),
1271                    dimension*sizeof(float));
1272         }
1273     }
1274 }
```

4.8.2.6 void IOHandler::formArray (float *** *data*, const std::vector< std::vector< float > > & *dataVec*, const int & *dimension*) [static]

Definition at line 630 of file IOHandler.cpp.

```
633 {
634     *data = new float*[dataVec.size()];
635 #pragma omp parallel for schedule(static) num_threads(8)
636     for (int i = 0; i < dataVec.size(); ++i)
637     {
638         const int& arraySize = dataVec[i].size();
639         (*data)[i] = new float[arraySize];
640         memcpy(&(*data)[i][0], &(dataVec[i][0]), arraySize*sizeof(float));
641     }
642 }
```

4.8.2.7 void IOHandler::generateGroups (const std::vector< std::vector< int > > & *storage*) [static]

Definition at line 1740 of file IOHandler.cpp.

```
1741 {
1742     if(storage.empty())
1743         return;
1744     std::ofstream readme("../dataset/Storage",ios::out|ios::app);
1745     if(!readme)
1746     {
1747         std::cout << "Error creating Storage!" << std::endl;
1748         exit(1);
1749     }
1750
1751     readme << std::endl;
1752     const int& groupSize = storage.size();
1753     std::vector<int> element;
1754     for(int i=0;i<groupSize;++i)
1755     {
1756         element = storage[i];
1757         if(element.empty())
1758             continue;
1759         for(int j=0;j<element.size();++j)
1760             readme << element[j] << " ";
1761         readme << std::endl;
1762     }
1763     std::cout << std::endl;
1764     readme.close();
1765 }
```

4.8.2.8 void IOHandler::generateGroups (const std::vector< std::vector< int > > & *storage*, const string & *fileName*) [static]

Definition at line 1774 of file IOHandler.cpp.

```
1775 {
1776     if(storage.empty())
1777         return;
1778     std::ofstream readme(("../dataset/"+fileName).c_str(),ios::out);
1779     if(!readme)
1780     {
1781         std::cout << "Error creating Storage!" << std::endl;
1782         exit(1);
1783     }
1784
1785     readme << std::endl;
1786     const int& groupSize = storage.size();
1787     std::vector<int> element;
1788     for(int i=0;i<groupSize;++i)
1789     {
1790         element = storage[i];
1791         if(element.empty())
1792             continue;
1793         for(int j=0;j<element.size();++j)
1794             readme << element[j] << " ";
1795         readme << std::endl;
1796     }
1797     std::cout << std::endl;
1798     readme.close();
1799 }
```

4.8.2.9 void IOHandler::generateReadme (const std::vector< string > & *activityList*, const std::vector< double > & *timeList*, const int & *normOption*, const int & *numClusters*, const float & *sValue*, const float & *threshold*) [static]

Definition at line 1635 of file IOHandler.cpp.

```
1641 {
1642     if(activityList.empty()||timeList.empty())
1643         return;
1644     std::ofstream readme("../dataset/README",ios::out | ios::app);
1645     if(!readme)
1646     {
1647         std::cout << "Error creating readme!" << std::endl;
1648         exit(1);
1649     }
1650     readme << "-----------------------------------------------" << std::endl;
1651     readme << "Norm: " << normOption << std::endl;
1652     readme << "Clusters: " << numClusters << std::endl;
1653     readme << "Silhouette: " << sValue << std::endl;
1654     readme << "Input threshold: " << threshold << std::endl;
1655     for (int i = 0; i < activityList.size(); ++i)
1656     {
1657         readme << activityList[i] << timeList[i] << " s." << std::endl;
1658     }
1659     readme << std::endl;
1660     readme.close();
1661 }
```

4.8.2.10 void IOHandler::generateReadme (const std::vector< string > & *activityList*, const std::vector< string > & *timeList*) [static]

Definition at line 1670 of file IOHandler.cpp.

```
1672 {
1673     if(activityList.empty()||timeList.empty())
1674         return;
1675     std::ofstream readme("../dataset/README",ios::out | ios::app);
1676     if(!readme)
1677     {
1678         std::cout << "Error creating readme!" << std::endl;
1679         exit(1);
1680     }
1681     readme << "-----------------------------------------------" << std::endl;
1682     for (int i = 0; i < activityList.size(); ++i)
1683     {
1684         readme << activityList[i] << timeList[i] << std::endl;
1685     }
1686     readme.close();
1687 }
```

4.8.2.11 void IOHandler::printClusters (const std::vector< std::vector< float > > & *dataVec*, const std::vector< int > & *group*, const std::vector< int > & *storage*, const string & *groupName*, const string & *fullName*, const int & *dimension*) [static]

Definition at line 1526 of file IOHandler.cpp.

```
1532 {
1533     if(group.empty()||storage.empty())
1534         return;
1535     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
1536     if(!fout)
1537     {
1538         std::cout << "Error opening the file!" << std::endl;
1539         exit(1);
1540     }
1541
1542     fout << "SCALARS " << groupName << " int 1" << std::endl;
1543     fout << "LOOKUP_TABLE " << groupName+string("_table") << std::endl;
```

```
1544
1545     int arraySize;
1546     for (int i = 0; i < dataVec.size(); ++i)
1547     {
1548         arraySize = dataVec[i].size()/dimension;
1549         for (int j = 0; j < arraySize; ++j)
1550         {
1551             fout << group[i] << std::endl;
1552         }
1553     }
1554
1555     fout << "SCALARS " <<  groupName + "_num" << " int 1" << std::endl;
1556     fout << "LOOKUP_TABLE " <<  groupName+string("_num_table") << std::endl;
1557
1558     for (int i = 0; i < dataVec.size(); ++i)
1559     {
1560         arraySize = dataVec[i].size()/dimension;
1561         for (int j = 0; j < arraySize; ++j)
1562         {
1563             fout << storage[group[i]] << std::endl;
1564         }
1565     }
1566     fout.close();
1567 }
```

4.8.2.12 void IOHandler::printClustersNoise (const std::vector< std::vector< float > > & *dataVec*, const std::vector< int > & *group*, const std::vector< int > & *storage*, const string & *groupName*, const string & *fullName*, const int & *dimension*) [static]

Definition at line 1580 of file IOHandler.cpp.

```
1586 {
1587     /* in case you've noise, so group_id would be -1 */
1588     if(group.empty()||storage.empty())
1589         return;
1590     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
1591     if(!fout)
1592     {
1593         std::cout << "Error opening the file!" << std::endl;
1594         exit(1);
1595     }
1596
1597     fout << "SCALARS " << groupName << " int 1" << std::endl;
1598     fout << "LOOKUP_TABLE " << groupName+string("_table") << std::endl;
1599
1600     int arraySize;
1601     for (int i = 0; i < dataVec.size(); ++i)
1602     {
1603         arraySize = dataVec[i].size()/dimension;
1604         for (int j = 0; j < arraySize; ++j)
1605         {
1606             fout << group[i] << std::endl;
1607         }
1608     }
1609
1610     fout << "SCALARS " <<  groupName + "_num" << " int 1" << std::endl;
1611     fout << "LOOKUP_TABLE " <<  groupName+string("_num_table") << std::endl;
1612
1613     for (int i = 0; i < dataVec.size(); ++i)
1614     {
1615         arraySize = dataVec[i].size()/dimension;
1616         for (int j = 0; j < arraySize; ++j)
1617         {
1618             fout << storage[group[i]+1] << std::endl;
1619         }
1620     }
1621     fout.close();
1622 }
```

4.8.2.13 void IOHandler::printFeature (const string & *fileName*, const std::vector< std::vector< float > > & *array*, const std::vector< float > & *sCluster*, const int & *dimension*) [static]

Definition at line 1323 of file IOHandler.cpp.

```
1327 {
1328     if(array.empty() || sCluster.empty())
1329         return;
1330     stringstream ss;
1331     ss << "../dataset/" << fileName;
1332     ofstream fout(ss.str().c_str(), ios::out);
1333     if(!fout)
1334     {
1335         std::cout << "Error creating file!" << std::endl;
1336         exit(-1);
1337     }
1338 
1339     int vertexCount = 0;
1340     for (int i = 0; i < array.size(); ++i)
1341     {
1342         vertexCount += array[i].size();
1343     }
1344     vertexCount /= dimension;
1345 
1346     fout << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
1347          << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
1348     fout << "POINTS " << vertexCount << " float" << std::endl;
1349 
1350     int subSize, arraySize;
1351     std::vector<float> tempVec;
1352     for (int i = 0; i < array.size(); ++i)
1353     {
1354         tempVec = array[i];
1355         subSize = tempVec.size()/dimension;
1356         for (int j = 0; j < subSize; ++j)
1357         {
1358             for (int k = 0; k < dimension; ++k)
1359             {
1360                 fout << tempVec[j*dimension+k] << " ";
1361             }
1362             fout << endl;
1363         }
1364     }
1365 
1366     fout << "LINES " << array.size() << " " << (vertexCount+array.size()) << std::endl;
1367 
1368     subSize = 0;
1369     for (int i = 0; i < array.size(); ++i)
1370     {
1371         arraySize = array[i].size()/dimension;
1372         fout << arraySize << " ";
1373         for (int j = 0; j < arraySize; ++j)
1374         {
1375             fout << subSize+j << " ";
1376         }
1377         subSize+=arraySize;
1378         fout << std::endl;
1379     }
1380     fout << "POINT_DATA" << " " << vertexCount << std::endl;
1381     fout << "SCALARS group int 1" << std::endl;
1382     fout << "LOOKUP_TABLE group_table" << std::endl;
1383 
1384     for (int i = 0; i < array.size(); ++i)
1385     {
1386         arraySize = array[i].size()/dimension;
1387         for (int j = 0; j < arraySize; ++j)
1388         {
1389             fout << i << std::endl;
1390         }
1391     }
1392 
1393     fout << "SCALARS silhouette float 1" << std::endl;
1394     fout << "LOOKUP_TABLE silhouette_table" << std::endl;
1395 
1396     for (int i = 0; i < array.size(); ++i)
1397     {
1398         arraySize = array[i].size()/dimension;
1399         for (int j = 0; j < arraySize; ++j)
1400         {
1401             fout << sCluster[i] << std::endl;
1402         }
1403     }
1404     fout.close();
1405 }
```

4.8.2.14 void IOHandler::printFeature (const string & *fileName*, const std::vector< std::vector< float > > & *array*, const std::vector< float > & *sCluster*, const std::vector< float > & *rotation*, const int & *dimension*) [static]

Definition at line 1417 of file IOHandler.cpp.

```
1422 {
1423     if(array.empty() || sCluster.empty())
1424             return;
1425         stringstream ss;
1426         ss << "../dataset/" << fileName;
1427         ofstream fout(ss.str().c_str(), ios::out);
1428         if(!fout)
1429         {
1430             std::cout << "Error creating file!" << std::endl;
1431             exit(-1);
1432         }
1433 
1434         int vertexCount = 0;
1435         for (int i = 0; i < array.size(); ++i)
1436         {
1437             vertexCount += array[i].size();
1438         }
1439         vertexCount /= dimension;
1440 
1441         fout << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
1442              << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
1443         fout << "POINTS " << vertexCount << " float" << std::endl;
1444 
1445         int subSize, arraySize;
1446         std::vector<float> tempVec;
1447         for (int i = 0; i < array.size(); ++i)
1448         {
1449             tempVec = array[i];
1450             subSize = tempVec.size()/dimension;
1451             for (int j = 0; j < subSize; ++j)
1452             {
1453                 for (int k = 0; k < dimension; ++k)
1454                 {
1455                     fout << tempVec[j*dimension+k] << " ";
1456                 }
1457                 fout << endl;
1458             }
1459         }
1460 
1461         fout << "LINES " << array.size() << " " << (vertexCount+array.size()) << std::endl;
1462 
1463         subSize = 0;
1464         for (int i = 0; i < array.size(); ++i)
1465         {
1466             arraySize = array[i].size()/dimension;
1467             fout << arraySize << " ";
1468             for (int j = 0; j < arraySize; ++j)
1469             {
1470                 fout << subSize+j << " ";
1471             }
1472             subSize+=arraySize;
1473             fout << std::endl;
1474         }
1475         fout << "POINT_DATA" << " " << vertexCount << std::endl;
1476         fout << "SCALARS group int 1" << std::endl;
1477         fout << "LOOKUP_TABLE group_table" << std::endl;
1478 
1479         for (int i = 0; i < array.size(); ++i)
1480         {
1481             arraySize = array[i].size()/dimension;
1482             for (int j = 0; j < arraySize; ++j)
1483             {
1484                 fout << i << std::endl;
1485             }
1486         }
1487 
1488         fout << "SCALARS silhouette float 1" << std::endl;
1489         fout << "LOOKUP_TABLE silhouette_table" << std::endl;
1490 
1491         for (int i = 0; i < array.size(); ++i)
1492         {
1493             arraySize = array[i].size()/dimension;
1494             for (int j = 0; j < arraySize; ++j)
1495             {
1496                 fout << sCluster[i] << std::endl;
1497             }
1498         }
1499 
```

```
1500         fout << "SCALARS rotation float 1" << std::endl;
1501         fout << "LOOKUP_TABLE rotation_table" << std::endl;
1502 
1503         for (int i = 0; i < array.size(); ++i)
1504         {
1505             arraySize = array[i].size()/dimension;
1506             for (int j = 0; j < arraySize; ++j)
1507             {
1508                 fout << rotation[i] << std::endl;
1509             }
1510         }
1511 
1512         fout.close();
1513 }
```

4.8.2.15 void IOHandler::printQuery (const int & *normOption,* const int & *order,* const StringQuery & *queryResult,* const std::vector< std::vector< float > > & *dataVec*) [static]

Definition at line 1120 of file IOHandler.cpp.

```
1124 {
1125     stringstream ss;
1126     ss << "../dataset/norm" << normOption << "_query" << order << "_target.vtk";
1127     const string& targetStr = ss.str();
1128     ss.str("");
1129     ss.clear();
1130 
1131     ss << "../dataset/norm" << normOption << "_query" << order << "_result.vtk";
1132     const string& resultStr = ss.str();
1133 
1134     /* print out the target streamline vtk file */
1135     ofstream fTarget(targetStr.c_str(),ios::out);
1136     if(!fTarget)
1137     {
1138         std::cout << "Error creating file!" << std::endl;
1139         exit(1);
1140     }
1141     std::cout << queryResult.index << std::endl;
1142     std::vector<float> targetVec = dataVec[queryResult.index];
1143     int pointNumber = targetVec.size()/3;
1144 
1145     fTarget << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
1146             << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
1147     fTarget << "POINTS " << pointNumber << " float" << std::endl;
1148 
1149     for (int i = 0; i < pointNumber; ++i)
1150     {
1151         fTarget << targetVec[i*3] << " " << targetVec[i*3+1]
1152                 << " " << targetVec[i*3+2] << std::endl;
1153     }
1154 
1155     fTarget << "LINES " << 1 << " " << (1+pointNumber) << std::endl;
1156     fTarget << pointNumber << " ";
1157     for (int i = 0; i < pointNumber; ++i)
1158     {
1159         fTarget << i << " ";
1160     }
1161     fTarget << std::endl;
1162     fTarget.close();
1163 
1164 
1165     /* print out the streamline query result vtk file */
1166     ofstream fResult(resultStr.c_str(),ios::out);
1167     if(!fResult)
1168     {
1169         std::cout << "Error creating file!" << std::endl;
1170         exit(1);
1171     }
1172     pointNumber = 0;
1173     const std::vector<int>& neighbor = queryResult.neighbor;
1174     for (int i = 0; i < neighbor.size(); ++i)
1175     {
1176         pointNumber += dataVec[neighbor[i]].size()/3;
1177     }
1178 
1179     fResult << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
1180             << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
1181     fResult << "POINTS " << pointNumber << " float" << std::endl;
1182 
```

```
1183     int subArraySize, indexNumber = 0;
1184     std::vector<float> tempVec;
1185     for (int i = 0; i < neighbor.size(); ++i)
1186     {
1187         tempVec = dataVec[neighbor[i]];
1188         subArraySize = tempVec.size()/3;
1189         for (int j = 0; j < subArraySize; ++j)
1190         {
1191             fResult << tempVec[3*j] << " " << tempVec[3*j+1] << " "
1192                     << tempVec[3*j+2] << std::endl;
1193         }
1194     }
1195
1196     fResult << "LINES " << neighbor.size() << " "
1197             << (neighbor.size()+pointNumber) << std::endl;
1198     for (int i = 0; i < neighbor.size(); ++i)
1199     {
1200         tempVec = dataVec[neighbor[i]];
1201         subArraySize = tempVec.size()/3;
1202         fResult << subArraySize << " ";
1203         for (int j = 0; j < subArraySize; ++j)
1204         {
1205             fResult << (indexNumber+j) << " ";
1206         }
1207         fResult << std::endl;
1208         indexNumber += subArraySize;
1209     }
1210     fResult.close();
1211
1212 }
```

4.8.2.16 void IOHandler::printToFull (const std::vector< std::vector< float > > & *dataVec*, const std::vector< int > & *group*, const std::vector< int > & *totalNum*, const string & *groupName*, const string & *fullName*, const int & *dimension*) [static]

Definition at line 655 of file IOHandler.cpp.

```
661 {
662     if(group.empty()||totalNum.empty())
663         return;
664     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
665     if(!fout)
666     {
667         std::cout << "Error opening the file!" << std::endl;
668         exit(1);
669     }
670
671     fout << "SCALARS " << groupName << " int 1" << std::endl;
672     fout << "LOOKUP_TABLE " << groupName+string("_table") << std::endl;
673
674     int arraySize;
675     for (int i = 0; i < dataVec.size(); ++i)
676     {
677         arraySize = dataVec[i].size()/dimension;
678         for (int j = 0; j < arraySize; ++j)
679         {
680             fout << group[i] << std::endl;
681         }
682     }
683
684     fout << "SCALARS " <<  groupName + "_num" << " int 1" << std::endl;
685     fout << "LOOKUP_TABLE " <<  groupName+string("_num_table") << std::endl;
686
687     for (int i = 0; i < dataVec.size(); ++i)
688     {
689         arraySize = dataVec[i].size()/dimension;
690         for (int j = 0; j < arraySize; ++j)
691         {
692             fout << totalNum[i] << std::endl;
693         }
694     }
695     fout.close();
696 }
```

4.8.2.17 void IOHandler::printToFull (const std::vector< std::vector< float > > & *dataVec*, const std::vector< float > & *sData*, const string & *groupName*, const string & *fullName*, const int & *dimension*) [static]

Definition at line 708 of file IOHandler.cpp.

```
713 {
714     if(sData.empty()||dataVec.empty())
715         return;
716     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
717     if(!fout)
718     {
719         std::cout << "Error opening the file!" << std::endl;
720         exit(1);
721     }
722
723     if(!sData.empty())
724     {
725         fout << "SCALARS " << groupName << " float 1" << std::endl;
726         fout << "LOOKUP_TABLE " << groupName+string("_table") << std::endl;
727
728         int arraySize;
729         for (int i = 0; i < dataVec.size(); ++i)
730         {
731             arraySize = dataVec[i].size()/dimension;
732             for (int j = 0; j < arraySize; ++j)
733             {
734                 fout << sData[i] << std::endl;
735             }
736         }
737     }
738
739     fout.close();
740 }
```

4.8.2.18 void IOHandler::printToFull (const std::vector< std::vector< float > > & *origin*, const std::vector< int > & *group*, const string & *fullName*, const string & *groupName*, const int & *dimension*) [static]

Definition at line 1286 of file IOHandler.cpp.

```
1291 {
1292     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
1293     if(!fout)
1294     {
1295         std::cout << "Error opening the file!" << std::endl;
1296         exit(1);
1297     }
1298
1299     fout << "SCALARS " << groupName << " int 1" << std::endl;
1300     fout << "LOOKUP_TABLE " << groupName+string("_table") << std::endl;
1301
1302     int arraySize;
1303     for (int i = 0; i < dataVec.size(); ++i)
1304     {
1305         arraySize = dataVec[i].size()/dimension;
1306         for (int j = 0; j < arraySize; ++j)
1307         {
1308             fout << group[i] << std::endl;
1309         }
1310     }
1311     fout.close();
1312 }
```

4.8.2.19 void IOHandler::printToFull (const std::vector< std::vector< float > > & *dataVec*, const std::vector< int > & *group*, const std::vector< float > & *sCluster*, const string & *groupName*, const string & *fullName*, const int & *dimension*) [static]

Definition at line 753 of file IOHandler.cpp.

```
759 {
760     if(dataVec.empty()||group.empty()||sCluster.empty())
761         return;
762     std::ofstream fout(fullName.c_str(), ios::out | ios::app );
763     if(!fout)
764     {
765         std::cout << "Error opening the file!" << std::endl;
766         exit(1);
767     }
768 
769     if(!sCluster.empty())
770     {
771         fout << "SCALARS " <<  groupName << " float 1" << std::endl;
772         fout << "LOOKUP_TABLE " <<  groupName+string("_table") << std::endl;
773 
774         int arraySize;
775         for (int i = 0; i < dataVec.size(); ++i)
776         {
777             arraySize = dataVec[i].size()/dimension;
778             for (int j = 0; j < arraySize; ++j)
779             {
780                 if(group[i]<0)
781                     fout << 0 << std::endl;
782                 else
783                     fout << sCluster[group[i]] << std::endl;
784             }
785         }
786     }
787     fout.close();
788 }
```

4.8.2.20 void IOHandler::printTXT (float ** *data,* const int & *Row,* const int & *Column*) [static]

Definition at line 1222 of file IOHandler.cpp.

```
1225 {
1226     std::ofstream fout("../dataset/full.txt", ios::out);
1227     if(!fout)
1228     {
1229         std::cout << "Error creating a file!" << std::endl;
1230         exit(1);
1231     }
1232     float *array;
1233     for (int i = 0; i < Row; ++i)
1234     {
1235         array = data[i];
1236         for (int j = 0; j < Column; ++j)
1237         {
1238             fout << array[j] << " ";
1239         }
1240         fout << std::endl;
1241     }
1242     fout.close();
1243 }
```

4.8.2.21 void IOHandler::printVTK (const string & *fileName,* const std::vector< std::vector< float > > & *dataVec,* const int & *vertexCount,* const int & *dimension,* const std::vector< int > & *clusterNumber,* const std::vector< float > & *sCluster*) [static]

Definition at line 186 of file IOHandler.cpp.

```
192 {
193     if(clusterNumber.empty()||sCluster.empty())
194         return;
195     std::ofstream fout(fileName.c_str(), ios::out);
196     if(!fout)
197     {
198         std::cout << "Error creating a new file!" << std::endl;
199         exit(1);
200     }
201     fout << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
```

```
202          << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
203     fout << "POINTS " << vertexCount << " float" << std::endl;
204
205     int subSize, arraySize;
206     std::vector<float> tempVec;
207     for (int i = 0; i < dataVec.size(); ++i)
208     {
209         tempVec = dataVec[i];
210         subSize = tempVec.size()/dimension;
211         for (int j = 0; j < subSize; ++j)
212         {
213             for (int k = 0; k < dimension; ++k)
214             {
215                 fout << tempVec[j*dimension+k] << " ";
216             }
217             fout << endl;
218         }
219     }
220
221     fout << "LINES " << dataVec.size() << " " << (vertexCount+dataVec.size()) << std::endl;
222
223     subSize = 0;
224     for (int i = 0; i < dataVec.size(); ++i)
225     {
226         arraySize = dataVec[i].size()/dimension;
227         fout << arraySize << " ";
228         for (int j = 0; j < arraySize; ++j)
229         {
230             fout << subSize+j << " ";
231         }
232         subSize+=arraySize;
233         fout << std::endl;
234     }
235     fout << "POINT_DATA" << " " << vertexCount << std::endl;
236     fout << "SCALARS group int 1" << std::endl;
237     fout << "LOOKUP_TABLE group_table" << std::endl;
238
239     for (int i = 0; i < dataVec.size(); ++i)
240     {
241         arraySize = dataVec[i].size()/dimension;
242         for (int j = 0; j < arraySize; ++j)
243         {
244             fout << clusterNumber[i] << std::endl;
245         }
246     }
247
248     if(!sCluster.empty())
249     {
250         fout << "SCALARS sCluster float 1" << std::endl;
251         fout << "LOOKUP_TABLE sCluster_table" << std::endl;
252
253         for (int i = 0; i < dataVec.size(); ++i)
254         {
255             arraySize = dataVec[i].size()/dimension;
256             for (int j = 0; j < arraySize; ++j)
257             {
258                 fout << sCluster[clusterNumber[i]] << std::endl;
259             }
260         }
261     }
262
263     fout.close();
264 }
```

4.8.2.22 void IOHandler::printVTK (const string & *fileName,* const std::vector< std::vector< float > > & *dataVec,* const int & *vertexCount,* const int & *dimension*) [static]

Definition at line 275 of file IOHandler.cpp.

```
279 {
280     if(dataVec.empty())
281         return;
282     std::ifstream fin(fileName.c_str());
283     if(fin.good())
284         return;
285     std::ofstream fout(fileName.c_str(), ios::out);
286     if(!fout)
287     {
288         std::cout << "Error creating a new file!" << std::endl;
```

```
289         exit(1);
290     }
291     fout << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
292          << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
293     fout << "POINTS " << vertexCount << " float" << std::endl;
294 
295     int subSize, arraySize;
296     std::vector<float> tempVec;
297     for (int i = 0; i < dataVec.size(); ++i)
298     {
299         tempVec = dataVec[i];
300         subSize = tempVec.size()/dimension;
301         for (int j = 0; j < subSize; ++j)
302         {
303             for (int k = 0; k < dimension; ++k)
304             {
305                 fout << tempVec[j*dimension+k] << " ";
306             }
307             fout << endl;
308         }
309     }
310 
311     fout << "LINES " << dataVec.size() << " " << (vertexCount+dataVec.size()) << std::endl;
312 
313     subSize = 0;
314     for (int i = 0; i < dataVec.size(); ++i)
315     {
316         arraySize = dataVec[i].size()/dimension;
317         fout << arraySize << " ";
318         for (int j = 0; j < arraySize; ++j)
319         {
320             fout << subSize+j << " ";
321         }
322         subSize+=arraySize;
323         fout << std::endl;
324     }
325     fout << "POINT_DATA" << " " << vertexCount << std::endl;
326     fout << "SCALARS group int 1" << std::endl;
327     fout << "LOOKUP_TABLE group_table" << std::endl;
328 
329     for (int i = 0; i < dataVec.size(); ++i)
330     {
331         arraySize = dataVec[i].size()/dimension;
332         for (int j = 0; j < arraySize; ++j)
333         {
334             fout << i << std::endl;
335         }
336     }
337 
338     fout.close();
339 }
```

4.8.2.23 void IOHandler::printVTK (const string & *fileName,* const std::vector< MeanLine > & *dataVec,* const int & *vertexCount,* const int & *dimension,* const std::vector< float > & *sCluster*) [static]

Definition at line 351 of file IOHandler.cpp.

```
356 {
357     if(dataVec.empty())
358         return;
359     std::ofstream fout(fileName.c_str(), ios::out);
360     if(!fout)
361     {
362         std::cout << "Error creating a new file!" << std::endl;
363         exit(1);
364     }
365     fout << "# vtk DataFile Version 3.0" << std::endl << "Bernard streamline" << std::endl
366          << "ASCII" << std::endl << "DATASET POLYDATA" << std::endl;
367     fout << "POINTS " << vertexCount << " float" << std::endl;
368 
369     int subSize, arraySize;
370     std::vector<float> tempVec;
371     for (int i = 0; i < dataVec.size(); ++i)
372     {
373         tempVec = dataVec[i].minCenter;
374         subSize = tempVec.size()/dimension;
375         for (int j = 0; j < subSize; ++j)
376         {
377             for (int k = 0; k < dimension; ++k)
```

```
378             {
379                 fout << tempVec[j*dimension+k] << " ";
380             }
381             fout << endl;
382         }
383     }
384 
385     fout << "LINES " << dataVec.size() << " " << (vertexCount+dataVec.size()) << std::endl;
386 
387     subSize = 0;
388     for (int i = 0; i < dataVec.size(); ++i)
389     {
390         arraySize = dataVec[i].minCenter.size()/dimension;
391         fout << arraySize << " ";
392         for (int j = 0; j < arraySize; ++j)
393         {
394             fout << subSize+j << " ";
395         }
396         subSize+=arraySize;
397         fout << std::endl;
398     }
399     fout << "POINT_DATA" << " " << vertexCount << std::endl;
400     fout << "SCALARS group int 1" << std::endl;
401     fout << "LOOKUP_TABLE group_table" << std::endl;
402 
403     for (int i = 0; i < dataVec.size(); ++i)
404     {
405         arraySize = dataVec[i].minCenter.size()/dimension;
406         for (int j = 0; j < arraySize; ++j)
407         {
408             fout << dataVec[i].cluster << std::endl;
409         }
410     }
411 
412     if(!sCluster.empty())
413     {
414         fout << "SCALARS sCluster float 1" << std::endl;
415         fout << "LOOKUP_TABLE sCluster_table" << std::endl;
416 
417         for (int i = 0; i < dataVec.size(); ++i)
418         {
419             arraySize = dataVec[i].minCenter.size()/dimension;
420             for (int j = 0; j < arraySize; ++j)
421             {
422                 fout << sCluster[dataVec[i].cluster] << std::endl;
423             }
424         }
425     }
426 
427     fout.close();
428 }
```

4.8.2.24 void IOHandler::readClusteringNumber (std::unordered_map< int, int > & *clusMap*, const string & *fileName*) [static]

Definition at line 1832 of file IOHandler.cpp.

```
1833 {
1834     std::ifstream readme(("../dataset/"+fileName).c_str(), ios::in);
1835     if(!readme)
1836     {
1837         std::cout << "Error creating Storage!" << std::endl;
1838         exit(1);
1839     }
1840     string line;
1841     int scopePos;
1842     while(getline(readme, line))
1843     {
1844         scopePos = line.find(":");
1845         if(scopePos==std::string::npos)
1846         {
1847             std::cout << "Error for clustering number reader..." << std::endl;
1848             exit(1);
1849         }
1850         clusMap[std::atoi(line.substr(0,scopePos).c_str())] = std::atoi(line.substr(scopePos+1).c_str());
1851     }
1852     readme.close();
1853 }
```

4.8.2.25 void IOHandler::readFile (const string & *fileName*, std::vector< std::vector< float > > & *dataVec*, int & *vertexCount*, const int & *dimension*, int & *maxElement*) [static]

Definition at line 19 of file IOHandler.cpp.

```
24 {
25
26     vertexCount = 0;
27     std::ifstream fin(fileName.c_str(), ios::in);
28     if(!fin)
29     {
30         std::cout << "Error creating files!" << std::endl;
31         exit(1);
32     }
33     stringstream ss;
34     std::vector<float> tempVec;
35
36     string line, part;
37
38     /* read partial number of streamlines */
39     //int MAXNUMBER;
40     //std::cout << "Input maximal trajectory numbers: " << std::endl;
41     //std::cin >> MAXNUMBER;
42     // set currentNumber to record how many streamlines u want to read in
43     //int currentNumber = 0;
44
45
46     /* read partial dimensions of curves */
47     //int MAXDIMENSION;
48     //std::cout << "Input maximal dimensions: " << std::endl;
49     //std::cin >> MAXDIMENSION;
50     // set currentNumber to record how many streamlines u want to read in
51     //int currentDimensions;
52
53     std::vector<float> vec(3);
54     float temp;
55     maxElement = 0;
56     while(getline(fin, line) /* && currentNumber < MAXNUMBER*/)
57     {
58         //currentDimensions = 0;
59         int tag = 0, count = 0;
60         ss.str(line);
61         while(ss>>part /*&& currentDimensions<3*MAXDIMENSION*/)
62         {
63             /* operations below would remove duplicate vertices because that would damage our computation
    */
64             temp = atof(part.c_str());
65             if(tag>=3)
66             {
67                 if(count<3)
68                 {
69                     vec[count] = temp;
70                     ++tag;
71                     ++count;
72                 }
73                 if(count==3)
74                 {
75                     int size = tempVec.size();
76                     //
    if(!(abs(vec[0]-tempVec[size-3])<1.0e-5&&abs(vec[1]-tempVec[size-2])<1.0e-5&&abs(vec[2]-tempVec.back())<1.0e-5))
77                     //{
78                     tempVec.push_back(vec[0]);
79                     tempVec.push_back(vec[1]);
80                     tempVec.push_back(vec[2]);
81                     //}
82                     count = 0;
83                 }
84                 continue;
85             }
86             tempVec.push_back(temp);
87             ++tag;
88             //currentDimensions++;
89         }
90         /* accept only streamlines with at least three vertices */
91         if(tempVec.size()/3>2)
92         {
93             if(maxElement<tempVec.size())
94                 maxElement = tempVec.size();
95             dataVec.push_back(tempVec);
96             vertexCount+=tempVec.size();
97         }
98         tempVec.clear();
99         ss.clear();
```

```
100         ss.str("");
101         //currentNumber++;
102     }
103     fin.close();
104 
105 
106     vertexCount/=dimension;
107     std::cout << "File reader has been completed, and it toally has " << dataVec.size() << " trajectories
    and "
108               << vertexCount << " vertices!" << std::endl;
109     std::cout << "Max dimension is " << maxElement << std::endl;
110 }
```

4.8.2.26 void IOHandler::readFile (const string & *fileName,* std::vector< std::vector< float > > & *dataVec,* int & *vertexCount,* const int & *dimension,* const int & *trajectoryNum,* const int & *Frame*) [static]

Definition at line 123 of file IOHandler.cpp.

```
129 {
130     vertexCount = trajectoryNum*(Frame-1);
131     dataVec = std::vector< std::vector<float> >(trajectoryNum, std::vector<float> ((Frame-1)*dimension));
132 #pragma omp parallel for schedule(static) num_threads(8)
133     /* from 1 to Frame-1 then pay attention to i index */
134     for (int i = 1; i < Frame; ++i)
135     {
136         stringstream ss;
137         ss << fileName << i << ".txt";
138         std::ifstream fin(ss.str().c_str(), ios::in);
139         if(!fin)
140         {
141             std::cout << "File doesn't exist for this number!" << std::endl;
142             exit(1);
143         }
144         float firstFloat;
145         string line, linePart;
146 
147         ss.clear();
148         ss.str("");
149         for (int j = 0; j < trajectoryNum; ++j)
150         {
151             getline(fin, line);
152 
153             assert(!line.empty());
154 
155             ss.str(line);
156             ss >> linePart;
157 
158             ss >> linePart;
159             dataVec[j][(i-1)*dimension] = atof(linePart.c_str());
160 
161             ss >> linePart;
162             dataVec[j][(i-1)*dimension+1] = atof(linePart.c_str());
163 
164             ss >> linePart;
165             dataVec[j][(i-1)*dimension+2] = atof(linePart.c_str());
166         }
167 
168         fin.close();
169         std::cout << "File " << i << " has been read in successfully!" << std::endl;
170     }
171 
172 
173 }
```

4.8.2.27 void IOHandler::sampleArray (MatrixXf & *data,* const std::vector< std::vector< float > > & *dataVec,* const int & *dimension,* const int & *maxElements*) [static]

Definition at line 551 of file IOHandler.cpp.

```
555 {
556     /*maxElements = INT_MIN;
557     int arraySize;
558     for (int i = 0; i < dataVec.size(); ++i)
559     {
560         arraySize = dataVec[i].size();
561         if(maxElements < arraySize)
562             maxElements = arraySize;
563     }
564     std::cout << maxElements << std::endl;*/
565     //temp.row(i) = Eigen::VectorXf::Map(&each[0], 10); //must match the column size
566     data = Eigen::MatrixXf(dataVec.size(), maxElements);
567 #pragma omp parallel for schedule(static) num_threads(8)
568     for (int i = 0; i < dataVec.size(); ++i)
569     {
570         const std::vector<float>& eachVec = dataVec[i]; //cached vector<float>
571         if(eachVec.size()==maxElements)
572         {
573             data.row(i) = Eigen::VectorXf::Map(&eachVec[0], maxElements);
574         }
575         else
576         {
577             const int& pointNum = eachVec.size()/3; //current vec point length
578             const int& totalNum = maxElements/3; //totally maximal point length
579             const int& segNum = pointNum-1;
580             const int& averageAdd = (totalNum-pointNum)/segNum; //average point on each segment
581
582         //# of segments with averageAdd+1 sampled points
583             const int& averageRes = (totalNum-pointNum)%segNum;
584
585             int segmentLength;
586             int currentPoint = 0;
587
588             Eigen::Vector3f meanLength, insertedPoint;
589             Eigen::Vector3f front, end;
590
591             int j;
592             for(j=0; j<segNum;j++)  //traverse all segments
593             {
594                 for(int k=0; k<3;k++)
595                     data(i,3*currentPoint+k) = eachVec[3*j+k];
596                 currentPoint++;
597                 if(j<segNum-averageRes)
598                     segmentLength = averageAdd;
599                 else
600                     segmentLength = averageAdd+1;
601                 if(segmentLength>=1)
602                 {
603                     front << eachVec[3*j], eachVec[3*j+1], eachVec[3*j+2];
604                     end << eachVec[3*j+3], eachVec[3*j+4], eachVec[3*j+5];
605                     meanLength = (end-front)/(segmentLength+1);
606                     for(int k=1; k<=segmentLength; k++)
607                     {
608                         insertedPoint = front+k*meanLength;
609                         for(int s=0; s<3;s++)
610                             data(i,3*currentPoint+s) = insertedPoint(s);
611                         currentPoint++;
612                     }
613                 }
614             }
615             assert(currentPoint==totalNum-1);
616             for(int k=0; k<3;k++)
617                 data(i,3*currentPoint+k) = eachVec[3*j+k];
618         }
619     }
620 }
```

4.8.2.28 void IOHandler::uniformArcSampling (MatrixXf & *data*, const std::vector< std::vector< float > > & *dataVec*, const int & *dimension*, const int & *maxElements*) [static]

Definition at line 472 of file IOHandler.cpp.

```
476 {
477     const int& numOfRows = dataVec.size();
478
479     /* assign memory for required matrix */
480     const int& totalSize = 3*maxElements;
481     data = Eigen::MatrixXf(numOfRows, totalSize);
482 #pragma omp parallel for schedule(static) num_threads(8)
```

```
483     /* compute total length of streamline and record each cumulative length information */
484     for(int i=0;i<numOfRows;++i)
485     {
486         float entireLength = 0.0, lineLength;
487         const std::vector<float>& eachVec = dataVec[i];
488         const int& vecSize = eachVec.size();
489         const int& lineNum = vecSize/3-1;
490         Eigen::Vector3f lineSeg;
491 
492         vector<float> pairwise(lineNum);
493         for(int j=0;j<lineNum;++j)
494         {
495             lineSeg(0)=eachVec[3*j+3]-eachVec[3*j];
496             lineSeg(1)=eachVec[3*j+4]-eachVec[3*j+1];
497             lineSeg(2)=eachVec[3*j+5]-eachVec[3*j+2];
498             lineLength = lineSeg.norm();
499             entireLength+=lineLength;
500             pairwise[j]=entireLength;
501         }
502 
503         float eachLength = entireLength/(maxElements-1);
504         Eigen::VectorXf row_i(data.row(i).size());
505 
506         /* insert starting vertex */
507         row_i(0)=eachVec[0];
508         row_i(1)=eachVec[1];
509         row_i(2)=eachVec[2];
510 
511         /* insert ending vertex */
512         row_i(totalSize-3)=eachVec[vecSize-3];
513         row_i(totalSize-2)=eachVec[vecSize-2];
514         row_i(totalSize-1)=eachVec[vecSize-1];
515 
516         float tempLength, tempRatio, ratioComplement;
517         int preLine, preVertex, postVertex;
518         for(int j=1;j<=maxElements-2;++j)
519         {
520             /* current length */
521             tempLength = j*eachLength;
522 
523             /* pre-index of coordinates */
524             preLine = std::lower_bound(pairwise.begin(), pairwise.end(), tempLength)-pairwise.begin()-1;
525 
526             /* locate vertex index */
527             preVertex = preLine+1;
528             postVertex = preVertex+1;
529 
530             /* use linear interpolation to generate new coordinates */
531             tempRatio = (tempLength-pairwise[preLine])/(pairwise[preLine+1]-pairwise[preLine]);
532             ratioComplement = 1.0-tempRatio;
533             row_i(3*j) = tempRatio*eachVec[3*postVertex]+ratioComplement*eachVec[3*preVertex];
534             row_i(3*j+1) = tempRatio*eachVec[3*postVertex+1]+ratioComplement*eachVec[3*preVertex+1];
535             row_i(3*j+2) = tempRatio*eachVec[3*postVertex+2]+ratioComplement*eachVec[3*preVertex+2];
536         }
537 
538         data.row(i) = row_i;
539     }
540 }
```

4.8.2.29 void IOHandler::writeGroup (const std::vector< int > & *group*, const std::vector< std::vector< float > > & *dataVec*) [static]

Definition at line 1092 of file IOHandler.cpp.

```
1094 {
1095     if(group.empty()||dataVec.empty())
1096         return;
1097     std::ofstream readme("../dataset/group",ios::out);
1098     if(!readme)
1099     {
1100         std::cout << "Error creating readme!" << std::endl;
1101         exit(1);
1102     }
1103     assert(group.size()==dataVec.size());
1104     for (int i = 0; i < group.size(); ++i)
1105     {
1106         readme << group[i] << std::endl;
1107     }
1108     readme.close();
1109 }
```

4.8.2.30 void IOHandler::writeGroupSize (const std::vector< int > & *storage*) [static]

Definition at line 1715 of file IOHandler.cpp.

```
1716 {
1717     if(storage.empty())
1718         return;
1719     std::ofstream readme("../dataset/README",ios::out | ios::app);
1720     if(!readme)
1721     {
1722         std::cout << "Error creating readme!" << std::endl;
1723         exit(1);
1724     }
1725     readme << "Final cluster size: " << storage.size() << std::endl;
1726     for (int i = 0; i < storage.size(); ++i)
1727     {
1728         readme << storage[i] << " ";
1729     }
1730     readme << std::endl;
1731     readme.close();
1732 }
```

4.8.2.31 void IOHandler::writeReadme (const double & *PCA_KMeans_delta*, const double & *KMeans_delta*) [static]

Definition at line 817 of file IOHandler.cpp.

```
819 {
820     std::ofstream readme("../dataset/README",ios::out | ios::app);
821     if(!readme)
822     {
823         std::cout << "Error creating readme!" << std::endl;
824         exit(1);
825     }
826     readme << "PCA_KMeans time elapse is " << PCA_KMeans_delta << " s." << std::endl
827            << "KMeans time elapse is " << KMeans_delta << " s." << std::endl;
828     readme << std::endl;
829     readme.close();
830 }
```

4.8.2.32 void IOHandler::writeReadme (const string & *comment*, const std::vector< float > & *sAverage*) [static]

Definition at line 839 of file IOHandler.cpp.

```
841 {
842     if(sAverage.empty())
843         return;
844     std::ofstream readme("../dataset/README",ios::out | ios::app);
845     if(!readme)
846     {
847         std::cout << "Error creating readme!" << std::endl;
848         exit(1);
849     }
850     readme << comment << std::endl;
851     for (int i = 0; i < sAverage.size(); ++i)
852     {
853         readme << sAverage[i] << std::endl;
854     }
855     readme << std::endl;
856     readme.close();
857 }
```

4.8.2.33 void IOHandler::writeReadme (const std::vector< string > & *timeName,* const std::vector< double > & *timeDiff,* const int & *cluster*) [static]

Definition at line 867 of file IOHandler.cpp.

```
870 {
871     if(timeName.empty()||timeDiff.empty())
872         return;
873     std::ofstream readme("../dataset/README",ios::out | ios::app);
874     if(!readme)
875     {
876         std::cout << "Error creating readme!" << std::endl;
877         exit(1);
878     }
879     assert(timeName.size()==timeDiff.size());
880     for (int i = 0; i < timeName.size(); ++i)
881     {
882         readme << timeName[i] << " is " << timeDiff[i] << " s." << std::endl;
883     }
884     readme << std::endl;
885     readme << "Preset cluster number in K-means is: " << cluster << std::endl;
886     readme << std::endl;
887     readme.close();
888 }
```

4.8.2.34 void IOHandler::writeReadme (const std::vector< string > & *timeName,* const std::vector< string > & *timeDiff,* const int & *cluster*) [static]

Definition at line 898 of file IOHandler.cpp.

```
901 {
902     if(timeName.empty()||timeDiff.empty())
903         return;
904     std::ofstream readme("../dataset/README",ios::out | ios::app);
905     if(!readme)
906     {
907         std::cout << "Error creating readme!" << std::endl;
908         exit(1);
909     }
910     assert(timeName.size()==timeDiff.size());
911     for (int i = 0; i < timeName.size(); ++i)
912     {
913         readme << timeName[i] << " " << timeDiff[i] << std::endl;
914     }
915     readme << "Preset cluster number in K-means is: " << cluster << std::endl;
916     readme << std::endl;
917     readme.close();
918 }
```

4.8.2.35 void IOHandler::writeReadme (const std::vector< ExtractedLine > & *closest,* const std::vector< ExtractedLine > & *furthest,* const int & *normOption*) [static]

Definition at line 928 of file IOHandler.cpp.

```
931 {
932     if(closest.empty()||furthest.empty())
933         return;
934     std::ofstream readme("../dataset/README",ios::out | ios::app);
935     if(!readme)
936     {
937         std::cout << "Error creating readme!" << std::endl;
938         exit(1);
939     }
940     const string& normStr = "Norm_"+to_string(normOption);
941     readme << std::endl;
942     readme << normStr+ " closest streamline set has " << closest.size() << " streamlines" << std::endl;
943     for (int i = 0; i < closest.size(); ++i)
944     {
```

```
945         readme << closest[i].lineNum << " ";
946     }
947     readme << std::endl;
948 
949     readme << std::endl;
950     readme << normStr+ " furthest streamline set has " << furthest.size() << " streamlines" << std::endl;
951     for (int i = 0; i < furthest.size(); ++i)
952     {
953         readme << furthest[i].lineNum << " ";
954     }
955     readme << std::endl;
956     readme.close();
957 }
```

4.8.2.36 void IOHandler::writeReadme (const std::vector< ExtractedLine > & *closest,* const std::vector< ExtractedLine > & *furthest*) [static]

Definition at line 966 of file IOHandler.cpp.

```
968 {
969     if(closest.empty()||furthest.empty())
970         return;
971     std::ofstream readme("../dataset/README",ios::out | ios::app);
972     if(!readme)
973     {
974         std::cout << "Error creating readme!" << std::endl;
975         exit(1);
976     }
977     readme << std::endl;
978     readme << "PCA closest streamline set has " << closest.size() << " streamlines" << std::endl;
979     for (int i = 0; i < closest.size(); ++i)
980     {
981         readme << closest[i].lineNum << " ";
982     }
983     readme << std::endl;
984 
985     readme << std::endl;
986     readme << "PCA furthest streamline set has " << furthest.size() << " streamlines" << std::endl;
987     for (int i = 0; i < furthest.size(); ++i)
988     {
989         readme << furthest[i].lineNum << " ";
990     }
991     readme << std::endl;
992     readme.close();
993 }
```

4.8.2.37 void IOHandler::writeReadme (const float & *closestAverage,* const float & *furthestAverage*) [static]

Definition at line 1026 of file IOHandler.cpp.

```
1027 {
1028     std::ofstream readme("../dataset/README",ios::out | ios::app);
1029     if(!readme)
1030     {
1031         std::cout << "Error creating readme!" << std::endl;
1032         exit(1);
1033     }
1034     readme << "The average rotation of closest is: " << closestAverage
1035            << ", of furthest is: " << furthestAverage << std::endl;
1036 }
```

4.8.2.38 void IOHandler::writeReadme (const string & *comments*) [static]

Definition at line 1695 of file IOHandler.cpp.

```
1696 {
1697     if(comments.empty())
1698         return;
1699     std::ofstream readme("../dataset/README",ios::out | ios::app);
1700     if(!readme)
1701     {
1702         std::cout << "Error creating readme!" << std::endl;
1703         exit(1);
1704     }
1705     readme << comments << std::endl;
1706     readme.close();
1707 }
```

4.8.2.39 void IOHandler::writeReadme (const float & *entropy,* const Silhouette & *sil,* const string & *norm_str*) [static]

Definition at line 1003 of file IOHandler.cpp.

```
1004 {
1005     std::ofstream readme("../dataset/README",ios::out | ios::app);
1006     if(!readme)
1007     {
1008         std::cout << "Error creating readme!" << std::endl;
1009         exit(1);
1010     }
1011     readme << norm_str << std::endl;
1012     readme << "The average silhouette: " << sil.sAverage
1013            << ", the gamma statistic is: " << sil.gammaStatistic
1014            << ", the entropy is: " << entropy
1015            << ", the DB index is: " << sil.dbIndex
1016            << std::endl;
1017 }
```

4.8.2.40 void IOHandler::writeReadMe (const float & *value,* const string & *dataSet,* const string & *clustering,* const string & *value_name*) [static]

Definition at line 1810 of file IOHandler.cpp.

```
1812 {
1813     std::ofstream out_file("../dataset/README", ios::out|ios::app);
1814     if (!out_file)
1815     {
1816         std::cout << "Error for creating README!" << std::endl;
1817         exit(1);
1818     }
1819     out_file << "----------------------------------------------------------------------" << std::endl;
1820     out_file << "The " << value_name << " of " << clustering << " on dataset "
1821              << dataSet << " is " << value << std::endl;
1822     out_file.close();
1823 }
```

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

- IOHandler.h
- IOHandler.cpp

4.9 MeanLine Struct Reference

`#include <IOHandler.h>`

Public Member Functions

- MeanLine (const std::vector< float > &minCenter, const int &cluster)

Public Attributes

- std::vector< float > minCenter
- int cluster

4.9.1 Detailed Description

Definition at line 50 of file IOHandler.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 MeanLine::MeanLine (const std::vector< float > & *minCenter,* const int & *cluster*) [inline]

Definition at line 54 of file IOHandler.h.

```
56             :minCenter(minCenter), cluster(cluster)
57     {}
```

4.9.3 Member Data Documentation

4.9.3.1 int MeanLine::cluster

Definition at line 53 of file IOHandler.h.

4.9.3.2 std::vector<float> MeanLine::minCenter

Definition at line 52 of file IOHandler.h.

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

- IOHandler.h

4.10 MetricPreparation Struct Reference

`#include <Metric.h>`

Public Member Functions

- MetricPreparation (const int &Row, const int &Column)
- MetricPreparation ()
- ~MetricPreparation ()
- void preprocessing (const Eigen::MatrixXf &data, const int &Row, const int &Column, const int &normOption)

Public Attributes

- std::vector< float > rotation
- std::vector< std::vector< float > > rotationSequence
- std::vector< MultiVariate > normalMultivariate
- std::vector< VectorXf > unitLength
- std::vector< std::vector< float > > pairwise
- std::vector< std::vector< float > > pairwiseNorm
- int row
- int column

4.10.1 Detailed Description

Definition at line 163 of file Metric.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 MetricPreparation::MetricPreparation (const int & *Row,* const int & *Column*) [inline]

Definition at line 181 of file Metric.h.

```
183     {
184         /* don't need to allocate redundant memory since the dataset size would be huge */
185         row = Row;
186         column = Column;
187     }
```

4.10.2.2 MetricPreparation::MetricPreparation ( ) [inline]

Definition at line 192 of file Metric.h.

```
193     {
194         row = column = 0;
195     }
```

4.10.2.3 MetricPreparation::~MetricPreparation ( ) [inline]

Definition at line 200 of file Metric.h.

```
201     {
202         row = column = -1;
203     }
```

4.10.3 Member Function Documentation

4.10.3.1 void MetricPreparation::preprocessing (const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* const int & *normOption*) [inline]

Definition at line 213 of file Metric.h.

```
217     {
218         switch(normOption)
219         {
220             case 2:
221             case 5:
222             case 8:
223                 {
224                     pairwise = std::vector<std::vector<float> >(Row, std::vector<float>(Column-3));
225                     pairwiseNorm = std::vector<std::vector<float> >(Row, std::vector<float>((
    Column-3)/3));
226                     computePairWise(data, Row, Column-3, pairwise,
    pairwiseNorm);
227                 }
228                 break;
229 
230             case 4:
231                 {
232                     /*  if rotation used for judge similarity difference, has to use pre-defined cache */
233                     computeMeanRotation(data, Row, Column,
    rotation);
234                 }
235                 break;
236 
237             /*  pre-defined cache for sequence mean and standard deviation */
238             case 3:
239                 {
240                     rotationSequence = std::vector<std::vector<float> >(Row,
    std::vector<float>(2));
241                     getRotationSequence(data, Row, Column,
    rotationSequence);
242                 }
243                 break;
244 
245             case 6:
246                 {
247                     normalMultivariate = std::vector<MultiVariate>(Row,
    MultiVariate());
248                     getNormalSequence(data, Row, Column,
    normalMultivariate);
249                 }
250                 break;
251 
252             case 7:
253                 {
254                     rotationSequence = std::vector<std::vector<float> >(Row,
    std::vector<float>(2));
255                     getFixedSequence(data, Row, Column,
    rotationSequence);
256                 }
257                 break;
258 
259             case 9:
260                 {
261                     normalMultivariate = std::vector<MultiVariate>(Row,
    MultiVariate());
262                     getUnnormalizedSequence(data, Row, Column,
    normalMultivariate);
263                 }
264                 break;
265 
266             case 10:
267                 {
268                     unitLength = std::vector<VectorXf >(Row, VectorXf(Column));
269                     getUnitDirection(data, Row, Column,
    unitLength);
270                 }
271                 break;
272 
273             case 14:
274                 {
275                     pairwise = std::vector<std::vector<float> >(Row, std::vector<float>(
    BIN_SIZE));
276                     getSignatureBin(data, Row, Column, pairwise);
277                 }
```

```
278                 break;
279
280             case 16:
281                 {
282                     pairwise = std::vector<std::vector<float> >(Row, std::vector<float>(2));
283                     getBundleEntropy(data, Row, Column, pairwise);
284                 }
285
286             default:
287                 break;
288         }
289     }
```

4.10.4 Member Data Documentation

4.10.4.1 int MetricPreparation::column

Definition at line 172 of file Metric.h.

4.10.4.2 std::vector<MultiVariate> MetricPreparation::normalMultivariate

Definition at line 167 of file Metric.h.

4.10.4.3 std::vector<std::vector<float> > MetricPreparation::pairwise

Definition at line 169 of file Metric.h.

4.10.4.4 std::vector<std::vector<float> > MetricPreparation::pairwiseNorm

Definition at line 170 of file Metric.h.

4.10.4.5 std::vector<float> MetricPreparation::rotation

Definition at line 165 of file Metric.h.

4.10.4.6 std::vector<std::vector<float> > MetricPreparation::rotationSequence

Definition at line 166 of file Metric.h.

4.10.4.7 int MetricPreparation::row

Definition at line 172 of file Metric.h.

4.10.4.8 std::vector<VectorXf > MetricPreparation::unitLength

Definition at line 168 of file Metric.h.

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

- Metric.h

4.11 MultiVariate Struct Reference

`#include <PreComputing.h>`

Public Member Functions

- MultiVariate ()
- ~MultiVariate ()

Public Attributes

- Matrix3f covariance
- Vector3f meanVec

4.11.1 Detailed Description

Definition at line 32 of file PreComputing.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 MultiVariate::MultiVariate ( ) [inline]

Definition at line 36 of file PreComputing.h.

```
37     {}
```

4.11.2.2 MultiVariate::~MultiVariate ( ) [inline]

Definition at line 38 of file PreComputing.h.

```
39     {}
```

4.11.3 Member Data Documentation

4.11.3.1 Matrix3f MultiVariate::covariance

Definition at line 34 of file PreComputing.h.

4.11.3.2 Vector3f MultiVariate::meanVec

Definition at line 35 of file PreComputing.h.

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

- PreComputing.h

4.12 Silhouette Class Reference

`#include <Silhouette.h>`

Public Member Functions

- Silhouette ()
- ~Silhouette ()
- void computeValue (const int &normOption, const MatrixXf &array, const int &Row, const int &Column, const std::vector< int > &group, const MetricPreparation &object, const int &groupNumber, const bool &isPBF)
- void computeValue (const int &normOption, const MatrixXf &array, const int &Row, const int &Column, const std::vector< int > &group, const MetricPreparation &object, const int &groupNumber, const bool &isPBF, const std::vector< vector< int > > &storage)
- void computeValue (const Eigen::MatrixXf &cArray, const std::vector< int > &group, const int &groupNo, const bool &isPBF)
- void reset ()

Public Attributes

- std::vector< float > sData
- std::vector< float > sCluster
- float sAverage
- float dbIndex
- float gammaStatistic = -1.0

Private Member Functions

- const float getDist (const int &first, const int &second, const MetricPreparation &object, const MatrixXf &array, const int &normOption)
- const float getA_i (const std::vector< std::vector< int > > &storage, const std::vector< int > &group, const MatrixXf &array, const int &index, const MetricPreparation &object, const int &normOption)
- const float getB_i (const std::vector< std::vector< int > > &storage, const std::vector< int > &group, const MatrixXf &array, const int &index, const MetricPreparation &object, const int &normOption)
- void getMatrixM (const Eigen::MatrixXf &cArray, const std::vector< int > &group, const std::vector< std::vector< int > > &storage, Eigen::MatrixXf &distM, Eigen::MatrixXf &idealDistM)
- void getMatrixM (const Eigen::MatrixXf &cArray, const std::vector< int > &group, const std::vector< std::vector< int > > &storage, Eigen::MatrixXf &idealDistM)
- const float getA_i (const std::vector< std::vector< int > > &storage, const std::vector< int > &group, const Eigen::MatrixXf &array, const int &index, const bool &isPBF, const Eigen::MatrixXf &distM)
- const float getB_i (const std::vector< std::vector< int > > &storage, const std::vector< int > &group, const Eigen::MatrixXf &array, const int &index, const bool &isPBF, const Eigen::MatrixXf &distM)
- void computeSilhouette (const Eigen::MatrixXf &array, const std::vector< int > &group, const bool &isPBF, const std::vector< std::vector< int > > &storage, const Eigen::MatrixXf &distM)
- void computeSilhouette (const Eigen::MatrixXf &array, const std::vector< int > &group, const std::vector< std::vector< int > > &storage, const MetricPreparation &object, const int &normOption)
- void computeDBIndex (const Eigen::MatrixXf &array, const std::vector< int > &group, const std::vector< std::vector< int > > &storage)
- void computeDBIndex (const Eigen::MatrixXf &array, const std::vector< int > &group, const std::vector< std::vector< int > > &storage, const MetricPreparation &object, const int &normOption)
- void computeGammaStatistic (const Eigen::MatrixXf &distM, const Eigen::MatrixXf &idealDistM)
- void computeGammaStatistic (const Eigen::MatrixXf &idealDistM)

4.12.1 Detailed Description

Definition at line 16 of file Silhouette.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Silhouette::Silhouette ( )

Definition at line 12 of file Silhouette.cpp.

```
13 {
14     sData = std::vector<float>();
15     sCluster = std::vector<float>();
16 }
```

4.12.2.2 Silhouette::~Silhouette ( )

Definition at line 22 of file Silhouette.cpp.

```
23 {
24     reset();
25 }
```

4.12.3 Member Function Documentation

4.12.3.1 void Silhouette::computeDBIndex (const Eigen::MatrixXf & *array*, const std::vector< int > & *group*, const std::vector< std::vector< int > > & *storage*) [private]

Definition at line 664 of file Silhouette.cpp.

```
667 {
668     dbIndex = 0.0;
669 
670     const int& groupNumber = storage.size();
671 
672     const int& Column = array.cols();
673 
674     /* calculated the projected-space cenroid */
675     Eigen::MatrixXf centroid(groupNumber, Column);
676 
677     /* average distance of all elements in cluster to its centroid */
678     Eigen::VectorXf averageDist(groupNumber);
679 
680 #pragma omp parallel for schedule(static) num_threads(8)
681     for(int i=0;i<groupNumber;++i)
682     {
683         Eigen::VectorXf tempCentroid = Eigen::VectorXf::Zero(Column);
684 
685         const std::vector<int>& clusterVec = storage[i];
686         const int& clusterSize = clusterVec.size();
687 
688         for(int j=0;j<clusterSize;++j)
689             tempCentroid+=array.row(clusterVec[j]);
690 
691         /* get the centroid coordinates */
692         centroid.row(i) = tempCentroid/clusterSize;
693 
694         float inClusterSum = 0.0, temp_dist;
695         for(int j=0;j<clusterSize;++j)
696         {
697             //inClusterSum+=getDisimilarity(centroid.row(i),array,clusterVec[j],normOption,object);
```

```
698             inClusterSum+=(array.row(clusterVec[j])-centroid.row(i)).norm();
699         }
700         averageDist(i) = inClusterSum/clusterSize;
701     }
702 
703 #pragma omp parallel num_threads(8)
704     {
705     #pragma omp for nowait
706         for (int i = 0; i < groupNumber; ++i)
707         {
708             float maxValue = (float)INT_MIN, ratioDist;
709             for (int j=0;j<groupNumber;++j)
710             {
711                 if(i==j)
712                     continue;
713                 ratioDist = (averageDist(i)+averageDist(j))/(centroid.row(i)-centroid.row(j)).norm();
714 
715                 if(maxValue<ratioDist)
716                     maxValue=ratioDist;
717             }
718 
719         #pragma omp critical
720             dbIndex += maxValue;
721         }
722     }
723     dbIndex/=groupNumber;
724 }
```

4.12.3.2 void Silhouette::computeDBIndex (const Eigen::MatrixXf & *array*, const std::vector< int > & *group*, const std::vector< std::vector< int > > & *storage*, const MetricPreparation & *object*, const int & *normOption*) [private]

Definition at line 736 of file Silhouette.cpp.

```
741 {
742     dbIndex = 0.0;
743 
744     const int& groupNumber = storage.size();
745 
746     const int& Column = array.cols();
747 
748     /* calculated the projected-space cenroid */
749     Eigen::MatrixXf centroid(groupNumber, Column);
750 
751     /* average distance of all elements in cluster to its centroid */
752     Eigen::VectorXf averageDist(groupNumber);
753 
754 #pragma omp parallel for schedule(static) num_threads(8)
755     for(int i=0;i<groupNumber;++i)
756     {
757         Eigen::VectorXf tempCentroid = Eigen::VectorXf::Zero(Column);
758 
759         const std::vector<int>& clusterVec = storage[i];
760         const int& clusterSize = clusterVec.size();
761 
762         for(int j=0;j<clusterSize;++j)
763             tempCentroid+=array.row(clusterVec[j]);
764 
765         /* get the centroid coordinates */
766         centroid.row(i) = tempCentroid/clusterSize;
767 
768         float inClusterSum = 0.0, temp_dist;
769         for(int j=0;j<clusterSize;++j)
770         {
771             inClusterSum+=getDisimilarity(centroid.row(i),array,clusterVec[j],normOption,
    object);
772         }
773         averageDist(i) = inClusterSum/clusterSize;
774     }
775 
776 #pragma omp parallel num_threads(8)
777     {
778     #pragma omp for nowait
779         for (int i = 0; i < groupNumber; ++i)
780         {
781             float maxValue = (float)INT_MIN, ratioDist, centDist;
782             for (int j=0;j<groupNumber;++j)
783             {
784                 if(i==j)
```

```
785                     continue;
786                 centDist = getDisimilarity(centroid.row(i), centroid.row(j), normOption,
    object);
787
788                 //ratioDist = (averageDist(i)+averageDist(j))/(centroid.row(i)-centroid.row(j)).norm();
789                 ratioDist = (averageDist(i)+averageDist(j))/centDist;
790
791                 if(maxValue<ratioDist)
792                     maxValue=ratioDist;
793             }
794
795         #pragma omp critical
796             dbIndex += maxValue;
797         }
798     }
799     dbIndex/=groupNumber;
800 }
```

4.12.3.3 void Silhouette::computeGammaStatistic (const Eigen::MatrixXf & *distM,* const Eigen::MatrixXf & *idealDistM*) [private]

Definition at line 809 of file Silhouette.cpp.

```
811 {
812     const int& Row = distM.rows();
813
814     const int& totalNum = Row*(Row-1)/2;
815     /* mean of values */
816     double u_1 = 0.0, u_2 = 0.0;
817
818     /* E(X*X) */
819     double s_1 = 0.0, s_2 = 0.0, numerator = 0.0;
820
821     for(int i=0;i<Row-1;++i)
822     {
823         for(int j=i+1;j<Row;++j)
824         {
825             /* update the mean u_1, u_2 */
826             u_1+=distM(i,j);
827             u_2+=idealDistM(i,j);
828
829             /* update the numerator */
830             numerator+=distM(i,j)*idealDistM(i,j);
831
832             /* update the deviation */
833             s_1+=distM(i,j)*distM(i,j);
834             s_2+=idealDistM(i,j)*idealDistM(i,j);
835         }
836     }
837
838     /* get mean of distM and idealDistM */
839     u_1/=totalNum;
840     u_2/=totalNum;
841
842     /* get numerator for the computing */
843     numerator-=totalNum*u_1*u_2;
844
845     /* get standard deviation */
846     s_1=sqrt(s_1/totalNum-u_1*u_1);
847     s_2=sqrt(s_2/totalNum-u_2*u_2);
848
849     if(std::isnan(s_1) || std::isnan(s_2))
850     {
851         std::cout << "standard deviation has nan error!" << std::endl;
852         exit(1);
853     }
854     gammaStatistic = float(numerator/s_1/s_2/totalNum);
855 }
```

4.12.3.4 void Silhouette::computeGammaStatistic (const Eigen::MatrixXf & *idealDistM*) [private]

Definition at line 863 of file Silhouette.cpp.

```
864 {
865     const int& Row = idealDistM.rows();
866 
867     const int& totalNum = Row*(Row-1)/2;
868 
869     /* mean of values */
870     double u_1 = 0.0, u_2 = 0.0;
871 
872     /* E(X*X) */
873     double s_1 = 0.0, s_2 = 0.0, numerator = 0.0;
874 
875     for(int i=0;i<Row-1;++i)
876     {
877         for(int j=i+1;j<Row;++j)
878         {
879             u_1+=distanceMatrix[i][j];
880             u_2+=idealDistM(i,j);
881 
882             numerator+=distanceMatrix[i][j]*idealDistM(i,j);
883 
884             s_1+=distanceMatrix[i][j]*distanceMatrix[i][j];
885             s_2+=idealDistM(i,j)*idealDistM(i,j);
886         }
887     }
888 
889     /* get mean of distM and idealDistM */
890     u_1/=totalNum;
891     u_2/=totalNum;
892 
893     /* get numerator for the computing */
894     numerator-=totalNum*u_1*u_2;
895 
896     /* get standard deviation */
897     s_1=sqrt(s_1/totalNum-u_1*u_1);
898     s_2=sqrt(s_2/totalNum-u_2*u_2);
899 
900     if(std::isnan(s_1) || std::isnan(s_2))
901     {
902         std::cout << "standard deviation has nan error!" << std::endl;
903         exit(1);
904     }
905     gammaStatistic = float(numerator/s_1/s_2/totalNum);
906 }
```

4.12.3.5 void Silhouette::computeSilhouette (const Eigen::MatrixXf & *array*, const std::vector< int > & *group*, const bool & *isPBF*, const std::vector< std::vector< int > > & *storage*, const Eigen::MatrixXf & *distM*) [private]

Definition at line 540 of file Silhouette.cpp.

```
545 {
546     const int& Row = array.rows();
547 
548 // compute Silhouette value for each data
549     float sSummation = 0;
550 
551 #pragma omp parallel num_threads(8)
552     {
553     #pragma omp for nowait
554         for (int i = 0; i < Row; ++i)
555         {
556             const float& a_i = getA_i(storage, group, array, i, isPBF, distM);
557             const float& b_i = getB_i(storage, group, array, i, isPBF, distM);
558 
559             float s_i;
560             if(abs(a_i-b_i)<1.0e-8)
561                 s_i = 0;
562             else if(a_i<b_i)
563                 s_i = 1 - a_i/b_i;
564             else
565                 s_i = b_i/a_i - 1;
566             if(std::isnan(s_i))
567             {
568                 std::cout << "Error for nan number!" << std::endl;
569                 exit(1);
570             }
571             sData[i] = s_i;
572 
573         #pragma omp critical
574             sSummation += s_i;
```

```
575         }
576     }
577     sAverage = sSummation/group.size();
578 
579 #pragma omp parallel for schedule(static) num_threads(8)
580     for (int i = 0; i < sCluster.size(); ++i)
581     {
582         float& eachCluster = sCluster[i];
583         eachCluster = 0;
584         const std::vector<int>& clustSet = storage[i];
585         for (int j = 0; j < clustSet.size(); ++j)
586         {
587             eachCluster += sData[clustSet[j]];
588         }
589         eachCluster/=clustSet.size();
590     }
591 }
```

4.12.3.6 void Silhouette::computeSilhouette (const Eigen::MatrixXf & *array*, const std::vector< int > & *group*, const std::vector< std::vector< int > > & *storage*, const MetricPreparation & *object*, const int & *normOption*) [private]

Definition at line 603 of file Silhouette.cpp.

```
608 {
609     // compute Silhouette value for each data
610     float sSummation = 0;
611 
612 #pragma omp parallel num_threads(8)
613     {
614     #pragma omp for nowait
615         for (int i = 0; i < group.size(); ++i)
616         {
617             if(group[i]<0)
618                 continue;
619             const float& a_i = getA_i(storage, group, array, i, object, normOption);
620             const float& b_i = getB_i(storage, group, array, i, object, normOption);
621 
622             float s_i;
623             if(abs(a_i-b_i)<1.0e-8)
624                 s_i = 0;
625             else if(a_i<b_i)
626                 s_i = 1 - a_i/b_i;
627             else
628                 s_i = b_i/a_i - 1;
629             if(std::isnan(s_i))
630             {
631                 std::cout << "Error for nan number!" << std::endl;
632                 exit(1);
633             }
634             sData[i] = s_i;
635 
636         #pragma omp critical
637             sSummation += s_i;
638         }
639     }
640     sAverage = sSummation/(group.size());
641 
642 #pragma omp parallel for schedule(static) num_threads(8)
643     for (int i = 0; i < sCluster.size(); ++i)
644     {
645         float& eachCluster = sCluster[i];
646         eachCluster = 0;
647         const std::vector<int>& clustSet = storage[i];
648         for (int j = 0; j < clustSet.size(); ++j)
649         {
650             eachCluster += sData[clustSet[j]];
651         }
652         eachCluster/=clustSet.size();
653     }
654 }
```

4.12.3.7 void Silhouette::computeValue (const int & *normOption*, const MatrixXf & *array*, const int & *Row*, const int & *Column*, const std::vector< int > & *group*, const MetricPreparation & *object*, const int & *groupNumber*, const bool & *isPBF*)

Definition at line 51 of file Silhouette.cpp.

```
59 {
60
61     std::vector<std::vector<int> > storage(groupNumber, std::vector<int>());
62
63     //whether some group marked as -1 noise or not
64     int noise = 0;
65     for (int i = 0; i < group.size(); ++i)
66     {
67         if(group[i]<0)
68         {
69             ++noise;
70             continue;
71         }
72         else
73             storage[group[i]].push_back(i);
74     }
75     // compute the value
76     computeValue(normOption, array, Row, Column, group, object, groupNumber, isPBF, storage);
77
78 }
```

4.12.3.8 void Silhouette::computeValue (const int & *normOption*, const MatrixXf & *array*, const int & *Row*, const int & *Column*, const std::vector< int > & *group*, const MetricPreparation & *object*, const int & *groupNumber*, const bool & *isPBF*, const std::vector< vector< int > > & *storage*)

Definition at line 161 of file Silhouette.cpp.

```
170 {
171     sData.clear();
172     sCluster.clear();
173     sData = std::vector<float>(Row,0);
174     assert(Row==group.size());
175
176     //groupNumber doesn't include noise group
177     sCluster = std::vector<float>(groupNumber, 0);
178     /* if the silhouett computing is not for PBF dataset, then would use distanceMatrix */
179     Eigen::MatrixXf idealDistM;
180     if(!isPBF)
181     {
182         getMatrixM(array,group,storage,idealDistM);
183     }
184
185     std::cout << "Compute silhouette..." << std::endl;
186     /* compute silhouette value */
187     computeSilhouette(array, group, storage, object, normOption);
188     std::cout << "Silhouette is " << sAverage << std::endl;
189
190     std::cout << "Compute DB index..." << std::endl;
191     /* compute DB index */
192     computeDBIndex(array, group, storage, object, normOption);
193     std::cout << "DB index is " << dbIndex << std::endl;
194
195     /* compute Gamma statistic for distM and idealDistM */
196     if(!isPBF)
197     {
198         std::cout << "Compute gamma statistics..." << std::endl;
199         computeGammaStatistic(idealDistM);
200         std::cout << "Gamma statistics is " << gammaStatistic << std::endl;
201         /* garbage collection for eigen::matrix */
202         idealDistM.resize(0,0);
203     }
204 }
```

4.12.3.9 void Silhouette::computeValue (const Eigen::MatrixXf & *cArray*, const std::vector< int > & *group*, const int & *groupNo*, const bool & *isPBF*)

Definition at line 89 of file Silhouette.cpp.

```
93 {
94     sData.clear();
95     sCluster.clear();
96 
97     /* get Row and Column information */
98     const int& Row = array.rows();
99     const int& Column = array.cols();
100 
101     sData = std::vector<float>(Row,0);
102 
103     /* assert information */
104     assert(Row==group.size());
105 
106     std::vector<std::vector<int> > storage(groupNumber, std::vector<int>());
107     for (int i = 0; i < group.size(); ++i)
108     {
109         storage[group[i]].push_back(i);
110     }
111     /* record labeling information */
112     generateGroups(storage);
113 
114     //groupNumber doesn't include noise group
115     sCluster = std::vector<float>(groupNumber, 0);
116 
117     /* if the silhouett computing is not for PBF dataset, then would use distanceMatrix */
118     Eigen::MatrixXf distM, idealDistM;
119     if(!isPBF)  // not from PBF, so the distance matrix can be assigned
120     {
121         getMatrixM(array,group,storage,distM,idealDistM);
122     }
123 
124     std::cout << "Compute silhouette..." << std::endl;
125     /* compute silhouette value */
126     computeSilhouette(array, group, isPBF, storage, distM);
127 
128     std::cout << "silhouette is " << sAverage << std::endl;
129 
130     std::cout << "Compute DB index..." << std::endl;
131     /* compute DB index */
132     computeDBIndex(array, group, storage);
133     std::cout << "DB index is " << dbIndex << std::endl;
134     /* compute Gamma statistic for distM and idealDistM */
135     if(!isPBF)  // only compute Gamma statistics for non-PBF data set
136     {
137         std::cout << "Compute gamma statistics..." << std::endl;
138         computeGammaStatistic(distM,idealDistM);
139         std::cout << "Gamma statistics is " << gammaStatistic << std::endl;
140         /* garbage collection for eigen::matrix */
141         distM.resize(0,0);
142         idealDistM.resize(0,0);
143     }
144 
145 }
```

4.12.3.10 const float Silhouette::getA_i (const std::vector< std::vector< int > > & *storage*, const std::vector< int > & *group*, const MatrixXf & *array*, const int & *index*, const **MetricPreparation** & *object*, const int & *normOption*) [private]

Definition at line 217 of file Silhouette.cpp.

```
223 {
224     const std::vector<int>& clusterSet = storage[group[index]];
225     float inClusterDist = 0.0, dist;
226     for (int j = 0; j < clusterSet.size(); ++j)
227     {
228         if(clusterSet[j]!=index)
229         {
230             if(distanceMatrix)
231             {
232                 dist = distanceMatrix[index][clusterSet[j]];
```

```
233             }
234             else
235             {
236                 dist = getDist(index, clusterSet[j], object, array, normOption);
237             }
238             inClusterDist += dist;
239         }
240     }
241     if(std::isnan(inClusterDist))
242     {
243         std::cout << "a_i has nan error! " << inClusterDist << std::endl;
244         exit(1);
245     }
246     float a_i;
247     if(clusterSet.size()==1)
248         a_i = 0;
249     else
250         a_i = inClusterDist/(clusterSet.size()-1);
251     return a_i;
252 }
```

4.12.3.11 const float Silhouette::getA_i (const std::vector< std::vector< int > > & *storage*, const std::vector< int > & *group*, const Eigen::MatrixXf & *array*, const int & *index*, const bool & *isPBF*, const Eigen::MatrixXf & *distM*) [private]

Definition at line 435 of file Silhouette.cpp.

```
441 {
442     const std::vector<int>& clusterSet = storage[group[index]];
443     float inClusterDist = 0.0;
444
445     int candidate;
446     for (int j = 0; j < clusterSet.size(); ++j)
447     {
448         candidate = clusterSet[j];
449         if(candidate!=index)
450         {
451             if(!isPBF)
452                 inClusterDist += distM(index,candidate);
453             else
454                 inClusterDist += (array.row(index)-array.row(candidate)).norm();
455         }
456     }
457     if(std::isnan(inClusterDist))
458     {
459         std::cout << "a_i has nan error!" << std::endl;
460         exit(1);
461     }
462     float a_i;
463     if(clusterSet.size()==1)
464         a_i = 0;
465     else
466         a_i = inClusterDist/(clusterSet.size()-1);
467     return a_i;
468 }
```

4.12.3.12 const float Silhouette::getB_i (const std::vector< std::vector< int > > & *storage*, const std::vector< int > & *group*, const MatrixXf & *array*, const int & *index*, const **MetricPreparation** & *object*, const int & *normOption*) [private]

Definition at line 265 of file Silhouette.cpp.

```
271 {
272     float outClusterDist = FLT_MAX, perClusterDist = 0;
273     std::vector<int> outClusterSet;
274     if(storage.size()==1)
275         return 0;
276     for (int j = 0; j < storage.size(); ++j) //j is group no.
277     {
278         if(j!=group[index]) //the other cluster
```

```
279         {
280             outClusterSet = storage[j];//get integer list of this group
281             perClusterDist = 0;
282             for (int k = 0; k < outClusterSet.size(); ++k)
283             {
284                 if(distanceMatrix)
285                     perClusterDist+=distanceMatrix[index][outClusterSet[k]];
286                 else
287                     perClusterDist += getDist(index, outClusterSet[k], object, array, normOption);
288             }
289             if(perClusterDist<0)
290             {
291                 std::cout << "Error for negative distance!" << std::endl;
292                 exit(1);
293             }
294             perClusterDist/=outClusterSet.size();
295             if(outClusterDist>perClusterDist)
296                 outClusterDist=perClusterDist;
297         }
298     }
299     return outClusterDist;
300 }
```

4.12.3.13 const float Silhouette::getB_i (const std::vector< std::vector< int > > & *storage*, const std::vector< int > & *group*, const Eigen::MatrixXf & *array*, const int & *index*, const bool & *isPBF*, const Eigen::MatrixXf & *distM*) [private]

Definition at line 481 of file Silhouette.cpp.

```
487 {
488     float outClusterDist = FLT_MAX, perClusterDist = 0;
489     std::vector<int> outClusterSet;
490
491     int candidate, outClusterSize;
492     for (int j = 0; j < storage.size(); ++j) //j is group no.
493     {
494         if(j!=group[index]) //the other cluster
495         {
496             outClusterSet = storage[j];//get integer list of this group
497             perClusterDist = 0;
498
499             outClusterSize = outClusterSet.size();
500
501             /* empty cluster which is erroneous */
502             if(outClusterSize==0)
503             {
504                 std::cout << "Found empty clusters!" << std::endl;
505                 exit(1);
506             }
507
508             /* get average dist to all elements inside the cluster */
509             for (int k = 0; k < outClusterSize; ++k)
510             {
511                 candidate = outClusterSet[k];
512                 if(!isPBF)
513                     perClusterDist+=distM(index, candidate);
514                 else
515                     perClusterDist += (array.row(index)-array.row(candidate)).norm();
516             }
517             if(perClusterDist<0)
518             {
519                 std::cout << "Error for negative distance!" << std::endl;
520                 exit(1);
521             }
522             perClusterDist/=outClusterSize;
523             if(outClusterDist>perClusterDist)
524                 outClusterDist=perClusterDist;
525         }
526     }
527     return outClusterDist;
528 }
```

4.12.3.14 const float Silhouette::getDist (const int & *first*, const int & *second*, const MetricPreparation & *object*, const MatrixXf & *array*, const int & *normOption*) [private]

Definition at line 312 of file Silhouette.cpp.

```
317 {
318     float distance = getDisimilarity(array.row(first),array.row(second),
319                                      first,second,normOption,object);
320     if(distance<0)
321     {
322         std::cout << "Error for negative distance!" << std::endl;
323         exit(1);
324     }
325     if(isnan(distance) || isinf(distance))
326     {
327         std::cout << "Error for distance value that is nan or inf!" << std::endl;
328         exit(1);
329     }
330     return distance;
331 }
```

4.12.3.15 void Silhouette::getMatrixM (const Eigen::MatrixXf & *cArray*, const std::vector< int > & *group*, const std::vector< std::vector< int > > & *storage*, Eigen::MatrixXf & *distM*, Eigen::MatrixXf & *idealDistM*) [private]

Definition at line 343 of file Silhouette.cpp.

```
348 {
349     const int& Row = cArray.rows();
350     const int& Column = cArray.cols();
351
352     /* resize matrix size */
353     distM = Eigen::MatrixXf::Zero(Row,Row);
354     idealDistM = Eigen::MatrixXf::Constant(Row,Row,1.0);
355
356     /* of course here is not related to distanceMatrix which is a global variable */
357 #pragma omp parallel for schedule(static) num_threads(8)
358     for(int i=0;i<Row;++i)
359     {
360         for(int j=0;j<Row;++j)
361         {
362             if(i==j)
363                 continue;
364             /* assign the Euclidean distance of cArray */
365             distM(i,j)=(cArray.row(i)-cArray.row(j)).norm();
366         }
367     }
368
369     const int& groupNumber = storage.size();
370 #pragma omp parallel for schedule(static) num_threads(8)
371     for(int i=0;i<groupNumber;++i)
372     {
373         /* if i and j in same cluster, then set it to be zero */
374         const std::vector<int>& eachVec = storage[i];
375         const int& eachSize = eachVec.size();
376         for(int j=0;j<eachSize;++j)
377         {
378             for(int k=0;k<eachSize;++k)
379             {
380                 idealDistM(eachVec[j],eachVec[k]) = 0;
381             }
382         }
383     }
384 }
```

4.12.3.16 void Silhouette::getMatrixM (const Eigen::MatrixXf & *cArray,* const std::vector< int > & *group,* const std::vector< std::vector< int > > & *storage,* Eigen::MatrixXf & *idealDistM*) [private]

Definition at line 395 of file Silhouette.cpp.

```
399 {
400     const int& Row = cArray.rows();
401     const int& Column = cArray.cols();
402
403     /* resize matrix size */
404     idealDistM = Eigen::MatrixXf::Constant(Row,Row,1.0);
405
406     /* find the ideal matrix inside which the idealDistM(i,j)==0 only if i and j in same cluster */
407     const int& groupNumber = storage.size();
408 #pragma omp parallel for schedule(static) num_threads(8)
409     for(int i=0;i<groupNumber;++i)
410     {
411         /* if i and j in same cluster, then set it to be zero */
412         const std::vector<int>& eachVec = storage[i];
413         const int& eachSize = eachVec.size();
414         for(int j=0;j<eachSize;++j)
415         {
416             for(int k=0;k<eachSize;++k)
417             {
418                 idealDistM(eachVec[j],eachVec[k]) = 0;
419             }
420         }
421     }
422 }
```

4.12.3.17 void Silhouette::reset ( )

Definition at line 31 of file Silhouette.cpp.

```
32 {
33     sData.clear();
34     sCluster.clear();
35     sAverage = 0;
36 }
```

4.12.4 Member Data Documentation

4.12.4.1 float Silhouette::dbIndex

Definition at line 38 of file Silhouette.h.

4.12.4.2 float Silhouette::gammaStatistic = -1.0

Definition at line 43 of file Silhouette.h.

4.12.4.3 float Silhouette::sAverage

Definition at line 33 of file Silhouette.h.

4.12.4.4 std::vector<float> Silhouette::sCluster

Definition at line 28 of file Silhouette.h.

4.12.4.5 std::vector<float> Silhouette::sData

Definition at line 23 of file Silhouette.h.

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

- Silhouette.h
- Silhouette.cpp

4.13 StringQuery Struct Reference

`#include <IOHandler.h>`

Public Member Functions

- StringQuery ()
- StringQuery (const int &index, const std::vector< int > &neighbor)

Public Attributes

- int index
- std::vector< int > neighbor

4.13.1 Detailed Description

Definition at line 65 of file IOHandler.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 StringQuery::StringQuery ( ) [inline]

Definition at line 69 of file IOHandler.h.

```
70     {   }
```

4.13.2.2 StringQuery::StringQuery (const int & *index*, const std::vector< int > & *neighbor*) [inline]

Definition at line 71 of file IOHandler.h.

```
72                                          :
73                 index(index), neighbor(neighbor)
74     {   }
```

4.13.3 Member Data Documentation

4.13.3.1 int StringQuery::index

Definition at line 67 of file IOHandler.h.

4.13.3.2 std::vector<int> StringQuery::neighbor

Definition at line 68 of file IOHandler.h.

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

- IOHandler.h

4.14 ValidityMeasurement Class Reference

`#include <ValidityMeasurement.h>`

Public Member Functions

- ValidityMeasurement ()
- virtual ~ValidityMeasurement ()
- void computeValue (const int &normOption, const MatrixXf &array, const std::vector< int > &group, const MetricPreparation &object, const bool &isPBF)
- void computeValue (const MatrixXf &array, const std::vector< int > &group)

Public Attributes

- float f_c

Private Member Functions

- void getMST_Parent_Node (std::tuple< float, float, float > &values, const std::vector< int > &clusterNode, const MetricPreparation &object, const int &normOption, const MatrixXf &array, const bool &isPBF)
- void getMST_Parent_Node (std::tuple< float, float, float > &values, const std::vector< int > &clusterNode, const MatrixXf &array)
- const float get_Sc_by_range (const bool &isPBF, const Eigen::MatrixXf &distM, const std::vector< int > &clusterNode, const float &rangeValue, const MetricPreparation &object, const int &normOption, const MatrixXf &array, int &index)
- const float get_Sc_by_range (const Eigen::MatrixXf &distM, const std::vector< int > &clusterNode, const float &rangeValue, const MatrixXf &array, int &index)

Private Attributes

- float min_Sc
- float max_Sc

4.14.1 Detailed Description

Definition at line 23 of file ValidityMeasurement.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 ValidityMeasurement::ValidityMeasurement ( )

Definition at line 15 of file ValidityMeasurement.cpp.

```
15                                          {
16     // TODO Auto-generated constructor stub
17
18 }
```

4.14.2.2 ValidityMeasurement::~ValidityMeasurement ( ) [virtual]

Definition at line 24 of file ValidityMeasurement.cpp.

```
24                                           {
25     // TODO Auto-generated destructor stub
26 }
```

4.14.3 Member Function Documentation

4.14.3.1 void ValidityMeasurement::computeValue (const int & *normOption*, const MatrixXf & *array*, const std::vector< int > & *group*, const MetricPreparation & *object*, const bool & *isPBF*)

Definition at line 38 of file ValidityMeasurement.cpp.

```
40 {
41     std::cout << "Compute validity measurement..." << std::endl;
42     // get how many different groups it totally has
43     int max_group = -1;
44     const int& num_node = group.size();
45     for(int i=0; i<num_node; ++i)
46     {
47         if(group[i]==-1)
48             continue;
49         max_group = std::max(group[i], max_group);
50     }
51     max_group+=1;
52
53     std::vector<std::vector<int> > storage(max_group);
54
55     for(int i=0; i<num_node; ++i)
56     {
57         if(group[i]==-1)
58             continue;
59         storage[group[i]].push_back(i);
60     }
61
62     std::vector<std::tuple<float, float, float> > measureVec(max_group);
63
64     for(int i=0; i<max_group; ++i)
65     {
66         getMST_Parent_Node(measureVec[i], storage[i], object, normOption, array, isPBF);
67     }
68
69     float minSc = 0, maxSc = 0, aver_sigma = 0, std_sigma = 0, std_variance;
70     for(int i=0; i<max_group; ++i)
```

```
71      {
72          // get the min Sc by summation
73          minSc+=std::get<1>(measureVec[i]);
74          // get the max Sc by summation
75          maxSc+=std::get<2>(measureVec[i]);
76          std_variance = std::get<0>(measureVec[i]);
77          // get the average variance and standard variation of variance
78          aver_sigma+=std_variance;
79          std_sigma+=std_variance*std_variance;
80      }
81      aver_sigma/=max_group;
82      std_sigma = std_sigma/float(max_group-1)-float(max_group)/float(max_group-1)*aver_sigma*aver_sigma;
83
84      if(std_sigma<1.0E-10)
85      {
86          std_sigma=1.0E-10;
87      }
88      std_sigma=sqrt(std_sigma);
89
90      float h_DDc = aver_sigma+std_sigma;
91
92      minSc/=float(max_group);
93      maxSc/=float(max_group);
94
95      // compute g1_Sc
96      float g1_Sc = (1.0-minSc)*(1.0-maxSc);
97      if(g1_Sc<0)
98      {
99          std::cout << "Negative number for g1_Sc computation!" << std::endl;
100      }
101      g1_Sc = aver_sigma*sqrt(g1_Sc);
102
103      // compute g2_Sc
104      float g2_Sc = minSc*maxSc;
105      if(g2_Sc<0)
106      {
107          std::cout << "Negative number for g2_Sc computation!" << std::endl;
108      }
109      g2_Sc = aver_sigma/sqrt(g2_Sc);
110
111      // compute g_Sc
112      float g_Sc = (sqrt(g1_Sc*g2_Sc)+(g1_Sc+g2_Sc)/2.0)/2.0;
113
114      // compoute f_c
115      f_c = h_DDc*g_Sc;
116
117      if(isnan(f_c) || isinf(f_c))
118      {
119          std::cout << "Error for f_c to have inf or nan values!" << std::endl;
120      }
121
122      /* normalization of validity measurement */
123      float min_dist = FLT_MAX, max_dist = -1.0;
124      const int& row = array.rows();
125  #pragma omp parallel for reduction(min:min_dist) num_threads(8)
126      for(int i=0; i<row; ++i)
127      {
128          for(int j=0; j<row; ++j)
129          {
130              if(i==j)
131                  continue;
132              float dist;
133              if(distanceMatrix)
134                  dist = distanceMatrix[i][j];
135              else
136                  dist = getDisimilarity(array, i, j, normOption, object);
137              min_dist = std::min(min_dist, dist);
138          }
139      }
140
141  #pragma omp parallel for reduction(max:max_dist) num_threads(8)
142      for(int i=0; i<row; ++i)
143      {
144          for(int j=0; j<row; ++j)
145          {
146              if(i==j)
147                  continue;
148              float dist;
149              if(distanceMatrix)
150                  dist = distanceMatrix[i][j];
151              else
152                  dist = getDisimilarity(array, i, j, normOption, object);
153              max_dist = std::max(max_dist, dist);
154          }
155      }
156      std::cout << "min dist is " << min_dist << ", and max is " << max_dist << std::endl;
157      f_c/=(max_dist-min_dist)*(max_dist-min_dist);
```

```
158
159     // try to place the distance range into the file for further batch processing
160     std::ofstream fout("../dataset/dist_range", ios::app);
161     if(fout.fail())
162     {
163         std::cout << "Error for file operation!" << std::endl;
164         exit(1);
165     }
166
167     fout << "For norm " << normOption << ", min is " << min_dist << ", max is " << max_dist << ", and" <<
168             " (max - min) is " << (max_dist-min_dist) << std::endl;
169     fout << std::endl;
170     fout.close();
171
172     std::cout << "Validity measurement is " << f_c << std::endl;
173 }
```

4.14.3.2 void ValidityMeasurement::computeValue (const MatrixXf & *array*, const std::vector< int > & *group*)

Definition at line 182 of file ValidityMeasurement.cpp.

```
183 {
184     std::cout << "Compute validity measurement..." << std::endl;
185     // get how many different groups it totally has
186     int max_group = -1;
187     const int& num_node = group.size();
188     for(int i=0; i<num_node; ++i)
189     {
190         if(group[i]==-1)
191             continue;
192         max_group = std::max(group[i], max_group);
193     }
194     max_group+=1;
195
196     std::vector<std::vector<int> > storage(max_group);
197     for(int i=0; i<num_node; ++i)
198     {
199         if(group[i]==-1)
200             continue;
201         storage[group[i]].push_back(i);
202     }
203
204     std::vector<std::tuple<float, float, float> > measureVec(max_group);
205
206     for(int i=0; i<max_group; ++i)
207     {
208         getMST_Parent_Node(measureVec[i], storage[i], array);
209     }
210
211     float minSc = 0, maxSc = 0, aver_sigma = 0, std_sigma = 0, std_variance;
212     for(int i=0; i<max_group; ++i)
213     {
214         // get the min Sc by summation
215         minSc+=std::get<1>(measureVec[i]);
216         // get the max Sc by summation
217         maxSc+=std::get<2>(measureVec[i]);
218         std_variance = std::get<0>(measureVec[i]);
219         // get the average variance and standard variation of variance
220         aver_sigma+=std_variance;
221         std_sigma+=std_variance*std_variance;
222     }
223     aver_sigma/=float(max_group);
224     std_sigma = std_sigma/float(max_group-1)-float(max_group)/float(max_group-1)*aver_sigma*aver_sigma;
225     if(std_sigma<1.0E-10)
226     {
227         std_sigma = 1.0E-10;
228     }
229     else
230         std_sigma=sqrt(std_sigma);
231
232     float h_DDc = aver_sigma+std_sigma;
233
234     minSc/=float(max_group);
235     maxSc/=float(max_group);
236
237     // compute g1_Sc
238     float g1_Sc = (1.0-minSc)*(1.0-maxSc);
239     if(g1_Sc<0)
240     {
241         std::cout << "Negative number for g1_Sc computation!" << std::endl;
```

```
242     }
243     g1_Sc = aver_sigma*sqrt(g1_Sc);
244 
245     // compute g2_Sc
246     float g2_Sc = minSc*maxSc;
247     if(g2_Sc<0)
248     {
249         std::cout << "Negative number for g2_Sc computation!" << std::endl;
250     }
251     g2_Sc = aver_sigma/sqrt(g2_Sc);
252 
253     // compute g_Sc
254     float g_Sc = (sqrt(g1_Sc*g2_Sc)+(g1_Sc+g2_Sc)/2.0)/2.0;
255 
256     // compoute f_c
257     f_c = h_DDc*g_Sc;
258 
259     if(isnan(f_c) || isinf(f_c))
260     {
261         std::cout << "Error for f_c to have inf or nan values!" << std::endl;
262     }
263 
264     /* normalization of validity measurement */
265     float min_dist = FLT_MAX, max_dist = -1.0;
266     const int& row = array.rows();
267 #pragma omp parallel for reduction(min:min_dist) num_threads(8)
268     for(int i=0; i<row; ++i)
269     {
270         for(int j=0; j<row; ++j)
271         {
272             if(i==j)
273                 continue;
274             min_dist = std::min(min_dist, (array.row(i)-array.row(j)).norm());
275         }
276     }
277 
278 #pragma omp parallel for reduction(max:max_dist) num_threads(8)
279     for(int i=0; i<row; ++i)
280     {
281         for(int j=0; j<row; ++j)
282         {
283             if(i==j)
284                 continue;
285             max_dist = std::max(max_dist, (array.row(i)-array.row(j)).norm());
286         }
287     }
288     std::cout << "min dist is " << min_dist << ", and max is " << max_dist << std::endl;
289     f_c/=(max_dist-min_dist)*(max_dist-min_dist);
290 
291     // try to place the distance range into the file for further batch processing
292     std::ofstream fout("../dataset/dist_range", ios::app);
293     if(fout.fail())
294     {
295         std::cout << "Error for file operation!" << std::endl;
296         exit(1);
297     }
298 
299     fout << "For PCA, min is " << min_dist << ", max is " << max_dist << ", and" <<
300             " (max - min) is " << (max_dist-min_dist) << std::endl;
301     fout << std::endl;
302     fout.close();
303 
304     std::cout << "Validity measurement is " << f_c << std::endl;
305 }
```

4.14.3.3 const float ValidityMeasurement::get_Sc_by_range (const bool & *isPBF*, const Eigen::MatrixXf & *distM*, const std::vector< int > & *clusterNode*, const float & *rangeValue*, const MetricPreparation & *object*, const int & *normOption*, const MatrixXf & *array*, int & *index*) [private]

Definition at line 570 of file ValidityMeasurement.cpp.

```
574 {
575     const int& node_number = clusterNode.size();
576     float result = 0.0;
577 
578     index = 0;
579     int inside_whole, inside_cluster;
580     for(int i=0; i<node_number; ++i)
581     {
```

```
582         inside_whole = 0, inside_cluster = 0;
583         // count how many points in N_epsi(P_i) for the whole dataset
584     #pragma omp parallel num_threads(8)
585         {
586         #pragma omp for nowait
587             for(int j=0; j<array.rows(); ++j)
588             {
589                 // don't want to handle duplicates and itself
590                 if(clusterNode[i]==j)
591                     continue;
592                 float dist;
593                 if(distanceMatrix)
594                     dist = distanceMatrix[clusterNode[i]][j];
595                 else
596                     dist = getDisimilarity(array, clusterNode[i], j, normOption, object);
597
598             #pragma omp critical
599                 {
600                     if(dist<=rangeValue)
601                     {
602                         ++inside_whole;
603                     }
604                 }
605             }
606
607         }
608
609         // count how many points in N_epsi(P_i) for current cluster
610     #pragma omp parallel num_threads(8)
611         {
612         #pragma omp for nowait
613             for(int j=0; j<node_number; ++j)
614             {
615                 // don't want to handle duplicates and itself
616                 if(i==j)
617                     continue;
618                 float dist;
619                 if(isPBF)
620                     dist = distM(i,j);
621                 else
622                     dist = distanceMatrix[clusterNode[i]][clusterNode[j]];
623
624             #pragma omp critical
625                 if(dist<=rangeValue)
626                     ++inside_cluster;
627             }
628
629         }
630         assert(inside_cluster<=inside_whole);
631         if(inside_whole==0)
632             continue;
633         ++index;
634         result+=float(inside_cluster)/float(inside_whole);
635     }
636     return result;
637 }
```

4.14.3.4 const float ValidityMeasurement::get_Sc_by_range (const Eigen::MatrixXf & *distM*, const std::vector< int > & *clusterNode*, const float & *rangeValue*, const MatrixXf & *array*, int & *index*) [private]

Definition at line 649 of file ValidityMeasurement.cpp.

```
651 {
652     const int& node_number = clusterNode.size();
653     float result = 0.0;
654
655     index = 0;
656     int inside_whole, inside_cluster;
657     for(int i=0; i<node_number; ++i)
658     {
659         inside_whole = 0, inside_cluster = 0;
660         // count how many points in N_epsi(P_i) for the whole dataset
661     #pragma omp parallel num_threads(8)
662         {
663         #pragma omp for nowait
664             for(int j=0; j<array.rows(); ++j)
665             {
666                 // don't want to handle duplicates and itself
667                 if(i==j)
```

```
668                     continue;
669                 float dist = (array.row(clusterNode[i])-array.row(j)).norm();
670 
671             #pragma omp critical
672                 if(dist<=rangeValue)
673                     ++inside_whole;
674             }
675 
676         }
677 
678         // count how many points in N_epsi(P_i) for current cluster
679     #pragma omp parallel num_threads(8)
680         {
681         #pragma omp for nowait
682             for(int j=0; j<node_number; ++j)
683             {
684                 // don't want to handle duplicates and itself
685                 if(i==j)
686                     continue;
687                 float dist = distM(i,j);
688 
689             #pragma omp critical
690                 if(dist<=rangeValue)
691                     ++inside_cluster;
692             }
693 
694         }
695         assert(inside_cluster<=inside_whole);
696         if(inside_whole==0)
697             continue;
698         result+=float(inside_cluster)/float(inside_whole);
699         ++index;
700         assert(!std::isnan(result));
701     }
702     return result;
703 }
```

4.14.3.5 void ValidityMeasurement::getMST_Parent_Node (std::tuple< float, float, float > & *values*, const std::vector< int > & *clusterNode*, const MetricPreparation & *object*, const int & *normOption*, const MatrixXf & *array*, const bool & *isPBF*) [private]

Definition at line 318 of file ValidityMeasurement.cpp.

```
321 {
322     using namespace boost;
323     typedef adjacency_list < vecS, vecS, undirectedS, no_property, property < edge_weight_t, float > >
    Graph;
324     typedef graph_traits < Graph >::edge_descriptor Edge;
325     typedef graph_traits < Graph >::vertex_descriptor Vertex;
326     typedef std::pair<int, int> E;
327 
328     const int& num_nodes = clusterNode.size();
329 
330     if(num_nodes<=1)
331     {
332         values = std::make_tuple(0.0,0.0,0.0);
333         std::cout << "Find 1-candidate cluster!" << std::endl;
334         return;
335     }
336 
337     const int num_edges = num_nodes*(num_nodes-1)/2;
338     Eigen::MatrixXf distM;
339     // if distanceMatrix is not stored ahead of time
340     if(isPBF)
341         distM = Eigen::MatrixXf(num_nodes, num_nodes);
342     // assign [source, destination] index pair and weight lists
343     E *edge_array = new E[num_edges];
344     float *weights = new float[num_edges], dist;
345     int temp = 0;
346     for(int i=0; i<num_nodes-1; ++i)
347     {
348         for(int j=i+1; j<num_nodes; ++j)
349         {
350             // assign index pair
351             edge_array[temp] = std::make_pair(i, j);
352             // assign weight list
353             if(distanceMatrix)
354                 dist = distanceMatrix[clusterNode[i]][clusterNode[j]];
355             else
```

```
356                 dist = getDisimilarity(array, clusterNode[i], clusterNode[j], normOption,
    object);
357
358             if(isPBF)
359             {
360                 distM(i,j) = dist;
361                 distM(j,i) = dist;
362             }
363
364             weights[temp] = dist;
365             ++temp;
366         }
367     }
368
369 #if defined(BOOST_MSVC) && BOOST_MSVC <= 1300
370     Graph g(num_nodes);
371     property_map<Graph, edge_weight_t>::type weightmap = get(edge_weight, g);
372     for (std::size_t j = 0; j < num_edges; ++j) {
373         Edge e; bool inserted;
374         tie(e, inserted) = add_edge(edge_array[j].first, edge_array[j].second, g);
375         weightmap[e] = weights[j];
376     }
377 #else
378     Graph g(edge_array, edge_array + num_edges, weights, num_nodes);
379 #endif
380     property_map < Graph, edge_weight_t >::type weight = get(edge_weight, g);
381     std::vector < Edge > spanning_tree;
382     kruskal_minimum_spanning_tree(g, std::back_inserter(spanning_tree));
383
384     if(edge_array!=NULL)
385     {
386         delete[] edge_array;
387         edge_array = NULL;
388     }
389
390     if(weights!=NULL)
391     {
392         delete[] weights;
393         weights = NULL;
394     }
395
396     // compute the standard deviation for the distance in MST
397     double summation = 0.0, sq_summation = 0.0, average_mst_d, max_d_mst = -1.0;
398     const int& MST_EDGE_NUM = num_nodes-1;
399
400     for (std::vector < Edge >::iterator ei = spanning_tree.begin(); ei != spanning_tree.end(); ++ei)
401     {
402             dist = weight[*ei];
403             max_d_mst = std::max(double(dist), max_d_mst);
404             summation+=dist;
405             sq_summation+=dist*dist;
406     }
407
408     float variance;
409
410     if(MST_EDGE_NUM<=1)
411     {
412         variance = 0;
413         average_mst_d = summation;
414     }
415     else
416     {
417         average_mst_d = summation/float(MST_EDGE_NUM);
418         variance = sq_summation/float(MST_EDGE_NUM-1)-average_mst_d*summation/float(MST_EDGE_NUM-1);
419
420         if(variance<1.0E-10)
421         {
422             variance = 1.0E-10;
423         }
424         variance = sqrt(variance);
425     }
426
427     // compute the inner Sc value for this cluster
428     int min_index, max_index;
429     float min_Sc = get_Sc_by_range(isPBF, distM, clusterNode, max_d_mst, object,
    normOption, array, min_index);
430     float max_Sc = get_Sc_by_range(isPBF, distM, clusterNode, average_mst_d, object,
    normOption, array, max_index);
431
432     min_Sc/=float(min_index);
433     max_Sc/=float(max_index);
434
435     // store the standard deviation, min Sc and max Sc in the tuple
436     values = std::make_tuple(variance, min_Sc, max_Sc);
437 }
```

4.14.3.6 void ValidityMeasurement::getMST_Parent_Node (std::tuple< float, float, float > & *values*, const std::vector< int > & *clusterNode*, const MatrixXf & *array*) [private]

Definition at line 447 of file ValidityMeasurement.cpp.

```
449 {
450     using namespace boost;
451     typedef adjacency_list < vecS, vecS, undirectedS, no_property, property < edge_weight_t, float > >
    Graph;
452     typedef graph_traits < Graph >::edge_descriptor Edge;
453     typedef graph_traits < Graph >::vertex_descriptor Vertex;
454     typedef std::pair<int, int> E;
455     // get number of points in one cluster
456     const int& num_nodes = clusterNode.size();
457     if(num_nodes<=1)
458     {
459         values = std::make_tuple(0.0,0.0,0.0);
460         return;
461     }
462     const int num_edges = num_nodes*(num_nodes-1)/2;
463 
464     Eigen::MatrixXf distM = Eigen::MatrixXf(num_nodes, num_nodes);
465     // assign [source, destination] index pair and weight lists
466     E *edge_array = new E[num_edges];
467     float *weights = new float[num_edges], dist;
468     int temp = 0;
469     for(int i=0; i<num_nodes-1; ++i)
470     {
471         for(int j=i+1; j<num_nodes; ++j)
472         {
473             // assign index pair
474             edge_array[temp] = std::make_pair(i, j);
475 
476             dist = (array.row(clusterNode[i])-array.row(clusterNode[j])).norm();
477             distM(i,j) = dist;
478             distM(j,i) = dist;
479 
480             weights[temp] = dist;
481             ++temp;
482         }
483     }
484 
485 
486 #if defined(BOOST_MSVC) && BOOST_MSVC <= 1300
487     Graph g(num_nodes);
488     property_map<Graph, edge_weight_t>::type weightmap = get(edge_weight, g);
489     for (std::size_t j = 0; j < num_edges; ++j) {
490         Edge e; bool inserted;
491         tie(e, inserted) = add_edge(edge_array[j].first, edge_array[j].second, g);
492         weightmap[e] = weights[j];
493     }
494 #else
495     Graph g(edge_array, edge_array + num_edges, weights, num_nodes);
496 #endif
497     property_map < Graph, edge_weight_t >::type weight = get(edge_weight, g);
498     std::vector < Edge > spanning_tree;
499 
500     kruskal_minimum_spanning_tree(g, std::back_inserter(spanning_tree));
501 
502 
503     if(edge_array!=NULL)
504     {
505         delete[] edge_array;
506         edge_array = NULL;
507     }
508 
509     if(weights!=NULL)
510     {
511         delete[] weights;
512         weights = NULL;
513     }
514 
515     // compute the standard deviation for the distance in MST
516     double summation = 0.0, sq_summation = 0.0, average_mst_d, max_d_mst = -1.0;
517     const int& MST_EDGE_NUM = num_nodes-1;
518 
519     for (std::vector < Edge >::iterator ei = spanning_tree.begin(); ei != spanning_tree.end(); ++ei)
520     {
521         dist = weight[*ei];
522         summation+=dist;
523         sq_summation+=dist*dist;
524         max_d_mst=std::max(max_d_mst, double(dist));
525     }
```

```
526
527     float variance;
528     if(MST_EDGE_NUM<=1)
529     {
530         variance = 0;
531         average_mst_d = summation;
532     }
533     else
534     {
535         average_mst_d=summation/float(MST_EDGE_NUM);
536         variance = sq_summation/float(MST_EDGE_NUM-1)-average_mst_d*summation/float(MST_EDGE_NUM-1);
537
538         if(variance<1.0E-10)
539         {
540             variance = 1.0E-10;
541         }
542         variance = sqrt(variance);
543     }
544
545     // compute the inner Sc value for this cluster
546     int min_index, max_index;
547     float min_Sc = get_Sc_by_range(distM, clusterNode, max_d_mst, array, min_index);
548     float max_Sc = get_Sc_by_range(distM, clusterNode, average_mst_d, array, max_index
    );
549
550     min_Sc/=float(min_index);
551     max_Sc/=float(max_index);
552
553     // store the standard deviation, min Sc and max Sc in the tuple
554     values = std::make_tuple(variance, min_Sc, max_Sc);
555 }
```

4.14.4 Member Data Documentation

4.14.4.1 float ValidityMeasurement::f_c

Definition at line 29 of file ValidityMeasurement.h.

4.14.4.2 float ValidityMeasurement::max_Sc [private]

Definition at line 70 of file ValidityMeasurement.h.

4.14.4.3 float ValidityMeasurement::min_Sc [private]

Definition at line 70 of file ValidityMeasurement.h.

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

- ValidityMeasurement.h
- ValidityMeasurement.cpp

Chapter 5

File Documentation

5.1 CMakeLists.txt File Reference

Functions

- cmake_minimum_required (VERSION 2.6) set(component_SOURCES Distance.h Distance.cpp Initialization.↩ h Initialization.cpp IOHandler.h IOHandler.cpp Metric.h Metric.cpp PreComputing.h PreComputing.cpp Silhouette.h Silhouette.cpp ValidityMeasurement.h ValidityMeasurement.cpp DetermClusterNum.h Determ↩ ClusterNum.cpp) include(CheckCXXCompilerFlag) if(COMPILER_SUPPORTS_CXX11) set(CMAKE_CX↩ X_FLAGS"$

5.1.1 Function Documentation

5.1.1.1 cmake_minimum_required (VERSION 2. *6*)

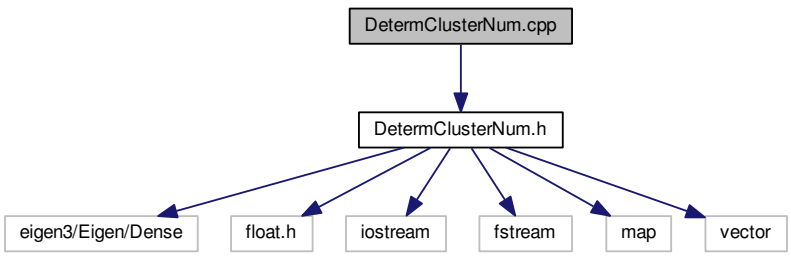
Definition at line 2 of file CMakeLists.txt.

```
27                          {CMAKE_CXX_FLAGS} -std=c++11")
```

5.2 DetermClusterNum.cpp File Reference

```
#include "DetermClusterNum.h"
```

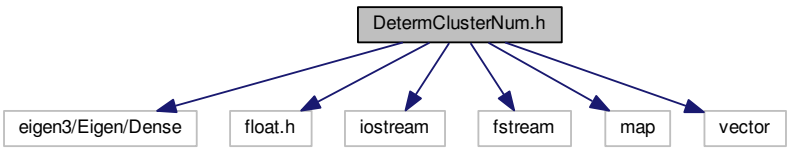
Include dependency graph for DetermClusterNum.cpp:



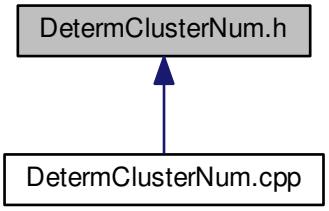
5.3 DetermClusterNum.h File Reference

```
#include <eigen3/Eigen/Dense>
#include <float.h>
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
```

Include dependency graph for DetermClusterNum.h:



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



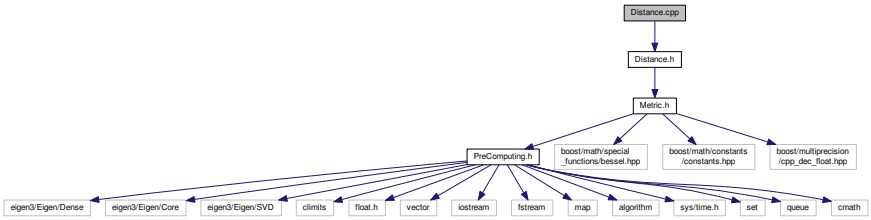
Classes

- class DetermClusterNum

5.4 Distance.cpp File Reference

```
#include "Distance.h"
```

Include dependency graph for Distance.cpp:



Functions

- const float getBMetric_3 (const VectorXf &row, const int &size, const int &i, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_3 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_3 (const int &first, const int &second, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_6 (const VectorXf &row, const int &size, const int &i, const std::vector< MultiVariate > &normalMultivariate)
- const float getBMetric_6 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_6 (const int &first, const int &second, const std::vector< MultiVariate > &normal↩ Multivariate)
- const float getBMetric_7 (const VectorXf &row, const int &size, const int &i, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_7 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_7 (const int &first, const int &second, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_9 (const VectorXf &row, const int &size, const int &i, const std::vector< MultiVariate > &normalMultivariate)
- const float getBMetric_9 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_9 (const int &first, const int &second, const std::vector< MultiVariate > &normal↩ Multivariate)
- const float getBMetric (const std::vector< float > &firstNorm3, const std::vector< float > &secondNorm3)
- const float getBMetric (const MultiVariate &centerNormal, const MultiVariate &neighNormal)
- const float getMetric_10 (const VectorXf &centroid, const int &size, const int &index, const std::vector< VectorXf > &unitLength)
- const float getMetric_10 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getMetric_10 (const int &first, const int &second, const std::vector< VectorXf > &unitLength)
- const float getNorm (const Eigen::VectorXf &centroid, const Eigen::VectorXf &r2, const int &index, const int &normOption, const std::vector< std::vector< float > > &pairwise, const std::vector< std::vector< float > > &objectNorm)
- const float getNorm (const VectorXf &centroid, const VectorXf &r2, const int &firstIndex, const int &second↩ Index, const int &normOption, const std::vector< std::vector< float > > &pairwise, const std::vector< std↩ ::vector< float > > &objectNorm)
- const float getNorm (const Eigen::VectorXf &r1, const Eigen::VectorXf &r2, const int &normOption)
- const float getDisimilarity (const MatrixXf &data, const int &first, const int &second, const int &normOption, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &others, const MatrixXf &data, const int &index, const int &norm↩ Option, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &first, const VectorXf &second, const int &firstIndex, const int &secondIndex, const int &normOption, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &first, const VectorXf &second, const int &normOption, const MetricPreparation &object)
- const float getMetric_MOP (const VectorXf &first, const VectorXf &second)
- const float getMetric_Hausdorff (const VectorXf &first, const VectorXf &second)
- void getDistanceMatrix (const MatrixXf &data, const int &normOption, const MetricPreparation &object)
- void deleteDistanceMatrix (const int &Row)
- const float getRotation (const std::vector< vector< float > > &streamline, std::vector< float > &rotation)
- const float getSignatureMetric (const Eigen::VectorXf &firstArray, const Eigen::VectorXf &secondArray, const std::vector< float > &firstHist, const std::vector< float > &secondHist)
- const float getSignatureMetric (const Eigen::VectorXf &centroid, const Eigen::VectorXf &first, const std↩ ::vector< float > &firstHist)
- const float getSignatureMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getProcrustesMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getProcrustesMetricSegment (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- void generateGroups (const std::vector< std::vector< int > > &storage)

- const float getEntropyMetric (const std::vector< float > &firstEntropy, const std::vector< float > &secondEntropy)
- const float getEntropyMetric (const std::vector< float > &firstEntropy, const Eigen::VectorXf &array)
- const float getEntropyMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getPathline_MCP (const Eigen::VectorXf &first, const Eigen::VectorXf &second)

Variables

- const int & PROCRUSTES_SIZE = 7
- float ** distanceMatrix = NULL

5.4.1 Function Documentation

5.4.1.1 void deleteDistanceMatrix (const int & *Row*)

Definition at line 1382 of file Distance.cpp.

```
1383 {
1384     if(distanceMatrix)
1385     {
1386     #pragma omp parallel for schedule(static) num_threads(8)
1387         for (int i = 0; i < Row; ++i)
1388         {
1389             if(distanceMatrix[i])
1390             {
1391                 delete[] distanceMatrix[i];
1392                 distanceMatrix[i] = NULL;
1393             }
1394         }
1395         delete[] distanceMatrix;
1396         distanceMatrix = NULL;
1397     }
1398 }
```

5.4.1.2 void generateGroups (const std::vector< std::vector< int > > & *storage*)

Definition at line 1808 of file Distance.cpp.

```
1809 {
1810     if(storage.empty())
1811         return;
1812     std::ofstream readme("../dataset/Storage",ios::out|ios::app);
1813     if(!readme)
1814     {
1815         std::cout << "Error creating Storage!" << std::endl;
1816         exit(1);
1817     }
1818
1819     readme << std::endl;
1820     const int& groupSize = storage.size();
1821     std::vector<int> element;
1822     for(int i=0;i<groupSize;++i)
1823     {
1824         element = storage[i];
1825         if(element.empty())
1826             continue;
1827         for(int j=0;j<element.size();++j)
1828             readme << element[j] << " ";
1829         readme << std::endl;
1830     }
1831     std::cout << std::endl;
1832     readme.close();
1833 }
```

5.4.1.3 const float getBMetric (const std::vector< float > & *firstNorm3,* const std::vector< float > & *secondNorm3*)

Definition at line 284 of file Distance.cpp.

```
287 {
288     // calculate mean and standard deviation of the two arrays
289     float u_a, u_b, sig_a, sig_b, sig_a_inverse, sig_b_inverse,
290           summation, sum_inverse, tempDist;
291     u_a = firstNorm3[0], u_b = secondNorm3[0];
292     sig_a = firstNorm3[1], sig_b = secondNorm3[1];
293     sig_a *= sig_a, sig_b *= sig_b;
294     if(sig_a<=1.0e-8)
295     {
296         sig_a = 1.0e-8;
297         sig_a_inverse = 1.0e8;
298     }
299     else
300         sig_a_inverse = 1.0/sig_a;
301     if(sig_b<=1.0e-8)
302     {
303         sig_b = 1.0e-8;
304         sig_b_inverse = 1.0/sig_b;
305     }
306     summation = sig_a+sig_b;
307     sum_inverse = 1.0/summation;
308     tempDist = 0.25*log(0.25*(sig_a*sig_b_inverse
309                +sig_b*sig_a_inverse+2))
310                + 0.25*(u_a-u_b)*(u_a-u_b)*sum_inverse;
311     return tempDist;
312 }
```

5.4.1.4 const float getBMetric (const MultiVariate & *centerNormal,* const MultiVariate & *neighNormal*)

Definition at line 321 of file Distance.cpp.

```
324 {
325     Matrix3f firstCov, secondCov, meanCov, meanCovInverse;
326     float sqrtInverse, meanCovDet;
327     firstCov = centerNormal.covariance;
328     secondCov = neighNormal.covariance;
329     meanCov = 0.5*(firstCov+secondCov);
330     if(meanCov.determinant()>1.0e-8)
331     {
332         meanCovInverse = static_cast<Matrix3f>(meanCov.inverse());
333         meanCovDet = meanCov.determinant();
334     }
335     else
336     {
337         meanCovInverse = pseudoInverse(meanCov);
338         meanCovDet = 1.0e8;
339     }
340     float detMulti = sqrt(firstCov.determinant()*secondCov.determinant());
341     sqrtInverse = detMulti>1.0e-8?float(1.0)/detMulti:1.0e8;
342     Vector3f meanDiff = centerNormal.meanVec-neighNormal.meanVec;
343     float tempDist = 0.125*meanDiff.transpose()*meanCovInverse*meanDiff
344                +0.2*log(meanCovDet*sqrtInverse);
345     return tempDist;
346 }
```

5.4.1.5 const float getBMetric_3 (const VectorXf & *row,* const int & *size,* const int & *i,* const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 51 of file Distance.cpp.

```
56 {
57     std::vector<float> firstNorm3, secondNorm3;
58     getSequence(row, size, firstNorm3);
59     secondNorm3 = rotationSequence[i];
60     return getBMetric(firstNorm3, secondNorm3);
61 }
```

5.4.1.6 const float getBMetric_3 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 72 of file Distance.cpp.

```
76 {
77     std::vector<float> firstNorm3, secondNorm3;
78     getSequence(firstRow, size, firstNorm3);
79     getSequence(secondRow, size, secondNorm3);
80     return getBMetric(firstNorm3, secondNorm3);
81 }
```

5.4.1.7 const float getBMetric_3 (const int & *first,* const int & *second,* const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 92 of file Distance.cpp.

```
96 {
97     return getBMetric(rotationSequence[first], rotationSequence[second]);
98 }
```

5.4.1.8 const float getBMetric_6 (const VectorXf & *row,* const int & *size,* const int & *i,* const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 110 of file Distance.cpp.

```
115 {
116     MultiVariate centerNormal, neighNormal;
117     getNormalMultivariate(row, size, centerNormal);
118     neighNormal = normalMultivariate[i];
119     return getBMetric(centerNormal, neighNormal);
120 }
```

5.4.1.9 const float getBMetric_6 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 131 of file Distance.cpp.

```
135 {
136     MultiVariate centerNormal, neighNormal;
137     getNormalMultivariate(firstRow, size, centerNormal);
138     getNormalMultivariate(secondRow, size, neighNormal);
139     return getBMetric(centerNormal, neighNormal);
140 }
```

5.4.1.10 const float getBMetric_6 (const int & *first,* const int & *second,* const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 151 of file Distance.cpp.

```
155 {
156     return getBMetric(normalMultivariate[first], normalMultivariate[second]);
157 }
```

5.4.1.11 const float getBMetric_7 (const VectorXf & *row*, const int & *size*, const int & *i*, const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 169 of file Distance.cpp.

```
174 {
175     std::vector<float> firstNorm3, secondNorm3;
176     getEachFixedSequence(row, size, firstNorm3);
177     secondNorm3 = rotationSequence[i];
178     return getBMetric(firstNorm3, secondNorm3);
179 }
```

5.4.1.12 const float getBMetric_7 (const VectorXf & *firstRow*, const int & *size*, const VectorXf & *secondRow*)

Definition at line 190 of file Distance.cpp.

```
194 {
195     std::vector<float> firstNorm3, secondNorm3;
196     getEachFixedSequence(firstRow, size, firstNorm3);
197     getEachFixedSequence(secondRow, size, secondNorm3);
198     return getBMetric(firstNorm3, secondNorm3);
199 }
```

5.4.1.13 const float getBMetric_7 (const int & *first*, const int & *second*, const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 210 of file Distance.cpp.

```
214 {
215     return getBMetric(rotationSequence[first], rotationSequence[second]);
216 }
```

5.4.1.14 const float getBMetric_9 (const VectorXf & *row*, const int & *size*, const int & *i*, const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 228 of file Distance.cpp.

```
233 {
234     MultiVariate centerNormal, neighNormal;
235     getUnnormalizedMultivariate(row, size, centerNormal);
236     neighNormal = normalMultivariate[i];
237     return getBMetric(centerNormal, neighNormal);
238 }
```

5.4.1.15 const float getBMetric_9 (const VectorXf & *firstRow*, const int & *size*, const VectorXf & *secondRow*)

Definition at line 249 of file Distance.cpp.

```
253 {
254     MultiVariate centerNormal, neighNormal;
255     getUnnormalizedMultivariate(firstRow, size, centerNormal);
256     getUnnormalizedMultivariate(secondRow, size, neighNormal);
257     return getBMetric(centerNormal, neighNormal);
258 }
```

5.4.1.16 const float getBMetric_9 (const int & *first,* const int & *second,* const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 269 of file Distance.cpp.

```
273 {
274     return getBMetric(normalMultivariate[first], normalMultivariate[second]);
275 }
```

5.4.1.17 const float getDisimilarity (const MatrixXf & *data,* const int & *first,* const int & *second,* const int & *normOption,* const MetricPreparation & *object*)

Definition at line 964 of file Distance.cpp.

```
969 {
970     return getDisimilarity(data.row(first), data.row(second),
971                            first, second, normOption, object);
972 }
```

5.4.1.18 const float getDisimilarity (const VectorXf & *others,* const MatrixXf & *data,* const int & *index,* const int & *normOption,* const MetricPreparation & *object*)

Definition at line 985 of file Distance.cpp.

```
990 {
991     float length;
992     switch(normOption)
993     {
994     case 0: // Euclidean distance, d_E
995     case 1: // Fraction norm, d_F
996     case 2: // geometric similarity measure, d_G
997     case 5:
998     case 8:
999     case 11:
1000         length = getNorm(others, data.row(index),index,normOption,
1001                    object.pairwise, object.pairwiseNorm);
1002         break;
1003
1004     case 3:
1005         length = getBMetric_3(others, others.size()/3-2, index,
1006                               object.rotationSequence);
1007         break;
1008
1009     case 4:
1010         length = abs(object.rotation[index]-
1011                  getRotation(others, others.size()/3-2));
1012         break;
1013
1014     case 6:
1015         length = getBMetric_6(others, others.size()/3-1, index,
1016                               object.normalMultivariate);
1017         break;
1018
1019     case 7:
1020         length = getBMetric_7(others, others.size()/3-1, index,
1021                               object.rotationSequence);
1022         break;
1023
1024     case 9:
1025         length = getBMetric_9(others, others.size()/3-1, index,
1026                               object.normalMultivariate);
1027         break;
1028
1029     case 10:
1030         length = getMetric_10(others, others.size()/3, index,
1031                               object.unitLength);
1032         break;
1033
```

```
1034     case 12:    // the MCP distance, i.e., d_M
1035         length = getMetric_MOP(others, data.row(index));
1036         break;
1037
1038     case 13:    // the Hausdorff distance, i.e., d_H
1039         length = getMetric_Hausdorff(others, data.row(index));
1040         break;
1041
1042     /* signature-based similarity metric with chi-squared test combined with mean-closest */
1043     case 14:    // the signature-based similarity, i.e., d_S
1044         length = getSignatureMetric(others,data.row(index),object.pairwise[index]);
1045         break;
1046
1047     /* adapted Procrustes distance */
1048     case 15:    // the Procrustes distance, i.e., d_P
1049         //length = getProcrustesMetric(others, data.row(index));
1050         length = std::min(getProcrustesMetricSegment(others, data.row(index)),
1051                 getProcrustesMetricSegment(data.row(index), others));
1052         break;
1053
1054     case 16:
1055         length = getEntropyMetric(object.pairwise[index], others);
1056         break;
1057
1058     case 17:    // the time-based MCP, i.e., d_T
1059         length = getPathline_MCP(others, data.row(index));
1060         break;
1061
1062     default:
1063         exit(1);
1064         break;
1065     }
1066
1067     return length;
1068 }
```

5.4.1.19 const float getDisimilarity (const VectorXf & *first*, const VectorXf & *second*, const int & *firstIndex*, const int & *secondIndex*, const int & *normOption*, const MetricPreparation & *object*)

Definition at line 1082 of file Distance.cpp.

```
1088 {
1089     float length;
1090     switch(normOption)
1091     {
1092     case 0: // Euclidean distance, d_E
1093     case 1: // Fraction norm, d_F
1094     case 2: // Geometric similarity, d_G
1095     case 5:
1096     case 8:
1097     case 11:
1098         length = getNorm(first, second,firstIndex,secondIndex, normOption,
1099                     object.pairwise, object.pairwiseNorm);
1100         break;
1101
1102     case 3:
1103         length = getBMetric_3(firstIndex, secondIndex,
1104                     object.rotationSequence);
1105         break;
1106
1107     case 4:
1108         length = abs(object.rotation[firstIndex]-
1109                     object.rotation[secondIndex]);
1110         break;
1111
1112     case 6:
1113         length = getBMetric_6(firstIndex, secondIndex,
1114                     object.normalMultivariate);
1115         break;
1116
1117     case 7:
1118         length = getBMetric_7(firstIndex, secondIndex,
1119                     object.rotationSequence);
1120         break;
1121
1122     case 9:
1123         length = getBMetric_9(firstIndex, secondIndex,
1124                     object.normalMultivariate);
1125         break;
```

```
1126
1127     case 10:
1128         length = getMetric_10(firstIndex, secondIndex,
1129                     object.unitLength);
1130         break;
1131
1132     case 12:    // the MCP distance, d_M
1133         length = getMetric_MOP(first, second);
1134         break;
1135
1136     case 13:    // the Hausdorff distance, d_H
1137         length = getMetric_Hausdorff(first, second);
1138         break;
1139
1140     case 14:    //  the signature-based distance, d_S
1141         length = getSignatureMetric(first,second,object.pairwise[firstIndex],object.
      pairwise[secondIndex]);
1142         break;
1143
1144     case 15:    // the Procrutes distance, d_P
1145         //length = getProcrustesMetric(first, second);
1146         length = std::min(getProcrustesMetricSegment(first,second),
      getProcrustesMetricSegment(second,first));
1147         break;
1148
1149     case 16:
1150         length = getEntropyMetric(object.pairwise[firstIndex], object.pairwise[secondIndex]
      );
1151         break;
1152
1153     case 17:    // the time-based MCP, d_T
1154         length = getPathline_MCP(first, second);
1155         break;
1156
1157     default:
1158         exit(1);
1159         break;
1160     }
1161     return length;
1162 }
```

5.4.1.20 const float getDisimilarity (const VectorXf & *first,* const VectorXf & *second,* const int & *normOption,* const MetricPreparation & *object*)

Definition at line 1174 of file Distance.cpp.

```
1178 {
1179     float length;
1180     switch(normOption)
1181     {
1182     case 0: // Euclidean distance, d_E
1183     case 1: // Fraction norm, d_F
1184     case 2: // Geometric similarity, d_G
1185     case 5:
1186     case 8:
1187     case 11:
1188         length = getNorm(first, second, normOption);
1189         break;
1190
1191     case 3:
1192         length = getBMetric_3(first, first.size()/3-2, second);
1193         break;
1194
1195     case 4:
1196         length = abs(getRotation(first, first.size()/3-2)-getRotation(second, second.
      size()/3-2));
1197         break;
1198
1199     case 6:
1200         length = getBMetric_6(first, first.size()/3-1,second);
1201         break;
1202
1203     case 7:
1204         length = getBMetric_7(first, first.size()/3-1, second);
1205         break;
1206
1207     case 9:
1208         length = getBMetric_9(first, first.size()/3-1, second);
1209         break;
```

```
1210
1211     case 10:
1212         length = getBMetric_9(first, first.size()/3, second);
1213         break;
1214
1215     case 12:    // the MCP distance, d_M
1216         length = getMetric_MOP(first, second);
1217         break;
1218
1219     case 13:    // the Hausdorff distance, d_H
1220         length = getMetric_Hausdorff(first, second);
1221         break;
1222
1223     /* signature-based similarity metric with chi-squared test combined with mean-closest */
1224     case 14:    // the signature-based similarity, d_S
1225         length = getSignatureMetric(first, second);
1226         break;
1227
1228     /* adapted Procrustes distance */
1229     case 15:    // the Procrustes distance, d_P
1230         //length = getProcrustesMetric(first, second);
1231         length = getProcrustesMetricSegment(first,second);
1232         break;
1233
1234     case 16:
1235         length = getEntropyMetric(first, second);
1236         break;
1237
1238     case 17:    // the time-based MCP, d_T
1239         length = getPathline_MCP(first, second);
1240         break;
1241
1242     default:
1243         exit(1);
1244         break;
1245     }
1246
1247     return length;
1248 }
```

5.4.1.21 void getDistanceMatrix (const MatrixXf & *data,* const int & *normOption,* const MetricPreparation & *object*)

Definition at line 1348 of file Distance.cpp.

```
1351 {
1352     const int& Row = data.rows();
1353     distanceMatrix = new float*[Row];
1354
1355     // assign the distance matrix
1356 #pragma omp parallel for schedule(static) num_threads(8)
1357     for (int i = 0; i < Row; ++i)
1358     {
1359         distanceMatrix[i] = new float[Row];
1360         for (int j = 0; j < Row; ++j)
1361         {
1362             /* don't wish to waste computation on diagonal element */
1363             if(i==j)
1364                 distanceMatrix[i][j] = 0.0;
1365             else
1366                 distanceMatrix[i][j] = getDisimilarity(data, i, j, normOption,
       object);
1367         }
1368     }
1369     // help check whether they already been assigned and whether they are symmetric or not
1370     std::cout << "Distance between 215 and 132 is " << distanceMatrix[215][132] << std::endl;
1371     std::cout << "Distance between 132 and 215 is " << distanceMatrix[132][215] << std::endl;
1372
1373     std::cout << "Finished computing distance matrix!" << std::endl;
1374 }
```

5.4.1.22 const float getEntropyMetric (const std::vector< float > & *firstEntropy,* const std::vector< float > & *secondEntropy*)

Definition at line 1845 of file Distance.cpp.

```
1847 {
1848     assert(firstEntropy.size()==2);
1849     assert(secondEntropy.size()==2);
1850 
1851     float first = firstEntropy[0]-secondEntropy[0];
1852     float second = firstEntropy[1]-secondEntropy[1];
1853 
1854     return sqrt(first*first+second*second);
1855 }
```

5.4.1.23 const float getEntropyMetric (const std::vector< float > & *firstEntropy,* const Eigen::VectorXf & *array*)

Definition at line 1868 of file Distance.cpp.

```
1870 {
1871     assert(firstEntropy.size()==2);
1872 
1873     std::vector<float> secondEntropy;
1874 
1875     getLinearAngularEntropy(array, BUNDLE_SIZE, secondEntropy);
1876 
1877     return getEntropyMetric(firstEntropy, secondEntropy);
1878 }
```

5.4.1.24 const float getEntropyMetric (const Eigen::VectorXf & *first,* const Eigen::VectorXf & *second*)

Definition at line 1890 of file Distance.cpp.

```
1892 {
1893 
1894     std::vector<float> firstEntropy, secondEntropy;
1895 
1896     getLinearAngularEntropy(first, BUNDLE_SIZE, firstEntropy);
1897     getLinearAngularEntropy(second, BUNDLE_SIZE, secondEntropy);
1898 
1899     return getEntropyMetric(firstEntropy, secondEntropy);
1900 }
```

5.4.1.25 const float getMetric_10 (const VectorXf & *centroid,* const int & *size,* const int & *index,* const std::vector< VectorXf > & *unitLength*)

Definition at line 358 of file Distance.cpp.

```
362 {
363     const VectorXf& x = unitLength[index];
364     VectorXf y(size*3);
365     getUnitDirection_byEach(centroid,size,y);
366 
367     float length = x.dot(y)/x.size();
368     length = min(1.0,(double)length);
369     length = max(-1.0,(double)length);
370     length = acos(length);
371     return length;
372 }
```

5.4.1.26 const float getMetric_10 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 383 of file Distance.cpp.

```
386 {
387     VectorXf x(size*3), y(size*3);
388     getUnitDirection_byEach(firstRow,size,x);
389     getUnitDirection_byEach(secondRow,size,y);
390
391     float length = x.dot(y)/x.size();
392     length = min(1.0,(double)length);
393     length = max(-1.0,(double)length);
394     length = acos(length);
395     return length;
396 }
```

5.4.1.27 const float getMetric_10 (const int & *first,* const int & *second,* const std::vector< VectorXf > & *unitLength*)

Definition at line 407 of file Distance.cpp.

```
410 {
411     const VectorXf& x = unitLength[first];
412     const VectorXf& y = unitLength[second];
413     float length = x.dot(y)/x.size();
414     length = min(1.0,(double)length);
415     length = max(-1.0,(double)length);
416     length = acos(length);
417     return length;
418 }
```

5.4.1.28 const float getMetric_Hausdorff (const VectorXf & *first,* const VectorXf & *second*)

Definition at line 1305 of file Distance.cpp.

```
1306 {
1307     // the max of first to second
1308     const int& vNum = first.size()/3;
1309     float result, f_to_s=-1.0, s_to_f=-1.0;
1310     for(int i=0;i<vNum;++i)
1311     {
1312         float minDist = FLT_MAX;
1313         Vector3f m_i = Vector3f(first(3*i),first(3*i+1),first(3*i+2));
1314         for(int j=0;j<vNum;++j)
1315         {
1316             Vector3f n_j = Vector3f(second(3*j),second(3*j+1),second(3*j+2));
1317             minDist = std::min((m_i-n_j).norm(),minDist);
1318         }
1319         s_to_f=std::max(s_to_f, minDist);
1320     }
1321
1322     // the max of second to first
1323     for(int i=0;i<vNum;++i)
1324     {
1325         float minDist = FLT_MAX;
1326         Vector3f m_i = Vector3f(second(3*i),second(3*i+1),second(3*i+2));
1327         for(int j=0;j<vNum;++j)
1328         {
1329             Vector3f n_j = Vector3f(first(3*j),first(3*j+1),first(3*j+2));
1330             minDist = std::min((m_i-n_j).norm(),minDist);
1331         }
1332         f_to_s=std::max(f_to_s, minDist);
1333     }
1334
1335     // max of the max
1336     result = std::max(f_to_s, s_to_f);
1337     return result;
1338 }
```

5.4.1.29 const float getMetric_MOP (const VectorXf & *first*, const VectorXf & *second*)

Definition at line 1258 of file Distance.cpp.

```
1259 {
1260     // The MCP of first to second
1261     const int& vNum = first.size()/3;
1262     float result, f_to_s, s_to_f;
1263     float summation = 0;
1264     for(int i=0;i<vNum;++i)
1265     {
1266         float minDist = FLT_MAX;
1267         Vector3f m_i = Vector3f(first(3*i),first(3*i+1),first(3*i+2));
1268         for(int j=0;j<vNum;++j)
1269         {
1270             Vector3f n_j = Vector3f(second(3*j),second(3*j+1),second(3*j+2));
1271             minDist = std::min((m_i-n_j).norm(),minDist);
1272         }
1273         summation+=minDist;
1274     }
1275     s_to_f = summation/vNum;
1276
1277     // The MCP of second to first
1278     summation = 0;
1279     for(int i=0;i<vNum;++i)
1280     {
1281         float minDist = FLT_MAX;
1282         Vector3f m_i = Vector3f(second(3*i),second(3*i+1),second(3*i+2));
1283         for(int j=0;j<vNum;++j)
1284         {
1285             Vector3f n_j = Vector3f(first(3*j),first(3*j+1),first(3*j+2));
1286             minDist = std::min((m_i-n_j).norm(),minDist);
1287         }
1288         summation+=minDist;
1289     }
1290     f_to_s = summation/vNum;
1291
1292     // get the average of that
1293     result = (f_to_s+s_to_f)/2.0;
1294     return result;
1295 }
```

5.4.1.30 const float getNorm (const Eigen::VectorXf & *centroid*, const Eigen::VectorXf & *r2*, const int & *index*, const int & *normOption*, const std::vector< std::vector< float > > & *pairwise*, const std::vector< std::vector< float > > & *objectNorm*)

Definition at line 432 of file Distance.cpp.

```
438 {
439     assert(centroid.size()==r2.size());
440     float length = 0.0;
441     switch(normOption)
442     {
443     case 0: // Euclidean distance
444     default:
445         length = (centroid-r2).norm();
446         break;
447
448     case 11:    // the norm 11
449         {
450             float dotPro = centroid.dot(r2);
451             float firstNorm = centroid.norm();
452             float secondNorm = r2.norm();
453             float firstInverse, secondInverse;
454             if(firstNorm<1.0e-8)
455                 firstInverse = 1.0e8;
456             else
457                 firstInverse = 1.0/firstNorm;
458             if(secondNorm<1.0e-8)
459                 secondInverse = 1.0e8;
460             else
461                 secondInverse = 1.0/secondNorm;
462             dotPro = dotPro*firstInverse*secondInverse;
463             dotPro = std::max(dotPro, float(-1.0));
464             dotPro = std::min(dotPro, float(1.0));
```

```
465             length = acos(dotPro)/M_PI;
466         }
467         break;
468 
469     case 1:  /* fraction norm by high-dimensional feature-space */
470         {
471             for (int i = 0; i < centroid.size(); ++i)
472             {
473                 length += pow(abs(centroid(i)-r2(i)),0.5);
474             }
475             length = pow(length,2.0);
476         }
477         break;
478 
479     case 2: /* mean value of dot product value, which means it's rotational invariant, or d_G */
480         {
481             const int& pointNum = centroid.size()/3-1;
482             float dotValue, leftNorm, rightNorm, result;
483 
484             std::vector<float> centroidWise;
485             std::vector<float> centroidWiseNorm;
486             getPairWise_byEach(centroid, pointNum, centroidWise, centroidWiseNorm);
487 
488             const std::vector<float>& i_Pairwise = pairwise[index];
489             const std::vector<float>& i_PairNorm = objectNorm[index];
490 
491             Vector3f left, right;
492             for (int i = 0; i < pointNum; ++i)
493             {
494                 leftNorm = centroidWiseNorm[i];
495                 rightNorm = i_PairNorm[i];
496                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
497                 {
498                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
499                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
500                     result = left.dot(right)/*/leftNorm/rightNorm*/;
501                     result = min(1.0,(double)result);
502                     result = max(-1.0,(double)result);
503                     length+=acos(result);
504                 }
505                 else
506                     length+=M_PI;
507             }
508             length /= pointNum;
509         }
510         break;
511 
512     case 5: /* rotational invariant line-wise acos angle with normal direction for
513                measuring whether counterclockwise or clockwise orientation */
514         {
515             const int& pointNum = centroid.size()/3-1;
516             float dotValue, leftNorm, rightNorm, normalDot, result;
517             Vector3f left, right, normal;
518 
519             std::vector<float> centroidWise;
520             std::vector<float> centroidNorm;
521             getPairWise_byEach(centroid, pointNum, centroidWise, centroidNorm);
522             const std::vector<float>& i_Pairwise = pairwise[index];
523             const std::vector<float>& i_PairNorm = objectNorm[index];
524 
525             left << /*centroid(3)-centroid(0),centroid(4)-centroid(1),centroid(5)-centroid(2)*/
526                     centroidWise[0], centroidWise[1], centroidWise[2];
527             right << /*r2(3)-r2(0),r2(4)-r2(1),r2(5)-r2(2)*/
528                     i_Pairwise[0], i_Pairwise[1], i_Pairwise[2];
529             const Vector3f& Normal = left.cross(right);
530 
531             for (int i = 0; i < pointNum; ++i)
532             {
533 
534                 leftNorm = centroidNorm[i];
535                 rightNorm = i_PairNorm[i];
536                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
537                 {
538                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
539                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
540                     result = left.dot(right)/*/leftNorm/rightNorm*/;
541                     result = min(1.0,(double)result);
542                     result = max(-1.0,(double)result);
543                     normal = left.cross(right);
544                     normalDot = Normal.dot(normal);
545                     if(normalDot<0)
546                         length+=-acos(result);
547                     else
548                         length+=acos(result);
549                 }
550                 else
551                     length+=M_PI;
```

```
552             }
553             length /= pointNum;
554             length = abs(length);
555         }
556         break;
557 
558     case 8: /* distance metric defined as mean * standard deviation */
559         {
560             const int& pointNum = centroid.size()/3-1;
561             float dotValue, leftNorm, rightNorm, stdevia = 0.0, angle, result;
562             Vector3f left, right;
563 
564             std::vector<float> centroidWise;
565             std::vector<float> centroidNorm;
566             getPairWise_byEach(centroid, pointNum, centroidWise, centroidNorm);
567 
568             const std::vector<float>& i_Pairwise = pairwise[index];
569             const std::vector<float>& i_PairNorm = objectNorm[index];
570 
571             for (int i = 0; i < pointNum; ++i)
572             {
573                 leftNorm = centroidNorm[i];
574                 rightNorm = i_PairNorm[i];
575 
576                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
577                 {
578                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
579                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
580                     result = left.dot(right)/*/leftNorm/rightNorm*/;
581                     result = min(1.0,(double)result);
582                     result = max(-1.0,(double)result);
583                     angle = acos(result);
584                     length+=angle;
585                     stdevia+=angle*angle;
586                 }
587                 else
588                 {
589                     angle=M_PI;
590                     length+=angle;
591                     stdevia+=angle*angle;
592                 }
593             }
594             length /= pointNum;
595             stdevia = stdevia/pointNum-length*length;
596             if(stdevia>0)
597                 stdevia = sqrt(stdevia/pointNum-length*length);
598             else
599                 stdevia = 1.0e-4;
600         }
601         break;
602     }
603 
604     return length;
605 }
```

5.4.1.31 const float getNorm (const VectorXf & *centroid,* const VectorXf & *r2,* const int & *firstIndex,* const int & *secondIndex,* const int & *normOption,* const std::vector< std::vector< float > > & *pairwise,* const std::vector< std::vector< float > > & *objectNorm*)

Definition at line 620 of file Distance.cpp.

```
627 {
628     assert(centroid.size()==r2.size());
629     float length = 0.0;
630     switch(normOption)
631     {
632     case 0:
633     default:
634         length = (centroid-r2).norm();
635         break;
636 
637     case 1:  /* fraction norm by high-dimensional feature-space, or d_F */
638         {
639             for (int i = 0; i < centroid.size(); ++i)
640             {
641                 length += pow(abs(centroid(i)-r2(i)),0.5);
642             }
643             length = pow(length,2.0);
644         }
```

```
645         break;
646
647     case 11:
648         {
649             float dotPro = centroid.dot(r2);
650             float firstNorm = centroid.norm();
651             float secondNorm = r2.norm();
652             float firstInverse, secondInverse;
653             if(firstNorm<1.0e-8)
654                 firstInverse = 1.0e8;
655             else
656                 firstInverse = 1.0/firstNorm;
657             if(secondNorm<1.0e-8)
658                 secondInverse = 1.0e8;
659             else
660                 secondInverse = 1.0/secondNorm;
661             dotPro = dotPro*firstInverse*secondInverse;
662             dotPro = std::max(dotPro, float(-1.0));
663             dotPro = std::min(dotPro, float(1.0));
664             length = acos(dotPro)/M_PI;
665         }
666         break;
667
668     case 2: /* mean value of dot product value, which means it's rotational invariant, or d_G */
669         {
670             const int& pointNum = centroid.size()/3-1;
671             float dotValue, leftNorm, rightNorm, result;
672
673             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
674             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
675
676             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
677             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
678
679             Vector3f left, right;
680             for (int i = 0; i < pointNum; ++i)
681             {
682                 leftNorm = i_PairNorm[i];
683                 rightNorm = j_PairNorm[i];
684                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
685                 {
686                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
687                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
688                     result = left.dot(right)/*/leftNorm/rightNorm*/;
689                     result = min(1.0,(double)result);
690                     result = max(-1.0,(double)result);
691                     length+=acos(result);
692                 }
693                 else
694                     length+=M_PI;
695             }
696             length /= pointNum;
697         }
698         break;
699
700     case 5: /* rotational invariant line-wise acos angle with normal direction for
701                measuring whether counterclockwise or clockwise orientation */
702         {
703             const int& pointNum = centroid.size()/3-1;
704             float dotValue, leftNorm, rightNorm, normalDot, result;
705             Vector3f left, right, normal;
706
707             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
708             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
709
710             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
711             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
712
713             left << /*centroid(3)-centroid(0),centroid(4)-centroid(1),centroid(5)-centroid(2)*/
714                     i_Pairwise[0], i_Pairwise[1], i_Pairwise[2];
715             right << /*r2(3)-r2(0),r2(4)-r2(1),r2(5)-r2(2)*/
716                     j_Pairwise[0], j_Pairwise[1], j_Pairwise[2];
717             const Vector3f& Normal = left.cross(right);
718
719             for (int i = 0; i < pointNum; ++i)
720             {
721                 leftNorm = i_PairNorm[i];
722                 rightNorm = j_PairNorm[i];
723
724                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
725                 {
726                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
727                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
728                     result = left.dot(right)/*/leftNorm/rightNorm*/;
729                     result = min(1.0,(double)result);
730                     result = max(-1.0,(double)result);
731                     normal = left.cross(right);
```

```
732                     normalDot = Normal.dot(normal);
733                     if(normalDot<0)
734                         length+=-acos(result);
735                     else
736                         length+=acos(result);
737                 }
738                 else
739                     length+=M_PI;
740             }
741             length /= pointNum;
742             length = abs(length);
743         }
744         break;
745 
746     case 8: /* distance metric defined as mean * standard deviation */
747         {
748             const int& pointNum = centroid.size()/3-1;
749             float dotValue, leftNorm, rightNorm, stdevia = 0.0, angle, result;
750             Vector3f left, right;
751 
752             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
753             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
754 
755             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
756             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
757 
758             for (int i = 0; i < pointNum; ++i)
759             {
760                 leftNorm = i_PairNorm[i];
761                 rightNorm = j_PairNorm[i];
762 
763                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
764                 {
765                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
766                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
767                     result = left.dot(right)/*/leftNorm/rightNorm*/;
768                     result = min(1.0,(double)result);
769                     result = max(-1.0,(double)result);
770                     angle = acos(result);
771                     length+=angle;
772                     stdevia+=angle*angle;
773                 }
774                 else
775                 {
776                     angle=M_PI;
777                     length+=angle;
778                     stdevia+=angle*angle;
779                 }
780             }
781             length /= pointNum;
782             stdevia = stdevia/pointNum-length*length;
783             if(stdevia>0)
784                 stdevia = sqrt(stdevia/pointNum-length*length);
785             else
786                 stdevia = 1.0e-4;
787         }
788         break;
789 
790     }
791 
792     return length;
793 }
```

5.4.1.32 const float getNorm (const Eigen::VectorXf & *r1*, const Eigen::VectorXf & *r2*, const int & *normOption*)

Definition at line 804 of file Distance.cpp.

```
807 {
808     assert(r1.size()==r2.size());
809     float length = 0.0;
810     switch(normOption)
811     {
812     case 0:
813     default:
814         length = (r1-r2).norm();
815         break;
816 
817     case 1:  /* fraction norm by high-dimensional feature-space */
818         {
819             for (int i = 0; i < r1.size(); ++i)
```

```
820             {
821                 length += pow(abs(r1(i)-r2(i)),0.5);
822             }
823             length = pow(length,2.0);
824         }
825         break;
826
827     case 2: /* mean value of dot product value, which means it's rotational invariant, or d_G */
828         {
829             const int& pointNum = r1.size()/3-1;
830             float dotValue, leftNorm, rightNorm, result;
831             Vector3f left, right;
832             for (int i = 0; i < pointNum; ++i)
833             {
834                 left << r1(3*i+3)-r1(3*i),r1(3*i+4)-r1(3*i+1),r1(3*i+5)-r1(3*i+2);
835                 right << r2(3*i+3)-r2(3*i),r2(3*i+4)-r2(3*i+1),r2(3*i+5)-r2(3*i+2);
836                 dotValue = left.dot(right);
837                 leftNorm = left.norm(), rightNorm = right.norm();
838                 if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
839                 {
840                     result = dotValue/*/leftNorm/rightNorm*/;
841                     result = min(1.0,(double)result);
842                     result = max(-1.0,(double)result);
843                     length+=acos(result);
844                 }
845                 else
846                     length+=M_PI;
847             }
848             length /= pointNum;
849         }
850         break;
851
852     case 5: /* rotational invariant line-wise acos angle with normal direction for
853                measuring whether counterclockwise or clockwise orientation */
854         {
855             const int& pointNum = r1.size()/3-1;
856             float dotValue, leftNorm, rightNorm, normalDot, result;
857             Vector3f left, right, normal;
858
859             left << r1(3)-r1(0),r1(4)-r1(1),r1(5)-r1(2);
860             right << r2(3)-r2(0),r2(4)-r2(1),r2(5)-r2(2);
861
862             const Vector3f& Normal = left.cross(right);
863
864             for (int i = 0; i < pointNum; ++i)
865             {
866                 left << r1(3*i+3)-r1(3*i),r1(3*i+4)-r1(3*i+1),r1(3*i+5)-r1(3*i+2);
867                 right << r2(3*i+3)-r2(3*i),r2(3*i+4)-r2(3*i+1),r2(3*i+5)-r2(3*i+2);
868                 normal = left.cross(right);
869                 dotValue = left.dot(right);
870                 normalDot = Normal.dot(normal);
871
872                 leftNorm = left.norm(), rightNorm = right.norm();
873                 if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
874                 {
875                     result = dotValue/*/leftNorm/rightNorm*/;
876                     result = min(1.0,(double)result);
877                     result = max(-1.0,(double)result);
878                     if(normalDot<0)
879                         length+=-acos(result);
880                     else
881                         length+=acos(result);
882                 }
883                 else
884                     length+=M_PI;
885             }
886             length /= pointNum;
887             length = abs(length);
888         }
889         break;
890
891     case 8: /* distance metric defined as mean * standard deviation */
892         {
893             const int& pointNum = r1.size()/3-1;
894             float dotValue, leftNorm, rightNorm, stdevia = 0.0, angle, result;
895             Vector3f left, right;
896             for (int i = 0; i < pointNum; ++i)
897             {
898                 left << r1(3*i+3)-r1(3*i),r1(3*i+4)-r1(3*i+1),r1(3*i+5)-r1(3*i+2);
899                 right << r2(3*i+3)-r2(3*i),r2(3*i+4)-r2(3*i+1),r2(3*i+5)-r2(3*i+2);
900                 dotValue = left.dot(right);
901                 leftNorm = left.norm(), rightNorm = right.norm();
902                 if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
903                 {
904                     result = dotValue/*/leftNorm/rightNorm*/;
905                     result = min(1.0,(double)result);
906                     result = max(-1.0,(double)result);
```

```
907                     angle = acos(result);
908                     std::cout << angle << std::endl;
909                     length+=angle;
910                     stdevia+=angle*angle;
911                 }
912                 else
913                 {
914                     angle=M_PI;
915                     length+=angle;
916                     stdevia+=angle*angle;
917                 }
918             }
919             length /= pointNum;
920             stdevia = stdevia/pointNum-length*length;
921             if(stdevia>0)
922                 stdevia = sqrt(stdevia/pointNum-length*length);
923             else
924                 stdevia = 1.0e-4;
925             length*=stdevia;
926         }
927         break;
928
929     case 11:
930         {
931             float dotPro = r1.dot(r2);
932             float firstNorm = r1.norm();
933             float secondNorm = r2.norm();
934             float firstInverse, secondInverse;
935             if(firstNorm<1.0e-8)
936                 firstInverse = 1.0e8;
937             else
938                 firstInverse = 1.0/firstNorm;
939             if(secondNorm<1.0e-8)
940                 secondInverse = 1.0e8;
941             else
942                 secondInverse = 1.0/secondNorm;
943             dotPro = dotPro*firstInverse*secondInverse;
944             dotPro = std::max(dotPro, float(-1.0));
945             dotPro = std::min(dotPro, float(1.0));
946             length = acos(dotPro)/M_PI;
947         }
948         break;
949     }
950
951     return length;
952 }
```

5.4.1.33 const float getPathline_MCP (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1910 of file Distance.cpp.

```
1912 {
1913     /* preset the initial time step is 0, then 1, 2, ... as long as it will be normalized */
1914     const int& t_M = first.size()/3-1;
1915     float dist = 0.0, a, b, c;
1916     Eigen::Vector3f temp, another, diff;
1917     for(int i=0; i<t_M; ++i)
1918     {
1919         temp=Eigen::Vector3f(first(i*3)-second(i*3), first(3*i+1)-second(3*i+1), first(3*i+2)-second(3*i+2)
      );
1920         another=Eigen::Vector3f(first(i*3+3)-second(i*3+3), first(3*i+4)-second(3*i+4), first(3*i+5)-second
      (3*i+5));
1921         diff=another-temp;
1922
1923         a=temp.transpose()*temp;
1924         b=temp.transpose()*diff;
1925         c=diff.transpose()*diff;
1926
1927         dist+=get_calculus(a, b, c);
1928     }
1929     return dist/t_M;
1930 }
```

5.4.1.34 const float getProcrustesMetric (const Eigen::VectorXf & *first,* const Eigen::VectorXf & *second*)

Definition at line 1557 of file Distance.cpp.

```
1559 {
1560     assert(first.size()==second.size());
1561 
1562     const int& vertexCount = first.size()/3;
1563 
1564     const int& vertexChanged = vertexCount-2*(PROCRUSTES_SIZE/2);
1565     const int& newSize = 3*vertexChanged;
1566 
1567     /* assign the segment list */
1568     Eigen::MatrixXf firstSegment(PROCRUSTES_SIZE,3), secondSegment(
      PROCRUSTES_SIZE,3), X0;
1569 
1570     int location, rightIndex;
1571 
1572     Eigen::Vector3f first_average, second_average, tempPoint;
1573 
1574     /* A is SVD target, rotation is optimal rotation matrix, and secondPrime is P' after superimposition */
1575     Eigen::MatrixXf A, rotation, secondPrime = Eigen::MatrixXf(PROCRUSTES_SIZE,3);
1576 
1577     float optimalScaling, traceA, pointDist;
1578 
1579     float result = 0.0;
1580 
1581     /* for all points, assign to them a point set with size of PROCRUSTES_SIZE neighboring points */
1582     for(int i=0;i<vertexChanged;++i)
1583     {
1584         rightIndex = i+PROCRUSTES_SIZE;
1585 
1586         first_average = second_average = Eigen::VectorXf::Zero(3);
1587 
1588         /* get the point set of neighboring 7 points and average */
1589         for(int j=i;j<rightIndex;++j)
1590         {
1591             location = j-i;
1592             for(int k=0;k<3;++k)
1593             {
1594                 firstSegment(location,k)=first(3*j+k);
1595                 secondSegment(location,k)=second(3*j+k);
1596             }
1597 
1598             first_average+=firstSegment.row(location);
1599             second_average+=secondSegment.row(location);
1600         }
1601 
1602         first_average/=PROCRUSTES_SIZE;
1603         second_average/=PROCRUSTES_SIZE;
1604 
1605         /* reserve the matrix */
1606         X0 = firstSegment;
1607 
1608         /* centralization for the point set */
1609         for(int j=0;j<PROCRUSTES_SIZE;++j)
1610         {
1611             firstSegment.row(j) = firstSegment.row(j)-first_average.transpose();
1612             secondSegment.row(j) = secondSegment.row(j)-second_average.transpose();
1613         }
1614 
1615         /* get ssqX and ssqY */
1616         float ssqX = (firstSegment.cwiseProduct(firstSegment)).sum();
1617         float ssqY = (secondSegment.cwiseProduct(secondSegment)).sum();
1618 
1619         /* check whether negative or not */
1620         assert(ssqX > 0 && ssqY > 0);
1621 
1622         ssqX = sqrt(ssqX);
1623         ssqY = sqrt(ssqY);
1624 
1625         /* scaling for the point set */
1626         firstSegment/=ssqX;
1627         secondSegment/=ssqY;
1628 
1629         /* get the optimal rotational matrix by othogonal Procrutes analysis */
1630         A = firstSegment.transpose()*secondSegment;
1631 
1632         /* perform SVD on A */
1633         JacobiSVD<MatrixXf> svd(A, ComputeThinU | ComputeThinV);
1634 
1635         /* get the optimal 3D rotation */
1636         rotation = svd.matrixV()*(svd.matrixU().transpose());
1637 
```

```
1638         /* get trace for singular value matrix */
1639         traceA = svd.singularValues().sum();
1640 
1641         /* get optimal scaling */
1642         optimalScaling = traceA*ssqX/ssqY;
1643 
1644         /* preset the average to the P' */
1645         for(int j=0;j<PROCRUSTES_SIZE;++j)
1646             secondPrime.row(j) = first_average;
1647 
1648         /* get P' in superimposed space */
1649         secondPrime = ssqX*traceA*secondSegment*rotation+secondPrime;
1650 
1651         /* compute the distance and store them in the std::vector<float> */
1652         pointDist = 0.0;
1653         for(int j=0;j<PROCRUSTES_SIZE;++j)
1654         {
1655             tempPoint = X0.row(j)-secondPrime.row(j);
1656             pointDist+= tempPoint.transpose()*tempPoint;
1657         }
1658 
1659         /* get the average of P(x,y')^2 */
1660 
1661         // either by computing the matrix
1662         //result+=pointDist;
1663 
1664         // or directly using trace of the matrix
1665         float requiredD = 1.0-traceA*traceA;
1666         result+=requiredD*requiredD;
1667     }
1668     return result/vertexChanged;
1669 }
```

5.4.1.35 const float getProcrustesMetricSegment (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1681 of file Distance.cpp.

```
1683 {
1684     assert(first.size()==second.size());
1685 
1686     const int& vertexCount = first.size()/3;
1687 
1688     const int& vertexChanged = vertexCount/PROCRUSTES_SIZE;
1689     const int& newSize = 3*vertexChanged;
1690 
1691     /* assign the segment list */
1692     Eigen::MatrixXf firstSegment(PROCRUSTES_SIZE,3), secondSegment(
    PROCRUSTES_SIZE,3), X0;
1693 
1694     int location, rightIndex;
1695 
1696     Eigen::Vector3f first_average, second_average, tempPoint;
1697 
1698     /* A is SVD target, rotation is optimal rotation matrix, and secondPrime is P' after superimposition */
1699     Eigen::MatrixXf A, rotation, secondPrime = Eigen::MatrixXf(PROCRUSTES_SIZE,3);
1700 
1701     float optimalScaling, traceA, pointDist;
1702 
1703     float result = 0.0;
1704 
1705     int effective = 0;
1706     /* for all points, assign to them a point set with size of PROCRUSTES_SIZE neighboring points */
1707     for(int i=0;i<vertexChanged;++i)
1708     {
1709         rightIndex = PROCRUSTES_SIZE*i+PROCRUSTES_SIZE;
1710 
1711         first_average = second_average = Eigen::VectorXf::Zero(3);
1712 
1713         /* get the point set of neighboring 7 points and average */
1714         for(int j=PROCRUSTES_SIZE*i;j<rightIndex;++j)
1715         {
1716             location = j-PROCRUSTES_SIZE*i;
1717             for(int k=0;k<3;++k)
1718             {
1719                 firstSegment(location,k)=first(3*j+k);
1720                 secondSegment(location,k)=second(3*j+k);
1721             }
1722 
1723             first_average+=firstSegment.row(location);
1724             second_average+=secondSegment.row(location);
```

```
1725         }
1726
1727         first_average/=PROCRUSTES_SIZE;
1728         second_average/=PROCRUSTES_SIZE;
1729
1730         /* reserve the matrix */
1731         X0 = firstSegment;
1732
1733         /* centralization for the point set */
1734         for(int j=0;j<PROCRUSTES_SIZE;++j)
1735         {
1736             firstSegment.row(j) = firstSegment.row(j)-first_average.transpose();
1737             secondSegment.row(j) = secondSegment.row(j)-second_average.transpose();
1738         }
1739
1740         /* get ssqX and ssqY */
1741         float ssqX = (firstSegment.cwiseProduct(firstSegment)).sum();
1742         float ssqY = (secondSegment.cwiseProduct(secondSegment)).sum();
1743
1744         /* check whether negative or not */
1745         assert(ssqX > 0 && ssqY > 0);
1746
1747         if(ssqX<1.0e-14 || ssqY<1.0e-14)
1748             continue;
1749
1750         ssqX = sqrt(ssqX);
1751         ssqY = sqrt(ssqY);
1752
1753         if(ssqX<1.0e-8 || ssqY<1.0e-8)
1754             continue;
1755
1756         /* scaling for the point set */
1757         firstSegment/=ssqX;
1758         secondSegment/=ssqY;
1759
1760         /* get the optimal rotational matrix by othogonal Procrutes analysis */
1761         A = firstSegment.transpose()*secondSegment;
1762
1763         /* perform SVD on A */
1764         JacobiSVD<MatrixXf> svd(A, ComputeThinU | ComputeThinV);
1765
1766         /* get the optimal 3D rotation */
1767         rotation = svd.matrixV()*(svd.matrixU().transpose());
1768
1769         /* get trace for singular value matrix */
1770         traceA = svd.singularValues().sum();
1771
1772         /* get optimal scaling */
1773         optimalScaling = traceA*ssqX/ssqY;
1774
1775         /* preset the average to the P' */
1776         for(int j=0;j<PROCRUSTES_SIZE;++j)
1777             secondPrime.row(j) = first_average;
1778
1779         /* get P' in superimposed space */
1780         secondPrime = ssqX*traceA*secondSegment*rotation+secondPrime;
1781
1782         /* compute the distance and store them in the std::vector<float> */
1783         pointDist = 0.0;
1784         for(int j=0;j<PROCRUSTES_SIZE;++j)
1785         {
1786             tempPoint = X0.row(j)-secondPrime.row(j);
1787             pointDist+= tempPoint.transpose()*tempPoint;
1788         }
1789         /* get the average of P(x,y')^2 */
1790         result+=pointDist;
1791         ++effective;
1792     }
1793
1794     if(effective==0)
1795     {
1796         return 1.0e-8;
1797     }
1798     else
1799         return result/effective;
1800 }
```

5.4.1.36 const float getRotation (const std::vector< vector< float > > & *streamline*, std::vector< float > & *rotation*)

Definition at line 1406 of file Distance.cpp.

```
1407 {
1408     if(streamline.empty())
1409         return -1;
1410     float result = 0, eachSum;
1411     const int& size = streamline.size();
1412     rotation = std::vector<float>(size);
1413     std::vector<float> eachLine;
1414     Eigen::Vector3f first, second;
1415     int lineSize;
1416     for(int i=0;i<size;++i)
1417     {
1418         eachSum = 0;
1419         eachLine = streamline[i];
1420         lineSize = eachLine.size()/3-2;
1421         // calculate the summation of discrete curvatures
1422         for(int j=0;j<lineSize;++j)
1423         {
1424             first<<eachLine[3*j+3]-eachLine[3*j],eachLine[3*j+4]-eachLine[3*j+1],eachLine[3*j+5]-eachLine[3
    *j+2];
1425             second<<eachLine[3*j+6]-eachLine[3*j+3],eachLine[3*j+7]-eachLine[3*j+4],eachLine[3*j+8]-
    eachLine[3*j+5];
1426
1427             float firstNorm = first.norm(), secondNorm = second.norm();
1428             if(firstNorm>=1.0e-8 && secondNorm>=1.0e-8)
1429             {
1430                 float angle = first.dot(second)/firstNorm/secondNorm;
1431                 angle = std::max(angle,float(-1.0));
1432                 angle = std::min(angle,float(1.0));
1433                 eachSum+=acos(angle);
1434             }
1435         }
1436         // get the mean of discrete curvatures
1437         rotation[i]=eachSum;
1438         result+=eachSum;
1439     }
1440     result/=size;
1441     return result;
1442 }
```

5.4.1.37 const float getSignatureMetric (const Eigen::VectorXf & *firstArray*, const Eigen::VectorXf & *secondArray*, const std::vector< float > & *firstHist*, const std::vector< float > & *secondHist*)

Definition at line 1454 of file Distance.cpp.

```
1458 {
1459     /* would choose alpha = 0.5, and 10% of subset vertices for mean_dist */
1460     const float& Alpha = 0.5;
1461     const int& SUBSET = 10;
1462
1463     /* assert whether the size is the same */
1464     assert(firstArray.size()==secondArray.size());
1465     assert(firstHist.size()==secondHist.size());
1466
1467     const int& histSize = firstHist.size();
1468     const int& vertexCount = firstArray.size()/3;
1469     const int& size = vertexCount/SUBSET+1;
1470
1471     Eigen::VectorXf firstSubset(3*size), secondSubset(3*size);
1472
1473     /* get mean_dist between two sampled subsets */
1474     int tempPos = 0;
1475     for(int i=0;i<vertexCount;i+=SUBSET)
1476     {
1477         for(int j=0;j<3;++j)
1478         {
1479             firstSubset(3*tempPos+j)=firstArray(3*i+j);
1480             secondSubset(3*tempPos+j)=secondArray(3*i+j);
1481         }
1482         ++tempPos;
1483     }
1484
1485     /* get mean_dist */
1486     float result = getMetric_MOP(firstSubset, secondSubset);
1487
1488     float chi_test = 0.0, histDiff, histSum;
1489
1490     /* get chi_test for two histograms */
1491     for(int i=0;i<histSize;++i)
1492     {
```

```
1493         histDiff = firstHist[i]-secondHist[i];
1494         histSum = firstHist[i]+secondHist[i];
1495         /* check numerical error */
1496         if(histSum<1.0e-8)
1497             continue;
1498 
1499         chi_test+= histDiff*histDiff/histSum;
1500     }
1501 
1502     /* get combined distance */
1503     result = (1-Alpha)*chi_test + Alpha*result;
1504 
1505     return result;
1506 }
```

5.4.1.38 const float getSignatureMetric (const Eigen::VectorXf & *centroid,* const Eigen::VectorXf & *first,* const std::vector< float > & *firstHist*)

Definition at line 1517 of file Distance.cpp.

```
1520 {
1521     std::vector<float> centroidHist;
1522     /* get the bin-based histogram for signature */
1523     getSignatureHist(centroid, BIN_SIZE, centroidHist);
1524 
1525     return getSignatureMetric(centroid,first,centroidHist,firstHist);
1526 }
```

5.4.1.39 const float getSignatureMetric (const Eigen::VectorXf & *first,* const Eigen::VectorXf & *second*)

Definition at line 1536 of file Distance.cpp.

```
1538 {
1539     std::vector<float> firstHist, secondHist;
1540     /* get the bin-based histogram for signature */
1541     getSignatureHist(first, BIN_SIZE, firstHist);
1542     getSignatureHist(second, BIN_SIZE, secondHist);
1543 
1544     return getSignatureMetric(first,second,firstHist,secondHist);
1545 }
```

5.4.2 Variable Documentation

5.4.2.1 float** distanceMatrix = NULL

Definition at line 39 of file Distance.cpp.

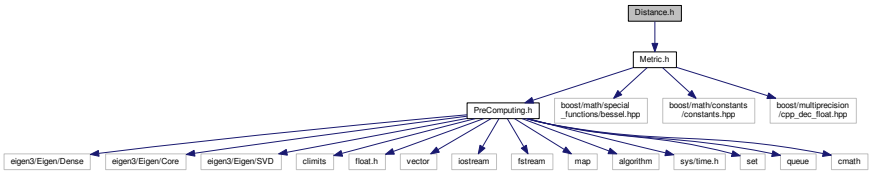
5.4.2.2 const int& PROCRUSTES_SIZE = 7

Definition at line 33 of file Distance.cpp.

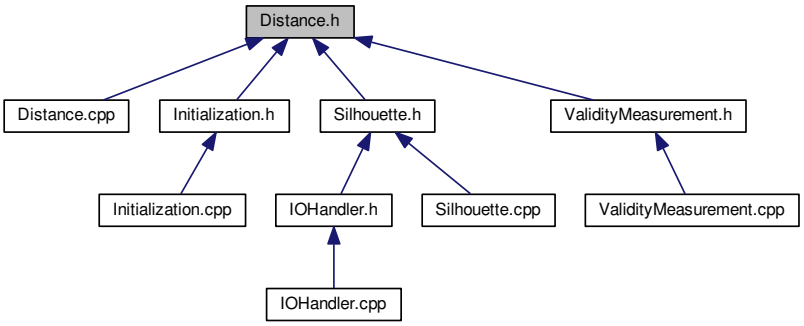
5.5 Distance.h File Reference

```
#include "Metric.h"
```

Include dependency graph for Distance.h:



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



Functions

- const float getBMetric_3 (const VectorXf &row, const int &size, const int &i, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_3 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_3 (const int &first, const int &second, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_6 (const VectorXf &row, const int &size, const int &i, const std::vector< MultiVariate > &normalMultivariate)
- const float getBMetric_6 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_6 (const int &first, const int &second, const std::vector< MultiVariate > &normal↩ Multivariate)
- const float getBMetric_7 (const VectorXf &row, const int &size, const int &i, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_7 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_7 (const int &first, const int &second, const std::vector< std::vector< float > > &rotationSequence)
- const float getBMetric_9 (const VectorXf &row, const int &size, const int &i, const std::vector< MultiVariate > &normalMultivariate)
- const float getBMetric_9 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getBMetric_9 (const int &first, const int &second, const std::vector< MultiVariate > &normal↩ Multivariate)

- const float getBMetric (const std::vector< float > &first, const std::vector< float > &second)
- const float getBMetric (const MultiVariate &first, const MultiVariate &second)
- const float getMetric_10 (const VectorXf &centroid, const int &size, const int &index, const std::vector< VectorXf > &unitLength)
- const float getMetric_10 (const VectorXf &firstRow, const int &size, const VectorXf &secondRow)
- const float getMetric_10 (const int &first, const int &second, const std::vector< VectorXf > &unitLength)
- const float getMetric_MOP (const VectorXf &first, const VectorXf &second)
- const float getMetric_Hausdorff (const VectorXf &first, const VectorXf &second)
- const float getNorm (const Eigen::VectorXf &centroid, const Eigen::VectorXf &r2, const int &index, const int &normOption, const std::vector< std::vector< float > > &object, const std::vector< std::vector< float > > &objectNorm)
- const float getNorm (const VectorXf &centroid, const VectorXf &r2, const int &firstIndex, const int &secondIndex, const int &normOption, const std::vector< std::vector< float > > &object, const std::vector< std::vector< float > > &objectNorm)
- const float getNorm (const VectorXf &r1, const VectorXf &r2, const int &normOption)
- const float getSignatureMetric (const Eigen::VectorXf &firstArray, const Eigen::VectorXf &secondArray, const std::vector< float > &firstHist, const std::vector< float > &secondHist)
- const float getSignatureMetric (const Eigen::VectorXf &centroid, const Eigen::VectorXf &first, const std::vector< float > &firstHist)
- const float getSignatureMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getProcrustesMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getProcrustesMetricSegment (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getEntropyMetric (const std::vector< float > &firstEntropy, const std::vector< float > &secondEntropy)
- const float getEntropyMetric (const std::vector< float > &firstEntropy, const Eigen::VectorXf &array)
- const float getEntropyMetric (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getPathline_MCP (const Eigen::VectorXf &first, const Eigen::VectorXf &second)
- const float getDisimilarity (const MatrixXf &data, const int &first, const int &second, const int &normOption, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &others, const MatrixXf &data, const int &index, const int &normOption, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &first, const VectorXf &second, const int &firstIndex, const int &secondIndex, const int &normOption, const MetricPreparation &object)
- void getDistanceMatrix (const MatrixXf &data, const int &normOption, const MetricPreparation &object)
- const float getDisimilarity (const VectorXf &first, const VectorXf &second, const int &normOption, const MetricPreparation &object)
- void deleteDistanceMatrix (const int &Row)
- const float getRotation (const std::vector< vector< float > > &streamline, std::vector< float > &rotation)
- void generateGroups (const std::vector< std::vector< int > > &storage)

Variables

- const int & PROCRUSTES_SIZE
- float ** distanceMatrix

5.5.1 Function Documentation

5.5.1.1 void deleteDistanceMatrix (const int & *Row*)

Definition at line 1382 of file Distance.cpp.

```
1383 {
1384     if(distanceMatrix)
1385     {
1386     #pragma omp parallel for schedule(static) num_threads(8)
1387         for (int i = 0; i < Row; ++i)
1388         {
1389             if(distanceMatrix[i])
1390             {
1391                 delete[] distanceMatrix[i];
1392                 distanceMatrix[i] = NULL;
1393             }
1394         }
1395         delete[] distanceMatrix;
1396         distanceMatrix = NULL;
1397     }
1398 }
```

5.5.1.2 void generateGroups (const std::vector< std::vector< int > > & *storage*)

Definition at line 1808 of file Distance.cpp.

```
1809 {
1810     if(storage.empty())
1811         return;
1812     std::ofstream readme("../dataset/Storage",ios::out|ios::app);
1813     if(!readme)
1814     {
1815         std::cout << "Error creating Storage!" << std::endl;
1816         exit(1);
1817     }
1818
1819     readme << std::endl;
1820     const int& groupSize = storage.size();
1821     std::vector<int> element;
1822     for(int i=0;i<groupSize;++i)
1823     {
1824         element = storage[i];
1825         if(element.empty())
1826             continue;
1827         for(int j=0;j<element.size();++j)
1828             readme << element[j] << " ";
1829         readme << std::endl;
1830     }
1831     std::cout << std::endl;
1832     readme.close();
1833 }
```

5.5.1.3 const float getBMetric (const std::vector< float > & *first*, const std::vector< float > & *second*)

Definition at line 284 of file Distance.cpp.

```
287 {
288     // calculate mean and standard deviation of the two arrays
289     float u_a, u_b, sig_a, sig_b, sig_a_inverse, sig_b_inverse,
290           summation, sum_inverse, tempDist;
291     u_a = firstNorm3[0], u_b = secondNorm3[0];
292     sig_a = firstNorm3[1], sig_b = secondNorm3[1];
293     sig_a *= sig_a, sig_b *= sig_b;
294     if(sig_a<=1.0e-8)
295     {
296         sig_a = 1.0e-8;
297         sig_a_inverse = 1.0e8;
298     }
299     else
300         sig_a_inverse = 1.0/sig_a;
301     if(sig_b<=1.0e-8)
302     {
303         sig_b = 1.0e-8;
304         sig_b_inverse = 1.0/sig_b;
305     }
306     summation = sig_a+sig_b;
307     sum_inverse = 1.0/summation;
308     tempDist = 0.25*log(0.25*(sig_a*sig_b_inverse
309                +sig_b*sig_a_inverse+2))
310                + 0.25*(u_a-u_b)*(u_a-u_b)*sum_inverse;
311     return tempDist;
312 }
```

5.5.1.4 const float getBMetric (const MultiVariate & *first,* const MultiVariate & *second*)

Definition at line 321 of file Distance.cpp.

```
324 {
325     Matrix3f firstCov, secondCov, meanCov, meanCovInverse;
326     float sqrtInverse, meanCovDet;
327     firstCov = centerNormal.covariance;
328     secondCov = neighNormal.covariance;
329     meanCov = 0.5*(firstCov+secondCov);
330     if(meanCov.determinant()>1.0e-8)
331     {
332         meanCovInverse = static_cast<Matrix3f>(meanCov.inverse());
333         meanCovDet = meanCov.determinant();
334     }
335     else
336     {
337         meanCovInverse = pseudoInverse(meanCov);
338         meanCovDet = 1.0e8;
339     }
340     float detMulti = sqrt(firstCov.determinant()*secondCov.determinant());
341     sqrtInverse = detMulti>1.0e-8?float(1.0)/detMulti:1.0e8;
342     Vector3f meanDiff = centerNormal.meanVec-neighNormal.meanVec;
343     float tempDist = 0.125*meanDiff.transpose()*meanCovInverse*meanDiff
344             +0.2*log(meanCovDet*sqrtInverse);
345     return tempDist;
346 }
```

5.5.1.5 const float getBMetric_3 (const VectorXf & *row,* const int & *size,* const int & *i,* const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 51 of file Distance.cpp.

```
56 {
57     std::vector<float> firstNorm3, secondNorm3;
58     getSequence(row, size, firstNorm3);
59     secondNorm3 = rotationSequence[i];
60     return getBMetric(firstNorm3, secondNorm3);
61 }
```

5.5.1.6 const float getBMetric_3 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 72 of file Distance.cpp.

```
76 {
77     std::vector<float> firstNorm3, secondNorm3;
78     getSequence(firstRow, size, firstNorm3);
79     getSequence(secondRow, size, secondNorm3);
80     return getBMetric(firstNorm3, secondNorm3);
81 }
```

5.5.1.7 const float getBMetric_3 (const int & *first,* const int & *second,* const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 92 of file Distance.cpp.

```
96 {
97     return getBMetric(rotationSequence[first], rotationSequence[second]);
98 }
```

5.5.1.8 const float getBMetric_6 (const VectorXf & *row,* const int & *size,* const int & *i,* const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 110 of file Distance.cpp.

```
115 {
116     MultiVariate centerNormal, neighNormal;
117     getNormalMultivariate(row, size, centerNormal);
118     neighNormal = normalMultivariate[i];
119     return getBMetric(centerNormal, neighNormal);
120 }
```

5.5.1.9 const float getBMetric_6 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 131 of file Distance.cpp.

```
135 {
136     MultiVariate centerNormal, neighNormal;
137     getNormalMultivariate(firstRow, size, centerNormal);
138     getNormalMultivariate(secondRow, size, neighNormal);
139     return getBMetric(centerNormal, neighNormal);
140 }
```

5.5.1.10 const float getBMetric_6 (const int & *first,* const int & *second,* const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 151 of file Distance.cpp.

```
155 {
156     return getBMetric(normalMultivariate[first], normalMultivariate[second]);
157 }
```

5.5.1.11 const float getBMetric_7 (const VectorXf & *row,* const int & *size,* const int & *i,* const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 169 of file Distance.cpp.

```
174 {
175     std::vector<float> firstNorm3, secondNorm3;
176     getEachFixedSequence(row, size, firstNorm3);
177     secondNorm3 = rotationSequence[i];
178     return getBMetric(firstNorm3, secondNorm3);
179 }
```

5.5.1.12 const float getBMetric_7 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 190 of file Distance.cpp.

```
194 {
195     std::vector<float> firstNorm3, secondNorm3;
196     getEachFixedSequence(firstRow, size, firstNorm3);
197     getEachFixedSequence(secondRow, size, secondNorm3);
198     return getBMetric(firstNorm3, secondNorm3);
199 }
```

5.5.1.13 const float getBMetric_7 (const int & *first*, const int & *second*, const std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 210 of file Distance.cpp.

```
214 {
215     return getBMetric(rotationSequence[first], rotationSequence[second]);
216 }
```

5.5.1.14 const float getBMetric_9 (const VectorXf & *row*, const int & *size*, const int & *i*, const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 228 of file Distance.cpp.

```
233 {
234     MultiVariate centerNormal, neighNormal;
235     getUnnormalizedMultivariate(row, size, centerNormal);
236     neighNormal = normalMultivariate[i];
237     return getBMetric(centerNormal, neighNormal);
238 }
```

5.5.1.15 const float getBMetric_9 (const VectorXf & *firstRow*, const int & *size*, const VectorXf & *secondRow*)

Definition at line 249 of file Distance.cpp.

```
253 {
254     MultiVariate centerNormal, neighNormal;
255     getUnnormalizedMultivariate(firstRow, size, centerNormal);
256     getUnnormalizedMultivariate(secondRow, size, neighNormal);
257     return getBMetric(centerNormal, neighNormal);
258 }
```

5.5.1.16 const float getBMetric_9 (const int & *first*, const int & *second*, const std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 269 of file Distance.cpp.

```
273 {
274     return getBMetric(normalMultivariate[first], normalMultivariate[second]);
275 }
```

5.5.1.17 const float getDisimilarity (const MatrixXf & *data*, const int & *first*, const int & *second*, const int & *normOption*, const MetricPreparation & *object*)

Definition at line 964 of file Distance.cpp.

```
969 {
970     return getDisimilarity(data.row(first), data.row(second),
971                            first, second, normOption, object);
972 }
```

5.5.1.18 const float getDisimilarity (const VectorXf & *others*, const MatrixXf & *data*, const int & *index*, const int & *normOption*, const **MetricPreparation** & *object*)

Definition at line 985 of file Distance.cpp.

```
990 {
991     float length;
992     switch(normOption)
993     {
994     case 0: // Euclidean distance, d_E
995     case 1: // Fraction norm, d_F
996     case 2: // geometric similarity measure, d_G
997     case 5:
998     case 8:
999     case 11:
1000         length = getNorm(others, data.row(index),index,normOption,
1001                     object.pairwise, object.pairwiseNorm);
1002         break;
1003
1004     case 3:
1005         length = getBMetric_3(others, others.size()/3-2, index,
1006                               object.rotationSequence);
1007         break;
1008
1009     case 4:
1010         length = abs(object.rotation[index]-
1011                  getRotation(others, others.size()/3-2));
1012         break;
1013
1014     case 6:
1015         length = getBMetric_6(others, others.size()/3-1, index,
1016                               object.normalMultivariate);
1017         break;
1018
1019     case 7:
1020         length = getBMetric_7(others, others.size()/3-1, index,
1021                               object.rotationSequence);
1022         break;
1023
1024     case 9:
1025         length = getBMetric_9(others, others.size()/3-1, index,
1026                               object.normalMultivariate);
1027         break;
1028
1029     case 10:
1030         length = getMetric_10(others, others.size()/3, index,
1031                               object.unitLength);
1032         break;
1033
1034     case 12:    // the MCP distance, i.e., d_M
1035         length = getMetric_MOP(others, data.row(index));
1036         break;
1037
1038     case 13:    // the Hausdorff distance, i.e., d_H
1039         length = getMetric_Hausdorff(others, data.row(index));
1040         break;
1041
1042     /* signature-based similarity metric with chi-squared test combined with mean-closest */
1043     case 14:    // the signature-based similarity, i.e., d_S
1044         length = getSignatureMetric(others,data.row(index),object.pairwise[index]);
1045         break;
1046
1047     /* adapted Procrustes distance */
1048     case 15:    // the Procrustes distance, i.e., d_P
1049         //length = getProcrustesMetric(others, data.row(index));
1050         length = std::min(getProcrustesMetricSegment(others, data.row(index)),
1051                  getProcrustesMetricSegment(data.row(index), others));
1052         break;
1053
1054     case 16:
1055         length = getEntropyMetric(object.pairwise[index], others);
1056         break;
1057
1058     case 17:    // the time-based MCP, i.e., d_T
1059         length = getPathline_MCP(others, data.row(index));
1060         break;
1061
1062     default:
1063         exit(1);
1064         break;
1065     }
1066
1067     return length;
1068 }
```

5.5.1.19 const float getDisimilarity (const VectorXf & *first*, const VectorXf & *second*, const int & *firstIndex*, const int & *secondIndex*, const int & *normOption*, const MetricPreparation & *object*)

Definition at line 1082 of file Distance.cpp.

```
1088 {
1089     float length;
1090     switch(normOption)
1091     {
1092     case 0: // Euclidean distance, d_E
1093     case 1: // Fraction norm, d_F
1094     case 2: // Geometric similarity, d_G
1095     case 5:
1096     case 8:
1097     case 11:
1098         length = getNorm(first, second,firstIndex,secondIndex, normOption,
1099                     object.pairwise, object.pairwiseNorm);
1100         break;
1101
1102     case 3:
1103         length = getBMetric_3(firstIndex, secondIndex,
1104                     object.rotationSequence);
1105         break;
1106
1107     case 4:
1108         length = abs(object.rotation[firstIndex]-
1109                     object.rotation[secondIndex]);
1110         break;
1111
1112     case 6:
1113         length = getBMetric_6(firstIndex, secondIndex,
1114                     object.normalMultivariate);
1115         break;
1116
1117     case 7:
1118         length = getBMetric_7(firstIndex, secondIndex,
1119                     object.rotationSequence);
1120         break;
1121
1122     case 9:
1123         length = getBMetric_9(firstIndex, secondIndex,
1124                     object.normalMultivariate);
1125         break;
1126
1127     case 10:
1128         length = getMetric_10(firstIndex, secondIndex,
1129                     object.unitLength);
1130         break;
1131
1132     case 12:    // the MCP distance, d_M
1133         length = getMetric_MOP(first, second);
1134         break;
1135
1136     case 13:    // the Hausdorff distance, d_H
1137         length = getMetric_Hausdorff(first, second);
1138         break;
1139
1140     case 14:    //  the signature-based distance, d_S
1141         length = getSignatureMetric(first,second,object.pairwise[firstIndex],object.
     pairwise[secondIndex]);
1142         break;
1143
1144     case 15:    // the Procrutes distance, d_P
1145         //length = getProcrustesMetric(first, second);
1146         length = std::min(getProcrustesMetricSegment(first,second),
     getProcrustesMetricSegment(second,first));
1147         break;
1148
1149     case 16:
1150         length = getEntropyMetric(object.pairwise[firstIndex], object.pairwise[secondIndex]
     );
1151         break;
1152
1153     case 17:    // the time-based MCP, d_T
1154         length = getPathline_MCP(first, second);
1155         break;
1156
1157     default:
1158         exit(1);
1159         break;
1160     }
1161     return length;
1162 }
```

5.5.1.20 const float getDisimilarity (const VectorXf & *first*, const VectorXf & *second*, const int & *normOption*, const MetricPreparation & *object*)

Definition at line 1174 of file Distance.cpp.

```
1178 {
1179     float length;
1180     switch(normOption)
1181     {
1182     case 0: // Euclidean distance, d_E
1183     case 1: // Fraction norm, d_F
1184     case 2: // Geometric similarity, d_G
1185     case 5:
1186     case 8:
1187     case 11:
1188         length = getNorm(first, second, normOption);
1189         break;
1190
1191     case 3:
1192         length = getBMetric_3(first, first.size()/3-2, second);
1193         break;
1194
1195     case 4:
1196         length = abs(getRotation(first, first.size()/3-2)-getRotation(second, second.
     size()/3-2));
1197         break;
1198
1199     case 6:
1200         length = getBMetric_6(first, first.size()/3-1,second);
1201         break;
1202
1203     case 7:
1204         length = getBMetric_7(first, first.size()/3-1, second);
1205         break;
1206
1207     case 9:
1208         length = getBMetric_9(first, first.size()/3-1, second);
1209         break;
1210
1211     case 10:
1212         length = getBMetric_9(first, first.size()/3, second);
1213         break;
1214
1215     case 12:    // the MCP distance, d_M
1216         length = getMetric_MOP(first, second);
1217         break;
1218
1219     case 13:    // the Hausdorff distance, d_H
1220         length = getMetric_Hausdorff(first, second);
1221         break;
1222
1223     /* signature-based similarity metric with chi-squared test combined with mean-closest */
1224     case 14:    // the signature-based similarity, d_S
1225         length = getSignatureMetric(first, second);
1226         break;
1227
1228     /* adapted Procrustes distance */
1229     case 15:    // the Procrustes distance, d_P
1230         //length = getProcrustesMetric(first, second);
1231         length = getProcrustesMetricSegment(first,second);
1232         break;
1233
1234     case 16:
1235         length = getEntropyMetric(first, second);
1236         break;
1237
1238     case 17:    // the time-based MCP, d_T
1239         length = getPathline_MCP(first, second);
1240         break;
1241
1242     default:
1243         exit(1);
1244         break;
1245     }
1246
1247     return length;
1248 }
```

5.5.1.21 void getDistanceMatrix (const MatrixXf & *data*, const int & *normOption*, const MetricPreparation & *object*)

Definition at line 1348 of file Distance.cpp.

```
1351 {
1352     const int& Row = data.rows();
1353     distanceMatrix = new float*[Row];
1354 
1355     // assign the distance matrix
1356 #pragma omp parallel for schedule(static) num_threads(8)
1357     for (int i = 0; i < Row; ++i)
1358     {
1359         distanceMatrix[i] = new float[Row];
1360         for (int j = 0; j < Row; ++j)
1361         {
1362             /* don't wish to waste computation on diagonal element */
1363             if(i==j)
1364                 distanceMatrix[i][j] = 0.0;
1365             else
1366                 distanceMatrix[i][j] = getDisimilarity(data, i, j, normOption,
       object);
1367         }
1368     }
1369     // help check whether they already been assigned and whether they are symmetric or not
1370     std::cout << "Distance between 215 and 132 is " << distanceMatrix[215][132] << std::endl;
1371     std::cout << "Distance between 132 and 215 is " << distanceMatrix[132][215] << std::endl;
1372 
1373     std::cout << "Finished computing distance matrix!" << std::endl;
1374 }
```

5.5.1.22 const float getEntropyMetric (const std::vector< float > & *firstEntropy*, const std::vector< float > & *secondEntropy*)

Definition at line 1845 of file Distance.cpp.

```
1847 {
1848     assert(firstEntropy.size()==2);
1849     assert(secondEntropy.size()==2);
1850 
1851     float first = firstEntropy[0]-secondEntropy[0];
1852     float second = firstEntropy[1]-secondEntropy[1];
1853 
1854     return sqrt(first*first+second*second);
1855 }
```

5.5.1.23 const float getEntropyMetric (const std::vector< float > & *firstEntropy*, const Eigen::VectorXf & *array*)

Definition at line 1868 of file Distance.cpp.

```
1870 {
1871     assert(firstEntropy.size()==2);
1872 
1873     std::vector<float> secondEntropy;
1874 
1875     getLinearAngularEntropy(array, BUNDLE_SIZE, secondEntropy);
1876 
1877     return getEntropyMetric(firstEntropy, secondEntropy);
1878 }
```

5.5.1.24 const float getEntropyMetric (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1890 of file Distance.cpp.

```
1892 {
1893 
1894     std::vector<float> firstEntropy, secondEntropy;
1895 
1896     getLinearAngularEntropy(first, BUNDLE_SIZE, firstEntropy);
1897     getLinearAngularEntropy(second, BUNDLE_SIZE, secondEntropy);
1898 
1899     return getEntropyMetric(firstEntropy, secondEntropy);
1900 }
```

5.5.1.25 const float getMetric_10 (const VectorXf & *centroid,* const int & *size,* const int & *index,* const std::vector< VectorXf > & *unitLength*)

Definition at line 358 of file Distance.cpp.

```
362 {
363     const VectorXf& x = unitLength[index];
364     VectorXf y(size*3);
365     getUnitDirection_byEach(centroid,size,y);
366
367     float length = x.dot(y)/x.size();
368     length = min(1.0,(double)length);
369     length = max(-1.0,(double)length);
370     length = acos(length);
371     return length;
372 }
```

5.5.1.26 const float getMetric_10 (const VectorXf & *firstRow,* const int & *size,* const VectorXf & *secondRow*)

Definition at line 383 of file Distance.cpp.

```
386 {
387     VectorXf x(size*3), y(size*3);
388     getUnitDirection_byEach(firstRow,size,x);
389     getUnitDirection_byEach(secondRow,size,y);
390
391     float length = x.dot(y)/x.size();
392     length = min(1.0,(double)length);
393     length = max(-1.0,(double)length);
394     length = acos(length);
395     return length;
396 }
```

5.5.1.27 const float getMetric_10 (const int & *first,* const int & *second,* const std::vector< VectorXf > & *unitLength*)

Definition at line 407 of file Distance.cpp.

```
410 {
411     const VectorXf& x = unitLength[first];
412     const VectorXf& y = unitLength[second];
413     float length = x.dot(y)/x.size();
414     length = min(1.0,(double)length);
415     length = max(-1.0,(double)length);
416     length = acos(length);
417     return length;
418 }
```

5.5.1.28 const float getMetric_Hausdorff (const VectorXf & *first,* const VectorXf & *second*)

Definition at line 1305 of file Distance.cpp.

```
1306 {
1307     // the max of first to second
1308     const int& vNum = first.size()/3;
1309     float result, f_to_s=-1.0, s_to_f=-1.0;
1310     for(int i=0;i<vNum;++i)
1311     {
1312         float minDist = FLT_MAX;
1313         Vector3f m_i = Vector3f(first(3*i),first(3*i+1),first(3*i+2));
1314         for(int j=0;j<vNum;++j)
1315         {
1316             Vector3f n_j = Vector3f(second(3*j),second(3*j+1),second(3*j+2));
1317             minDist = std::min((m_i-n_j).norm(),minDist);
1318         }
1319         s_to_f=std::max(s_to_f, minDist);
1320     }
1321
1322     // the max of second to first
1323     for(int i=0;i<vNum;++i)
1324     {
1325         float minDist = FLT_MAX;
1326         Vector3f m_i = Vector3f(second(3*i),second(3*i+1),second(3*i+2));
1327         for(int j=0;j<vNum;++j)
1328         {
1329             Vector3f n_j = Vector3f(first(3*j),first(3*j+1),first(3*j+2));
1330             minDist = std::min((m_i-n_j).norm(),minDist);
1331         }
1332         f_to_s=std::max(f_to_s, minDist);
1333     }
1334
1335     // max of the max
1336     result = std::max(f_to_s, s_to_f);
1337     return result;
1338 }
```

5.5.1.29 const float getMetric_MOP (const VectorXf & *first*, const VectorXf & *second*)

Definition at line 1258 of file Distance.cpp.

```
1259 {
1260     // The MCP of first to second
1261     const int& vNum = first.size()/3;
1262     float result, f_to_s, s_to_f;
1263     float summation = 0;
1264     for(int i=0;i<vNum;++i)
1265     {
1266         float minDist = FLT_MAX;
1267         Vector3f m_i = Vector3f(first(3*i),first(3*i+1),first(3*i+2));
1268         for(int j=0;j<vNum;++j)
1269         {
1270             Vector3f n_j = Vector3f(second(3*j),second(3*j+1),second(3*j+2));
1271             minDist = std::min((m_i-n_j).norm(),minDist);
1272         }
1273         summation+=minDist;
1274     }
1275     s_to_f = summation/vNum;
1276
1277     // The MCP of second to first
1278     summation = 0;
1279     for(int i=0;i<vNum;++i)
1280     {
1281         float minDist = FLT_MAX;
1282         Vector3f m_i = Vector3f(second(3*i),second(3*i+1),second(3*i+2));
1283         for(int j=0;j<vNum;++j)
1284         {
1285             Vector3f n_j = Vector3f(first(3*j),first(3*j+1),first(3*j+2));
1286             minDist = std::min((m_i-n_j).norm(),minDist);
1287         }
1288         summation+=minDist;
1289     }
1290     f_to_s = summation/vNum;
1291
1292     // get the average of that
1293     result = (f_to_s+s_to_f)/2.0;
1294     return result;
1295 }
```

5.5.1.30 const float getNorm (const Eigen::VectorXf & *centroid*, const Eigen::VectorXf & *r2*, const int & *index*, const int & *normOption*, const std::vector< std::vector< float > > & *object*, const std::vector< std::vector< float > > & *objectNorm*)

Definition at line 432 of file Distance.cpp.

```
438 {
439     assert(centroid.size()==r2.size());
440     float length = 0.0;
441     switch(normOption)
442     {
443     case 0: // Euclidean distance
444     default:
445         length = (centroid-r2).norm();
446         break;
447 
448     case 11:    // the norm 11
449         {
450             float dotPro = centroid.dot(r2);
451             float firstNorm = centroid.norm();
452             float secondNorm = r2.norm();
453             float firstInverse, secondInverse;
454             if(firstNorm<1.0e-8)
455                 firstInverse = 1.0e8;
456             else
457                 firstInverse = 1.0/firstNorm;
458             if(secondNorm<1.0e-8)
459                 secondInverse = 1.0e8;
460             else
461                 secondInverse = 1.0/secondNorm;
462             dotPro = dotPro*firstInverse*secondInverse;
463             dotPro = std::max(dotPro, float(-1.0));
464             dotPro = std::min(dotPro, float(1.0));
465             length = acos(dotPro)/M_PI;
466         }
467         break;
468 
469     case 1:  /* fraction norm by high-dimensional feature-space */
470         {
471             for (int i = 0; i < centroid.size(); ++i)
472             {
473                 length += pow(abs(centroid(i)-r2(i)),0.5);
474             }
475             length = pow(length,2.0);
476         }
477         break;
478 
479     case 2: /* mean value of dot product value, which means it's rotational invariant, or d_G */
480         {
481             const int& pointNum = centroid.size()/3-1;
482             float dotValue, leftNorm, rightNorm, result;
483 
484             std::vector<float> centroidWise;
485             std::vector<float> centroidWiseNorm;
486             getPairWise_byEach(centroid, pointNum, centroidWise, centroidWiseNorm);
487 
488             const std::vector<float>& i_Pairwise = pairwise[index];
489             const std::vector<float>& i_PairNorm = objectNorm[index];
490 
491             Vector3f left, right;
492             for (int i = 0; i < pointNum; ++i)
493             {
494                 leftNorm = centroidWiseNorm[i];
495                 rightNorm = i_PairNorm[i];
496                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
497                 {
498                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
499                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
500                     result = left.dot(right)/*/leftNorm/rightNorm*/;
501                     result = min(1.0,(double)result);
502                     result = max(-1.0,(double)result);
503                     length+=acos(result);
504                 }
505                 else
506                     length+=M_PI;
507             }
508             length /= pointNum;
509         }
510         break;
511 
512     case 5: /* rotational invariant line-wise acos angle with normal direction for
513                measuring whether counterclockwise or clockwise orientation */
514         {
```

```
515             const int& pointNum = centroid.size()/3-1;
516             float dotValue, leftNorm, rightNorm, normalDot, result;
517             Vector3f left, right, normal;
518 
519             std::vector<float> centroidWise;
520             std::vector<float> centroidNorm;
521             getPairWise_byEach(centroid, pointNum, centroidWise, centroidNorm);
522             const std::vector<float>& i_Pairwise = pairwise[index];
523             const std::vector<float>& i_PairNorm = objectNorm[index];
524 
525             left << /*centroid(3)-centroid(0),centroid(4)-centroid(1),centroid(5)-centroid(2)*/
526                     centroidWise[0], centroidWise[1], centroidWise[2];
527             right << /*r2(3)-r2(0),r2(4)-r2(1),r2(5)-r2(2)*/
528                     i_Pairwise[0], i_Pairwise[1], i_Pairwise[2];
529             const Vector3f& Normal = left.cross(right);
530 
531             for (int i = 0; i < pointNum; ++i)
532             {
533 
534                 leftNorm = centroidNorm[i];
535                 rightNorm = i_PairNorm[i];
536                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
537                 {
538                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
539                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
540                     result = left.dot(right)/*/leftNorm/rightNorm*/;
541                     result = min(1.0,(double)result);
542                     result = max(-1.0,(double)result);
543                     normal = left.cross(right);
544                     normalDot = Normal.dot(normal);
545                     if(normalDot<0)
546                         length+=-acos(result);
547                     else
548                         length+=acos(result);
549                 }
550                 else
551                     length+=M_PI;
552             }
553             length /= pointNum;
554             length = abs(length);
555         }
556         break;
557 
558     case 8: /* distance metric defined as mean * standard deviation */
559         {
560             const int& pointNum = centroid.size()/3-1;
561             float dotValue, leftNorm, rightNorm, stdevia = 0.0, angle, result;
562             Vector3f left, right;
563 
564             std::vector<float> centroidWise;
565             std::vector<float> centroidNorm;
566             getPairWise_byEach(centroid, pointNum, centroidWise, centroidNorm);
567 
568             const std::vector<float>& i_Pairwise = pairwise[index];
569             const std::vector<float>& i_PairNorm = objectNorm[index];
570 
571             for (int i = 0; i < pointNum; ++i)
572             {
573                 leftNorm = centroidNorm[i];
574                 rightNorm = i_PairNorm[i];
575 
576                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
577                 {
578                     left << centroidWise[3*i], centroidWise[3*i+1], centroidWise[3*i+2];
579                     right << i_Pairwise[3*i],i_Pairwise[3*i+1],i_Pairwise[3*i+2];
580                     result = left.dot(right)/*/leftNorm/rightNorm*/;
581                     result = min(1.0,(double)result);
582                     result = max(-1.0,(double)result);
583                     angle = acos(result);
584                     length+=angle;
585                     stdevia+=angle*angle;
586                 }
587                 else
588                 {
589                     angle=M_PI;
590                     length+=angle;
591                     stdevia+=angle*angle;
592                 }
593             }
594             length /= pointNum;
595             stdevia = stdevia/pointNum-length*length;
596             if(stdevia>0)
597                 stdevia = sqrt(stdevia/pointNum-length*length);
598             else
599                 stdevia = 1.0e-4;
600         }
601         break;
```

```
602     }
603
604     return length;
605 }
```

5.5.1.31 const float getNorm (const VectorXf & *centroid,* const VectorXf & *r2,* const int & *firstIndex,* const int & *secondIndex,* const int & *normOption,* const std::vector< std::vector< float > > & *object,* const std::vector< std::vector< float > > & *objectNorm*)

Definition at line 620 of file Distance.cpp.

```
627 {
628     assert(centroid.size()==r2.size());
629     float length = 0.0;
630     switch(normOption)
631     {
632     case 0:
633     default:
634         length = (centroid-r2).norm();
635         break;
636
637     case 1:  /* fraction norm by high-dimensional feature-space, or d_F */
638         {
639             for (int i = 0; i < centroid.size(); ++i)
640             {
641                 length += pow(abs(centroid(i)-r2(i)),0.5);
642             }
643             length = pow(length,2.0);
644         }
645         break;
646
647     case 11:
648         {
649             float dotPro = centroid.dot(r2);
650             float firstNorm = centroid.norm();
651             float secondNorm = r2.norm();
652             float firstInverse, secondInverse;
653             if(firstNorm<1.0e-8)
654                 firstInverse = 1.0e8;
655             else
656                 firstInverse = 1.0/firstNorm;
657             if(secondNorm<1.0e-8)
658                 secondInverse = 1.0e8;
659             else
660                 secondInverse = 1.0/secondNorm;
661             dotPro = dotPro*firstInverse*secondInverse;
662             dotPro = std::max(dotPro, float(-1.0));
663             dotPro = std::min(dotPro, float(1.0));
664             length = acos(dotPro)/M_PI;
665         }
666         break;
667
668     case 2: /* mean value of dot product value, which means it's rotational invariant, or d_G */
669         {
670             const int& pointNum = centroid.size()/3-1;
671             float dotValue, leftNorm, rightNorm, result;
672
673             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
674             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
675
676             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
677             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
678
679             Vector3f left, right;
680             for (int i = 0; i < pointNum; ++i)
681             {
682                 leftNorm = i_PairNorm[i];
683                 rightNorm = j_PairNorm[i];
684                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
685                 {
686                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
687                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
688                     result = left.dot(right)/*/leftNorm/rightNorm*/;
689                     result = min(1.0,(double)result);
690                     result = max(-1.0,(double)result);
691                     length+=acos(result);
692                 }
693                 else
694                     length+=M_PI;
```

```
695             }
696             length /= pointNum;
697         }
698         break;
699 
700     case 5: /* rotational invariant line-wise acos angle with normal direction for
701                measuring whether counterclockwise or clockwise orientation */
702         {
703             const int& pointNum = centroid.size()/3-1;
704             float dotValue, leftNorm, rightNorm, normalDot, result;
705             Vector3f left, right, normal;
706 
707             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
708             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
709 
710             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
711             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
712 
713             left << /*centroid(3)-centroid(0),centroid(4)-centroid(1),centroid(5)-centroid(2)*/
714                     i_Pairwise[0], i_Pairwise[1], i_Pairwise[2];
715             right << /*r2(3)-r2(0),r2(4)-r2(1),r2(5)-r2(2)*/
716                     j_Pairwise[0], j_Pairwise[1], j_Pairwise[2];
717             const Vector3f& Normal = left.cross(right);
718 
719             for (int i = 0; i < pointNum; ++i)
720             {
721                 leftNorm = i_PairNorm[i];
722                 rightNorm = j_PairNorm[i];
723 
724                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
725                 {
726                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
727                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
728                     result = left.dot(right)/*/leftNorm/rightNorm*/;
729                     result = min(1.0,(double)result);
730                     result = max(-1.0,(double)result);
731                     normal = left.cross(right);
732                     normalDot = Normal.dot(normal);
733                     if(normalDot<0)
734                         length+=-acos(result);
735                     else
736                         length+=acos(result);
737                 }
738                 else
739                     length+=M_PI;
740             }
741             length /= pointNum;
742             length = abs(length);
743         }
744         break;
745 
746     case 8: /* distance metric defined as mean * standard deviation */
747         {
748             const int& pointNum = centroid.size()/3-1;
749             float dotValue, leftNorm, rightNorm, stdevia = 0.0, angle, result;
750             Vector3f left, right;
751 
752             const std::vector<float>& i_Pairwise = pairwise[firstIndex];
753             const std::vector<float>& j_Pairwise = pairwise[secondIndex];
754 
755             const std::vector<float>& i_PairNorm = objectNorm[firstIndex];
756             const std::vector<float>& j_PairNorm = objectNorm[secondIndex];
757 
758             for (int i = 0; i < pointNum; ++i)
759             {
760                 leftNorm = i_PairNorm[i];
761                 rightNorm = j_PairNorm[i];
762 
763                 if(leftNorm >= 1.0e-8 && rightNorm >= 1.0e-8)
764                 {
765                     left << i_Pairwise[3*i], i_Pairwise[3*i+1], i_Pairwise[3*i+2];
766                     right << j_Pairwise[3*i],j_Pairwise[3*i+1],j_Pairwise[3*i+2];
767                     result = left.dot(right)/*/leftNorm/rightNorm*/;
768                     result = min(1.0,(double)result);
769                     result = max(-1.0,(double)result);
770                     angle = acos(result);
771                     length+=angle;
772                     stdevia+=angle*angle;
773                 }
774                 else
775                 {
776                     angle=M_PI;
777                     length+=angle;
778                     stdevia+=angle*angle;
779                 }
780             }
781             length /= pointNum;
```

```
782             stdevia = stdevia/pointNum-length*length;
783             if(stdevia>0)
784                 stdevia = sqrt(stdevia/pointNum-length*length);
785             else
786                 stdevia = 1.0e-4;
787         }
788         break;
789
790     }
791
792     return length;
793 }
```

5.5.1.32 const float getNorm (const VectorXf & *r1*, const VectorXf & *r2*, const int & *normOption*)

5.5.1.33 const float getPathline_MCP (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1910 of file Distance.cpp.

```
1912 {
1913     /* preset the initial time step is 0, then 1, 2, ... as long as it will be normalized */
1914     const int& t_M = first.size()/3-1;
1915     float dist = 0.0, a, b, c;
1916     Eigen::Vector3f temp, another, diff;
1917     for(int i=0; i<t_M; ++i)
1918     {
1919         temp=Eigen::Vector3f(first(i*3)-second(i*3), first(3*i+1)-second(3*i+1), first(3*i+2)-second(3*i+2)
    );
1920         another=Eigen::Vector3f(first(i*3+3)-second(i*3+3), first(3*i+4)-second(3*i+4), first(3*i+5)-second
    (3*i+5));
1921         diff=another-temp;
1922
1923         a=temp.transpose()*temp;
1924         b=temp.transpose()*diff;
1925         c=diff.transpose()*diff;
1926
1927         dist+=get_calculus(a, b, c);
1928     }
1929     return dist/t_M;
1930 }
```

5.5.1.34 const float getProcrustesMetric (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1557 of file Distance.cpp.

```
1559 {
1560     assert(first.size()==second.size());
1561
1562     const int& vertexCount = first.size()/3;
1563
1564     const int& vertexChanged = vertexCount-2*(PROCRUSTES_SIZE/2);
1565     const int& newSize = 3*vertexChanged;
1566
1567     /* assign the segment list */
1568     Eigen::MatrixXf firstSegment(PROCRUSTES_SIZE,3), secondSegment(
    PROCRUSTES_SIZE,3), X0;
1569
1570     int location, rightIndex;
1571
1572     Eigen::Vector3f first_average, second_average, tempPoint;
1573
1574     /* A is SVD target, rotation is optimal rotation matrix, and secondPrime is P' after superimposition */
1575     Eigen::MatrixXf A, rotation, secondPrime = Eigen::MatrixXf(PROCRUSTES_SIZE,3);
1576
1577     float optimalScaling, traceA, pointDist;
1578
1579     float result = 0.0;
1580
1581     /* for all points, assign to them a point set with size of PROCRUSTES_SIZE neighboring points */
1582     for(int i=0;i<vertexChanged;++i)
1583     {
```

```
1584         rightIndex = i+PROCRUSTES_SIZE;
1585
1586         first_average = second_average = Eigen::VectorXf::Zero(3);
1587
1588         /* get the point set of neighboring 7 points and average */
1589         for(int j=i;j<rightIndex;++j)
1590         {
1591             location = j-i;
1592             for(int k=0;k<3;++k)
1593             {
1594                 firstSegment(location,k)=first(3*j+k);
1595                 secondSegment(location,k)=second(3*j+k);
1596             }
1597
1598             first_average+=firstSegment.row(location);
1599             second_average+=secondSegment.row(location);
1600         }
1601
1602         first_average/=PROCRUSTES_SIZE;
1603         second_average/=PROCRUSTES_SIZE;
1604
1605         /* reserve the matrix */
1606         X0 = firstSegment;
1607
1608         /* centralization for the point set */
1609         for(int j=0;j<PROCRUSTES_SIZE;++j)
1610         {
1611             firstSegment.row(j) = firstSegment.row(j)-first_average.transpose();
1612             secondSegment.row(j) = secondSegment.row(j)-second_average.transpose();
1613         }
1614
1615         /* get ssqX and ssqY */
1616         float ssqX = (firstSegment.cwiseProduct(firstSegment)).sum();
1617         float ssqY = (secondSegment.cwiseProduct(secondSegment)).sum();
1618
1619         /* check whether negative or not */
1620         assert(ssqX > 0 && ssqY > 0);
1621
1622         ssqX = sqrt(ssqX);
1623         ssqY = sqrt(ssqY);
1624
1625         /* scaling for the point set */
1626         firstSegment/=ssqX;
1627         secondSegment/=ssqY;
1628
1629         /* get the optimal rotational matrix by othogonal Procrutes analysis */
1630         A = firstSegment.transpose()*secondSegment;
1631
1632         /* perform SVD on A */
1633         JacobiSVD<MatrixXf> svd(A, ComputeThinU | ComputeThinV);
1634
1635         /* get the optimal 3D rotation */
1636         rotation = svd.matrixV()*(svd.matrixU().transpose());
1637
1638         /* get trace for singular value matrix */
1639         traceA = svd.singularValues().sum();
1640
1641         /* get optimal scaling */
1642         optimalScaling = traceA*ssqX/ssqY;
1643
1644         /* preset the average to the P' */
1645         for(int j=0;j<PROCRUSTES_SIZE;++j)
1646             secondPrime.row(j) = first_average;
1647
1648         /* get P' in superimposed space */
1649         secondPrime = ssqX*traceA*secondSegment*rotation+secondPrime;
1650
1651         /* compute the distance and store them in the std::vector<float> */
1652         pointDist = 0.0;
1653         for(int j=0;j<PROCRUSTES_SIZE;++j)
1654         {
1655             tempPoint = X0.row(j)-secondPrime.row(j);
1656             pointDist+= tempPoint.transpose()*tempPoint;
1657         }
1658
1659         /* get the average of P(x,y')^2 */
1660
1661         // either by computing the matrix
1662         //result+=pointDist;
1663
1664         // or directly using trace of the matrix
1665         float requiredD = 1.0-traceA*traceA;
1666         result+=requiredD*requiredD;
1667     }
1668     return result/vertexChanged;
1669 }
```

5.5.1.35 const float getProcrustesMetricSegment (const Eigen::VectorXf & *first*, const Eigen::VectorXf & *second*)

Definition at line 1681 of file Distance.cpp.

```
1683 {
1684     assert(first.size()==second.size());
1685
1686     const int& vertexCount = first.size()/3;
1687
1688     const int& vertexChanged = vertexCount/PROCRUSTES_SIZE;
1689     const int& newSize = 3*vertexChanged;
1690
1691     /* assign the segment list */
1692     Eigen::MatrixXf firstSegment(PROCRUSTES_SIZE,3), secondSegment(
      PROCRUSTES_SIZE,3), X0;
1693
1694     int location, rightIndex;
1695
1696     Eigen::Vector3f first_average, second_average, tempPoint;
1697
1698     /* A is SVD target, rotation is optimal rotation matrix, and secondPrime is P' after superimposition */
1699     Eigen::MatrixXf A, rotation, secondPrime = Eigen::MatrixXf(PROCRUSTES_SIZE,3);
1700
1701     float optimalScaling, traceA, pointDist;
1702
1703     float result = 0.0;
1704
1705     int effective = 0;
1706     /* for all points, assign to them a point set with size of PROCRUSTES_SIZE neighboring points */
1707     for(int i=0;i<vertexChanged;++i)
1708     {
1709         rightIndex = PROCRUSTES_SIZE*i+PROCRUSTES_SIZE;
1710
1711         first_average = second_average = Eigen::VectorXf::Zero(3);
1712
1713         /* get the point set of neighboring 7 points and average */
1714         for(int j=PROCRUSTES_SIZE*i;j<rightIndex;++j)
1715         {
1716             location = j-PROCRUSTES_SIZE*i;
1717             for(int k=0;k<3;++k)
1718             {
1719                 firstSegment(location,k)=first(3*j+k);
1720                 secondSegment(location,k)=second(3*j+k);
1721             }
1722
1723             first_average+=firstSegment.row(location);
1724             second_average+=secondSegment.row(location);
1725         }
1726
1727         first_average/=PROCRUSTES_SIZE;
1728         second_average/=PROCRUSTES_SIZE;
1729
1730         /* reserve the matrix */
1731         X0 = firstSegment;
1732
1733         /* centralization for the point set */
1734         for(int j=0;j<PROCRUSTES_SIZE;++j)
1735         {
1736             firstSegment.row(j) = firstSegment.row(j)-first_average.transpose();
1737             secondSegment.row(j) = secondSegment.row(j)-second_average.transpose();
1738         }
1739
1740         /* get ssqX and ssqY */
1741         float ssqX = (firstSegment.cwiseProduct(firstSegment)).sum();
1742         float ssqY = (secondSegment.cwiseProduct(secondSegment)).sum();
1743
1744         /* check whether negative or not */
1745         assert(ssqX > 0 && ssqY > 0);
1746
1747         if(ssqX<1.0e-14 || ssqY<1.0e-14)
1748             continue;
1749
1750         ssqX = sqrt(ssqX);
1751         ssqY = sqrt(ssqY);
1752
1753         if(ssqX<1.0e-8 || ssqY<1.0e-8)
1754             continue;
1755
1756         /* scaling for the point set */
1757         firstSegment/=ssqX;
1758         secondSegment/=ssqY;
1759
1760         /* get the optimal rotational matrix by othogonal Procrutes analysis */
1761         A = firstSegment.transpose()*secondSegment;
```

```
1762
1763         /* perform SVD on A */
1764         JacobiSVD<MatrixXf> svd(A, ComputeThinU | ComputeThinV);
1765
1766         /* get the optimal 3D rotation */
1767         rotation = svd.matrixV()*(svd.matrixU().transpose());
1768
1769         /* get trace for singular value matrix */
1770         traceA = svd.singularValues().sum();
1771
1772         /* get optimal scaling */
1773         optimalScaling = traceA*ssqX/ssqY;
1774
1775         /* preset the average to the P' */
1776         for(int j=0;j<PROCRUSTES_SIZE;++j)
1777             secondPrime.row(j) = first_average;
1778
1779         /* get P' in superimposed space */
1780         secondPrime = ssqX*traceA*secondSegment*rotation+secondPrime;
1781
1782         /* compute the distance and store them in the std::vector<float> */
1783         pointDist = 0.0;
1784         for(int j=0;j<PROCRUSTES_SIZE;++j)
1785         {
1786             tempPoint = X0.row(j)-secondPrime.row(j);
1787             pointDist+= tempPoint.transpose()*tempPoint;
1788         }
1789         /* get the average of P(x,y')^2 */
1790         result+=pointDist;
1791         ++effective;
1792     }
1793
1794     if(effective==0)
1795     {
1796         return 1.0e-8;
1797     }
1798     else
1799         return result/effective;
1800 }
```

5.5.1.36 const float getRotation (const std::vector< vector< float > > & *streamline*, std::vector< float > & *rotation*)

Definition at line 1406 of file Distance.cpp.

```
1407 {
1408     if(streamline.empty())
1409         return -1;
1410     float result = 0, eachSum;
1411     const int& size = streamline.size();
1412     rotation = std::vector<float>(size);
1413     std::vector<float> eachLine;
1414     Eigen::Vector3f first, second;
1415     int lineSize;
1416     for(int i=0;i<size;++i)
1417     {
1418         eachSum = 0;
1419         eachLine = streamline[i];
1420         lineSize = eachLine.size()/3-2;
1421         // calculate the summation of discrete curvatures
1422         for(int j=0;j<lineSize;++j)
1423         {
1424             first<<eachLine[3*j+3]-eachLine[3*j],eachLine[3*j+4]-eachLine[3*j+1],eachLine[3*j+5]-eachLine[3
    *j+2];
1425             second<<eachLine[3*j+6]-eachLine[3*j+3],eachLine[3*j+7]-eachLine[3*j+4],eachLine[3*j+8]-
    eachLine[3*j+5];
1426
1427             float firstNorm = first.norm(), secondNorm = second.norm();
1428             if(firstNorm>=1.0e-8 && secondNorm>=1.0e-8)
1429             {
1430                 float angle = first.dot(second)/firstNorm/secondNorm;
1431                 angle = std::max(angle,float(-1.0));
1432                 angle = std::min(angle,float(1.0));
1433                 eachSum+=acos(angle);
1434             }
1435         }
1436         // get the mean of discrete curvatures
1437         rotation[i]=eachSum;
1438         result+=eachSum;
1439     }
1440     result/=size;
1441     return result;
1442 }
```

5.5.1.37 const float getSignatureMetric (const Eigen::VectorXf & *firstArray,* const Eigen::VectorXf & *secondArray,* const std::vector< float > & *firstHist,* const std::vector< float > & *secondHist*)

Definition at line 1454 of file Distance.cpp.

```
1458 {
1459     /* would choose alpha = 0.5, and 10% of subset vertices for mean_dist */
1460     const float& Alpha = 0.5;
1461     const int& SUBSET = 10;
1462
1463     /* assert whether the size is the same */
1464     assert(firstArray.size()==secondArray.size());
1465     assert(firstHist.size()==secondHist.size());
1466
1467     const int& histSize = firstHist.size();
1468     const int& vertexCount = firstArray.size()/3;
1469     const int& size = vertexCount/SUBSET+1;
1470
1471     Eigen::VectorXf firstSubset(3*size), secondSubset(3*size);
1472
1473     /* get mean_dist between two sampled subsets */
1474     int tempPos = 0;
1475     for(int i=0;i<vertexCount;i+=SUBSET)
1476     {
1477         for(int j=0;j<3;++j)
1478         {
1479             firstSubset(3*tempPos+j)=firstArray(3*i+j);
1480             secondSubset(3*tempPos+j)=secondArray(3*i+j);
1481         }
1482         ++tempPos;
1483     }
1484
1485     /* get mean_dist */
1486     float result = getMetric_MOP(firstSubset, secondSubset);
1487
1488     float chi_test = 0.0, histDiff, histSum;
1489
1490     /* get chi_test for two histograms */
1491     for(int i=0;i<histSize;++i)
1492     {
1493         histDiff = firstHist[i]-secondHist[i];
1494         histSum = firstHist[i]+secondHist[i];
1495         /* check numerical error */
1496         if(histSum<1.0e-8)
1497             continue;
1498
1499         chi_test+= histDiff*histDiff/histSum;
1500     }
1501
1502     /* get combined distance */
1503     result = (1-Alpha)*chi_test + Alpha*result;
1504
1505     return result;
1506 }
```

5.5.1.38 const float getSignatureMetric (const Eigen::VectorXf & *centroid,* const Eigen::VectorXf & *first,* const std::vector< float > & *firstHist*)

Definition at line 1517 of file Distance.cpp.

```
1520 {
1521     std::vector<float> centroidHist;
1522     /* get the bin-based histogram for signature */
1523     getSignatureHist(centroid, BIN_SIZE, centroidHist);
1524
1525     return getSignatureMetric(centroid,first,centroidHist,firstHist);
1526 }
```

5.5.1.39 const float getSignatureMetric (const Eigen::VectorXf & *first,* const Eigen::VectorXf & *second*)

Definition at line 1536 of file Distance.cpp.

```
1538 {
1539     std::vector<float> firstHist, secondHist;
1540     /* get the bin-based histogram for signature */
1541     getSignatureHist(first, BIN_SIZE, firstHist);
1542     getSignatureHist(second, BIN_SIZE, secondHist);
1543
1544     return getSignatureMetric(first,second,firstHist,secondHist);
1545 }
```

5.5.2 Variable Documentation

5.5.2.1 float** distanceMatrix

Definition at line 39 of file Distance.cpp.

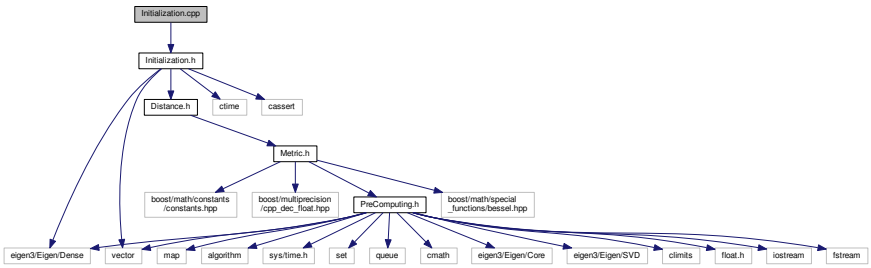
5.5.2.2 const int& PROCRUSTES_SIZE

Definition at line 33 of file Distance.cpp.

5.6 Initialization.cpp File Reference

```
#include "Initialization.h"
```

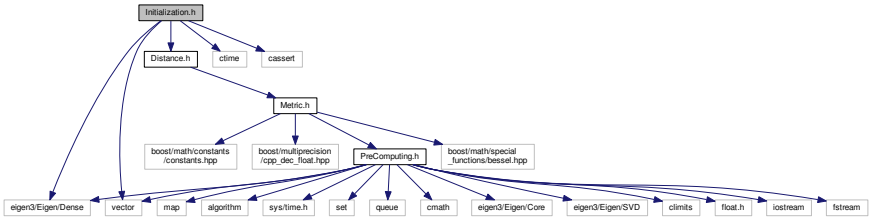
Include dependency graph for Initialization.cpp:



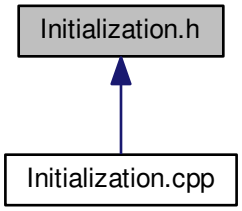
5.7 Initialization.h File Reference

```
#include <eigen3/Eigen/Dense>
#include <vector>
#include <ctime>
#include <cassert>
#include "Distance.h"
```

Include dependency graph for Initialization.h:



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



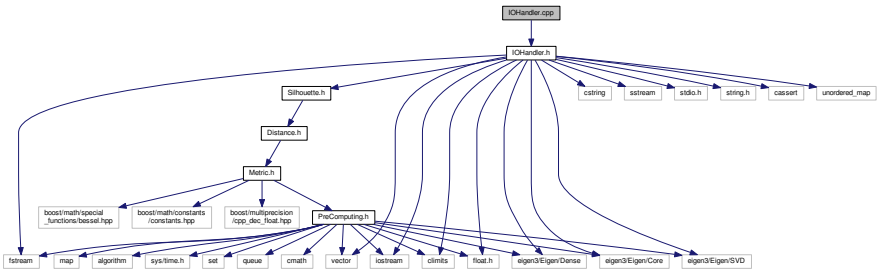
Classes

- class Initialization

5.8 IOHandler.cpp File Reference

```
#include "IOHandler.h"
```

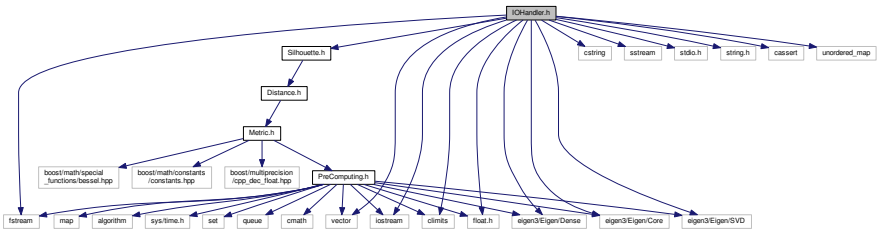
Include dependency graph for IOHandler.cpp:



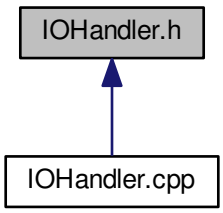
5.9 IOHandler.h File Reference

```
#include <fstream>
#include <vector>
#include <iostream>
#include <cstring>
#include <sstream>
#include <stdio.h>
#include <string.h>
#include <climits>
#include <cassert>
#include <float.h>
#include <unordered_map>
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/SVD>
#include "Silhouette.h"
```

Include dependency graph for IOHandler.h:



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



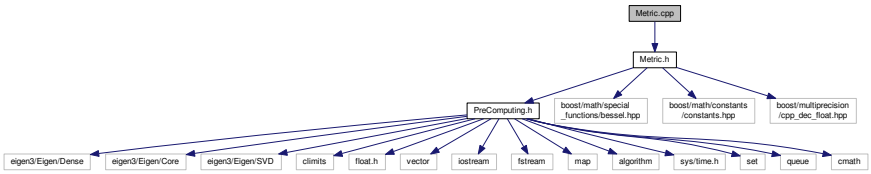
Classes

- struct ExtractedLine
- struct MeanLine
- struct StringQuery
- struct FeatureLine
- class IOHandler

5.10 Metric.cpp File Reference

```
#include "Metric.h"
```

Include dependency graph for Metric.cpp:



Functions

- void computeMeanRotation (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< float > &rotation)
- void getRotationSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &rotationSequence)
- void getNormalSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MultiVariate > &normalMultivariate)
- void getFixedSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &rotationSequence)
- void getUnnormalizedSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MultiVariate > &normalMultivariate)
- void getUnitDirection (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< VectorXf > &unitLength)
- void computePairWise (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise, std::vector< std::vector< float > > &pairwiseNorm)
- void getSignatureBin (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise)
- void getBundleEntropy (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise)
- const float get_calculus (const float &a, const float &b, const float &c)

Variables

- const int & BIN_SIZE = 20
- const int & BUNDLE_SIZE = 20

5.10.1 Function Documentation

5.10.1.1 void computeMeanRotation (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< float > & *rotation*)

Definition at line 30 of file Metric.cpp.

```
34 {
35     rotation = std::vector<float>(Row, 0.0);
36     const int& pointNum = Column/3-2;
37 #pragma omp parallel for schedule(static) num_threads(8)
38     for (int i = 0; i < Row; ++i)
39     {
40         rotation[i] = getRotation(data.row(i), pointNum);
41     }
42 }
```

5.10.1.2 void computePairWise (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*, std::vector< std::vector< float > > & *pairwiseNorm*)

Definition at line 164 of file Metric.cpp.

```
169 {
170     const int& pointNum = Column/3; // how many line segments
171 #pragma omp parallel for schedule(static) num_threads(8)
172     for (int i = 0; i < Row; ++i)
173     {
174         getPairWise_byEach(data.row(i), pointNum, pairwise[i], pairwiseNorm[i]);
175     }
176 }
```

5.10.1.3 const float get_calculus (const float & *a*, const float & *b*, const float & *c*)

Definition at line 230 of file Metric.cpp.

```
231 {
232     typedef boost::multiprecision::cpp_dec_float_50 mp_type;
233     float result = integral(0.0F, 1.0F, 0.00001F,
      cyl_bessel_j_integral_rep<float>(a,b,c));
234     return result;
235 }
```

5.10.1.4 void getBundleEntropy (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*)

Definition at line 209 of file Metric.cpp.

```
213 {
214 #pragma omp parallel for schedule(static) num_threads(8)
215     for (int i=0;i<Row;++i)
216     {
217         getLinearAngularEntropy(data.row(i),BUNDLE_SIZE,pairwise[i]);
218     }
219 }
```

5.10.1.5 void getFixedSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 97 of file Metric.cpp.

```
101 {
102     const int& pointNum = Column/3-1;
103 #pragma omp parallel for schedule(static) num_threads(8)
104     for (int i = 0; i < Row; ++i)
105     {
106         getEachFixedSequence(data.row(i), pointNum, rotationSequence[i]);
107     }
108 }
```

5.10.1.6 void getNormalSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 75 of file Metric.cpp.

```
79 {
80     const int& pointNum = Column/3-1;
81 #pragma omp parallel for schedule(static) num_threads(8)
82     for (int i = 0; i < Row; ++i)
83     {
84         getNormalMultivariate(data.row(i), pointNum, normalMultivariate[i]);
85     }
86 }
```

5.10.1.7 void getRotationSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 53 of file Metric.cpp.

```
57 {
58     const int& pointNum = Column/3-2;
59 #pragma omp parallel for schedule(static) num_threads(8)
60     for (int i = 0; i < Row; ++i)
61     {
62         getSequence(data.row(i), pointNum, rotationSequence[i]);
63     }
64 }
```

5.10.1.8 void getSignatureBin (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*)

Definition at line 187 of file Metric.cpp.

```
191 {
192 #pragma omp parallel for schedule(static) num_threads(8)
193     for (int i=0;i<Row;++i)
194     {
195         //getSignatureHist(data.row(i),BIN_SIZE,pairwise[i]);
196         getSignatureHistSampled(data.row(i),BIN_SIZE,pairwise[i]);
197     }
198 }
```

5.10.1.9 void getUnitDirection (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< VectorXf > & *unitLength*)

Definition at line 141 of file Metric.cpp.

```
145 {
146     const int& pointNum = Column/3;
147 #pragma omp parallel for schedule(static) num_threads(8)
148     for (int i = 0; i < Row; ++i)
149     {
150         getUnitDirection_byEach(data.row(i), pointNum, unitLength[i]);
151     }
152 }
```

5.10.1.10 void getUnnormalizedSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 119 of file Metric.cpp.

```
123 {
124     const int& pointNum = Column/3-1;
125 #pragma omp parallel for schedule(static) num_threads(8)
126     for (int i = 0; i < Row; ++i)
127     {
128         getUnnormalizedMultivariate(data.row(i), pointNum, normalMultivariate[i]
      );
129     }
130 }
```

5.10.2 Variable Documentation

5.10.2.1 const int& BIN_SIZE = 20

Definition at line 13 of file Metric.cpp.

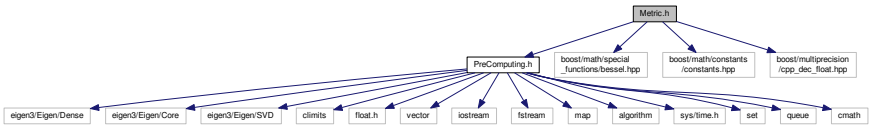
5.10.2.2 const int& BUNDLE_SIZE = 20

Definition at line 19 of file Metric.cpp.

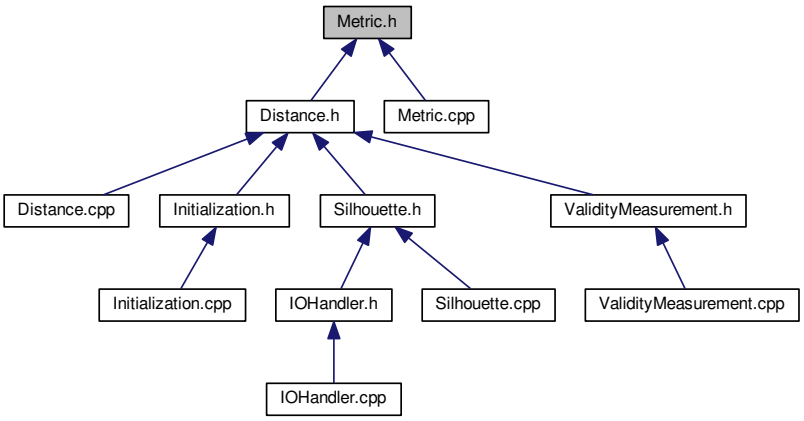
5.11 Metric.h File Reference

```
#include "PreComputing.h"
#include <boost/math/special_functions/bessel.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/multiprecision/cpp_dec_float.hpp>
```

Include dependency graph for Metric.h:



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



Classes

- struct MetricPreparation
- class cyl_bessel_j_integral_rep< value_type >

Functions

- void computeMeanRotation (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< float > &rotation)
- void getRotationSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &rotationSequence)
- void getNormalSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MultiVariate > &normalMultivariate)
- void getFixedSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &rotationSequence)
- void getUnnormalizedSequence (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MultiVariate > &normalMultivariate)
- void getUnitDirection (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< VectorXf > &unitLength)
- void computePairWise (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise, std::vector< std::vector< float > > &pairwiseNorm)
- void getSignatureBin (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise)
- void getBundleEntropy (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< std::vector< float > > &pairwise)
- template<typename value_type , typename function_type >
  value_type integral (const value_type a, const value_type b, const value_type tol, function_type func)
- const float get_calculus (const float &a, const float &b, const float &c)

Variables

- const int & BIN_SIZE
- const int & BUNDLE_SIZE

5.11.1 Function Documentation

5.11.1.1 void computeMeanRotation (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< float > & *rotation*)

Definition at line 30 of file Metric.cpp.

```
34 {
35     rotation = std::vector<float>(Row, 0.0);
36     const int& pointNum = Column/3-2;
37 #pragma omp parallel for schedule(static) num_threads(8)
38     for (int i = 0; i < Row; ++i)
39     {
40         rotation[i] = getRotation(data.row(i), pointNum);
41     }
42 }
```

5.11.1.2 void computePairWise (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*, std::vector< std::vector< float > > & *pairwiseNorm*)

Definition at line 164 of file Metric.cpp.

```
169 {
170     const int& pointNum = Column/3; // how many line segments
171 #pragma omp parallel for schedule(static) num_threads(8)
172     for (int i = 0; i < Row; ++i)
173     {
174         getPairWise_byEach(data.row(i), pointNum, pairwise[i], pairwiseNorm[i]);
175     }
176 }
```

5.11.1.3 const float get_calculus (const float & *a*, const float & *b*, const float & *c*)

Definition at line 230 of file Metric.cpp.

```
231 {
232     typedef boost::multiprecision::cpp_dec_float_50 mp_type;
233     float result = integral(0.0F, 1.0F, 0.00001F,
      cyl_bessel_j_integral_rep<float>(a,b,c));
234     return result;
235 }
```

5.11.1.4 void getBundleEntropy (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*)

Definition at line 209 of file Metric.cpp.

```
213 {
214 #pragma omp parallel for schedule(static) num_threads(8)
215     for (int i=0;i<Row;++i)
216     {
217         getLinearAngularEntropy(data.row(i),BUNDLE_SIZE,pairwise[i]);
218     }
219 }
```

5.11.1.5 void getFixedSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 97 of file Metric.cpp.

```
101 {
102     const int& pointNum = Column/3-1;
103 #pragma omp parallel for schedule(static) num_threads(8)
104     for (int i = 0; i < Row; ++i)
105     {
106         getEachFixedSequence(data.row(i), pointNum, rotationSequence[i]);
107     }
108 }
```

5.11.1.6 void getNormalSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 75 of file Metric.cpp.

```
79 {
80     const int& pointNum = Column/3-1;
81 #pragma omp parallel for schedule(static) num_threads(8)
82     for (int i = 0; i < Row; ++i)
83     {
84         getNormalMultivariate(data.row(i), pointNum, normalMultivariate[i]);
85     }
86 }
```

5.11.1.7 void getRotationSequence (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *rotationSequence*)

Definition at line 53 of file Metric.cpp.

```
57 {
58     const int& pointNum = Column/3-2;
59 #pragma omp parallel for schedule(static) num_threads(8)
60     for (int i = 0; i < Row; ++i)
61     {
62         getSequence(data.row(i), pointNum, rotationSequence[i]);
63     }
64 }
```

5.11.1.8 void getSignatureBin (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< std::vector< float > > & *pairwise*)

Definition at line 187 of file Metric.cpp.

```
191 {
192 #pragma omp parallel for schedule(static) num_threads(8)
193     for (int i=0;i<Row;++i)
194     {
195         //getSignatureHist(data.row(i),BIN_SIZE,pairwise[i]);
196         getSignatureHistSampled(data.row(i),BIN_SIZE,pairwise[i]);
197     }
198 }
```

5.11.1.9 void getUnitDirection (const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* std::vector< VectorXf > & *unitLength*)

Definition at line 141 of file Metric.cpp.

```
145 {
146     const int& pointNum = Column/3;
147 #pragma omp parallel for schedule(static) num_threads(8)
148     for (int i = 0; i < Row; ++i)
149     {
150         getUnitDirection_byEach(data.row(i), pointNum, unitLength[i]);
151     }
152 }
```

5.11.1.10 void getUnnormalizedSequence (const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* std::vector< MultiVariate > & *normalMultivariate*)

Definition at line 119 of file Metric.cpp.

```
123 {
124     const int& pointNum = Column/3-1;
125 #pragma omp parallel for schedule(static) num_threads(8)
126     for (int i = 0; i < Row; ++i)
127     {
128         getUnnormalizedMultivariate(data.row(i), pointNum, normalMultivariate[i]
      );
129     }
130 }
```

5.11.1.11 template<typename value_type , typename function_type > value_type integral (const value_type *a,* const value_type *b,* const value_type *tol,* function_type *func*) [inline]

Definition at line 305 of file Metric.h.

```
309 {
310     unsigned n = 1U;
311
312     value_type h = (b - a);
313     value_type I = (func(a) + func(b)) * (h / 2);
314
315     for(unsigned k = 0U; k < 8U; k++)
316     {
317         h /= 2;
318
319         value_type sum(0);
320         for(unsigned j = 1U; j <= n; j++)
321         {
322             sum += func(a + (value_type((j * 2) - 1) * h));
323         }
324
325         const value_type I0 = I;
326         I = (I / 2) + (h * sum);
327
328         const value_type ratio     = I0 / I;
329         const value_type delta     = ratio - 1;
330         const value_type delta_abs = ((delta < 0) ? -delta : delta);
331
332         if((k > 1U) && (delta_abs < tol))
333         {
334             break;
335         }
336
337         n *= 2U;
338     }
339
340     return I;
341 }
```

5.11.2 Variable Documentation

5.11.2.1 const int& BIN_SIZE

Definition at line 13 of file Metric.cpp.

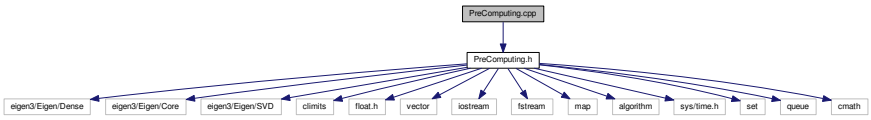
5.11.2.2 const int& BUNDLE_SIZE

Definition at line 19 of file Metric.cpp.

5.12 PreComputing.cpp File Reference

```
#include "PreComputing.h"
```

Include dependency graph for PreComputing.cpp:



Functions

- void getSequence (const VectorXf &array, const int &size, std::vector< float > &rowSequence)
- const float getRotation (const VectorXf &array, const int &size)
- void getNormalMultivariate (const VectorXf &array, const int &size, MultiVariate &rowSequence)
- void getEachFixedSequence (const VectorXf &array, const int &size, std::vector< float > &rowSequence)
- void getUnnormalizedMultivariate (const VectorXf &array, const int &size, MultiVariate &rowSequence)
- void getUnitDirection_byEach (const VectorXf &array, const int &pointNum, VectorXf &direction)
- void getPairWise_byEach (const VectorXf &data, const int &size, std::vector< float > &wiseVec, std::vector< float > &wiseNorm)
- void getSignatureHist (const Eigen::VectorXf &array, const int &binNum, std::vector< float > &histogram)
- void getSignatureHistSampled (const Eigen::VectorXf &array, const int &binNum, std::vector< float > &histogram)
- void getLinearAngularEntropy (const Eigen::VectorXf &array, const int &bundleSize, std::vector< float > &histogram)

5.12.1 Function Documentation

5.12.1.1 void getEachFixedSequence (const VectorXf & *array,* const int & *size,* std::vector< float > & *rowSequence*)

Definition at line 135 of file PreComputing.cpp.

```
138 {
139     rowSequence = std::vector<float>(2);
140     float dotValue, leftNorm, meanRotation = 0.0, deviation = 0.0, angle, result;
141     Vector3f left, xRay;
142     xRay << 1.0,0.0,0.0;
143     for (int j = 0; j < size; ++j)
144     {
145         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
146         dotValue = left.dot(xRay);
147         leftNorm = left.norm();
148         if(leftNorm >= 1.0e-8)
149         {
150             result = dotValue/leftNorm;
151             result = min(1.0,(double)result);
152             result = max(-1.0,(double)result);
153             angle = acos(result);
154             meanRotation += angle;
155             deviation += angle*angle;
156         }
157         else
158         {
159             angle = M_PI;
160             meanRotation += angle;
161             deviation += angle*angle;
162         }
163     }
164     meanRotation /= size;
165     rowSequence[0] = meanRotation;
166     int stdDevia = deviation/size-(meanRotation*meanRotation);
167     if(stdDevia<0)
168         stdDevia = 1.0e-8;
169     rowSequence[1] = sqrt(stdDevia);
170 }
```

5.12.1.2 void getLinearAngularEntropy (const Eigen::VectorXf & *array*, const int & *bundleSize*, std::vector< float > & *histogram*)

Definition at line 483 of file PreComputing.cpp.

```
486 {
487     /* if empty vector, should allocate memory ahead of time */
488     if(histogram.empty())
489         histogram = std::vector<float>(2);
490
491     /* get how many vertices you'll have */
492     const int& segmentNum = array.size()/3-1;
493
494     const int& curvatureNum = segmentNum-1;
495
496     /* should partition the whole streamlines into bunleSize segments, and compute the entropy */
497
498     std::vector<float> segmentVec(segmentNum), curvatureVec(curvatureNum);
499
500     Eigen::Vector3f firstSeg, secondSeg;
501
502     /* discrete curvature */
503     float curva;
504
505     float lengthSum = 0.0, curveSum = 0.0;
506     int vecIndex = 0;
507     for(int i=0;i<curvatureNum;++i)
508     {
509         for(int j=0;j<3;++j)
510         {
511             firstSeg(j)=array(3*i+3+j)-array(3*i+j);
512             secondSeg(j)=array(3*i+6+j)-array(3*i+3+j);
513         }
514
515         if(firstSeg.norm()<1.0e-6 || secondSeg.norm()<1.0e-6)
516         {
517             segmentVec[i] = 0.0;
518             curvatureVec[i]= 0.0;
519             continue;
520         }
521
522         float firstSegNorm = firstSeg.norm(), secondSegNorm = secondSeg.norm();
523         if(firstSegNorm<1.0e-8 || secondSegNorm<1.0e-8)
524             curva = 0.0;
525         else
```

```
526         {
527             curva = firstSeg.dot(secondSeg)/firstSegNorm/secondSegNorm;
528 
529             /* clip curvature into range [-1.0, 1.0] */
530             curva = std::min(float(1.0), curva);
531             curva = std::max(float(-1.0), curva);
532 
533             curva = acos(curva);
534         }
535 
536         /* store in the vector */
537         curvatureVec[i]=curva;
538         curveSum+=curva;
539 
540         /* store path */
541         segmentVec[i] = firstSeg.norm();
542         lengthSum+=segmentVec[i];
543     }
544 
545     int i = curvatureNum;
546     for(int j=0;j<3;++j)
547     {
548         firstSeg(j)=array(3*i+3+j)-array(3*i+j);
549     }
550     segmentVec[i] = firstSeg.norm();
551     lengthSum+=segmentVec[i];
552 
553 
554     /* should deal with exceptional case if lengthSum == 0 or curveSum == 0 */
555     if(lengthSum<1.0e-6)
556     {
557         histogram[0] = 1.0;
558     }
559     else
560     {
561         /* get ratio for the vec */
562         const int& segmentQuotient = segmentNum/bundleSize;
563         const int& segmentResidue = segmentNum%bundleSize;
564 
565         /* get the vec for bundleSize */
566         std::vector<float> lengthVec(bundleSize);
567 
568         float tempLength, linearEntropy = 0.0, prob;
569         int left, right;
570         for(int k = 0;k<bundleSize-1;++k)
571         {
572             tempLength = 0.0;
573             left = k*segmentQuotient, right = (k+1)*segmentQuotient;
574             for(int i = left;i<right;++i)
575                 tempLength+=segmentVec[i];
576 
577             prob = tempLength/lengthSum;
578 
579             if(prob>1.0e-6)
580                 linearEntropy += prob*log2f(prob);
581         }
582 
583         left = (bundleSize-1)*segmentQuotient, right = segmentNum;
584         tempLength = 0.0;
585         for(int i=left;i<right;++i)
586         {
587             tempLength+=segmentVec[i];
588         }
589         if(prob>1.0e-6)
590             prob = tempLength/lengthSum;
591         linearEntropy += prob*log2f(prob);
592 
593         linearEntropy = -linearEntropy/log2f(float(bundleSize));
594         histogram[0] = linearEntropy;
595     }
596 
597     /* deal with curveSum == 1.0 */
598     if(curveSum<1.0e-6)
599     {
600         histogram[1] = 1.0;
601     }
602     else
603     {
604         const int& curvatureQuotient = curvatureNum/bundleSize;
605         const int& curvatureResidue = curvatureNum%bundleSize;
606 
607         /* get the vec for bundleSize */
608         std::vector<float> curveVec(bundleSize);
609 
610         float tempCurve, angularEntropy = 0.0, prob;
611         int left, right;
612         for(int k = 0;k<bundleSize-1;++k)
```

```
613         {
614             tempCurve = 0.0;
615             left = k*curvatureQuotient, right = (k+1)*curvatureQuotient;
616             for(int i=left;i<right;++i)
617                 tempCurve+=curvatureVec[i];
618 
619             prob = tempCurve/curveSum;
620             if(prob>1.0e-6)
621                 angularEntropy += prob*log2f(prob);
622         }
623 
624         left = (bundleSize-1)*curvatureQuotient, right = curvatureNum;
625         tempCurve = 0.0;
626         for(int i=left;i<right;++i)
627         {
628             tempCurve+=curvatureVec[i];
629         }
630         prob = tempCurve/curveSum;
631         if(prob>1.0e-6)
632             angularEntropy += prob*log2f(prob);
633 
634         angularEntropy = -angularEntropy/log2f(float(bundleSize));
635         histogram[1] = angularEntropy;
636     }
637 }
```

5.12.1.3 void getNormalMultivariate (const VectorXf & *array*, const int & *size*, MultiVariate & *rowSequence*)

Definition at line 89 of file PreComputing.cpp.

```
92 {
93     MatrixXf normalDirection(size,3);
94     float leftNorm;
95     Vector3f left;
96     VectorXf unitOne(size);
97     for (int j = 0; j < size; ++j)
98     {
99         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
100         leftNorm = left.norm();
101         if(leftNorm >= 1.0e-8)
102         {
103             for(int k=0;k<3;k++)
104                 /* record each line segment normal direction */
105                 normalDirection(j,k) = left(k)/leftNorm;
106         }
107         else
108         {
109             for(int k=0;k<3;k++)
110                 /* if norm is small, mark them as zero to tell identical points */
111                 normalDirection(j,k) = 0.0;
112         }
113         unitOne(j) = 1.0;
114     }
115 
116     VectorXf meanNormal(3);
117     for (int i = 0; i < 3; ++i)
118     {
119         meanNormal(i) = normalDirection.transpose().row(i).mean();
120     }
121 
122     MatrixXf tempMatrix = normalDirection-unitOne*meanNormal.transpose();
123     rowSequence.covariance = tempMatrix.transpose()*tempMatrix/(size-1);
124     rowSequence.meanVec = meanNormal;
125 }
```

5.12.1.4 void getPairWise_byEach (const VectorXf & *data*, const int & *size*, std::vector< float > & *wiseVec*, std::vector< float > & *wiseNorm*)

Definition at line 263 of file PreComputing.cpp.

```
267 {
268     if(wiseVec.empty())
269         wiseVec = std::vector<float>(3*size);
270 
271     if(wiseNorm.empty())
272         wiseNorm = std::vector<float>(size);
273 
274     for (int i = 0; i < size; ++i)
275     {
276         float leftNorm;
277         Vector3f left;
278         left << data(3*i+3)-data(3*i),data(3*i+4)-data(3*i+1),data(3*i+5)-data(3*i+2);
279         leftNorm = left.norm();
280         if(leftNorm >= 1.0e-8)
281         {
282             for (int j = 0; j < 3; ++j)
283             {
284                 wiseVec[3*i+j] = left(j)/leftNorm;
285             }
286             wiseNorm[i] = leftNorm;
287         }
288         else
289         {
290             for (int j = 0; j < 3; ++j)
291             {
292                 wiseVec[3*i+j] = 0.0;
293             }
294             wiseNorm[i] = 0.0;
295         }
296     }
297 }
```

5.12.1.5 const float getRotation (const VectorXf & *array*, const int & *size*)

Definition at line 57 of file PreComputing.cpp.

```
59 {
60     float dotValue, leftNorm, rightNorm, meanRotation = 0.0, result;
61     Vector3f left, right;
62     for (int j = 0; j < size; ++j)
63     {
64         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
65         right << array(j*3+6)-array(j*3+3), array(j*3+7)-array(j*3+4), array(j*3+8)-array(j*3+5);
66         dotValue = left.dot(right);
67         leftNorm = left.norm();
68         rightNorm = right.norm();
69         if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
70         {
71             result = dotValue/leftNorm/rightNorm;
72             result = min(1.0,(double)result);
73             result = max(-1.0,(double)result);
74             meanRotation += acos(result);
75         }
76     }
77     meanRotation/=size;
78     return meanRotation;
79 }
```

5.12.1.6 void getSequence (const VectorXf & *array*, const int & *size*, std::vector< float > & *rowSequence*)

Definition at line 17 of file PreComputing.cpp.

```
20 {
21     rowSequence = std::vector<float>(2);
22     float dotValue, leftNorm, rightNorm, meanRotation = 0.0, deviation = 0.0, angle, result;
23     Vector3f left, right;
24     for (int j = 0; j < size; ++j)
25     {
26         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
27         right << array(j*3+6)-array(j*3+3), array(j*3+7)-array(j*3+4), array(j*3+8)-array(j*3+5);
28         dotValue = left.dot(right);
29         leftNorm = left.norm();
```

```
30          rightNorm = right.norm();
31          if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
32          {
33              result = dotValue/leftNorm/rightNorm;
34              result = min(1.0,(double)result);
35              result = max(-1.0,(double)result);
36              angle = acos(result);
37              meanRotation += angle;
38              deviation += angle*angle;
39          }
40      }
41      meanRotation /= size;
42      rowSequence[0] = meanRotation;
43      int stdDevia = deviation/size-(meanRotation*meanRotation);
44      if(stdDevia<0)
45          stdDevia = 1.0e-8;
46      rowSequence[1] = sqrt(stdDevia);
47  }
```

5.12.1.7 void getSignatureHist (const Eigen::VectorXf & *array*, const int & *binNum*, std::vector< float > & *histogram*)

Definition at line 307 of file PreComputing.cpp.

```
310 {
311     /* if empty vector, should allocate memory ahead of time */
312     if(histogram.empty())
313         histogram = std::vector<float>(binNum);
314
315     /* get how many vertices you'll have */
316     const int& segmentNum = array.size()/3-1;
317
318     /* how many vertices on each bin on average */
319     const int& binSize = segmentNum/binNum;
320
321     /* first several has binSize+1 vertices, while the rest have binSize vertices */
322     const int& residueNum = segmentNum%binNum;
323
324     if(binSize<1)
325     {
326         std::cout << "Error for bin size calculation!" << std::endl;
327         exit(1);
328     }
329
330     int totalVertexOnBin = binSize+1, index = 0;
331     float dotValue, leftNorm, rightNorm, meanRotation = 0.0, result, rotationSum;
332     Vector3f left, right;
333     for (int i = 0; i < binNum; ++i)
334     {
335         /* would reduce that to binSize if i>=residueNum */
336         if(i==residueNum)
337             totalVertexOnBin = binSize;
338
339         /* reset the rotationSum */
340         rotationSum = 0.0;
341         for(int j=0;j<totalVertexOnBin;++j)
342         {
343             left << array(index*3+3)-array(index*3),
344                     array(index*3+4)-array(index*3+1),
345                     array(index*3+5)-array(index*3+2);
346             right << array(index*3+6)-array(index*3+3),
347                      array(index*3+7)-array(index*3+4),
348                      array(index*3+8)-array(index*3+5);
349             dotValue = left.dot(right);
350             leftNorm = left.norm();
351             rightNorm = right.norm();
352             if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
353             {
354                 result = dotValue/leftNorm/rightNorm;
355                 result = min(1.0,(double)result);
356                 result = max(-1.0,(double)result);
357                 rotationSum += acos(result);
358             }
359             ++index;
360         }
361
362         histogram[i] = rotationSum;
363     }
364     assert(index==segmentNum);
365 }
```

5.12.1.8 void getSignatureHistSampled (const Eigen::VectorXf & *array*, const int & *binNum*, std::vector< float > & *histogram*)

Definition at line 375 of file PreComputing.cpp.

```
378 {
379     /* if empty vector, should allocate memory ahead of time */
380     if(histogram.empty())
381         histogram = std::vector<float>(binNum);
382 
383     /* get how many vertices you'll have */
384     const int& segmentNum = array.size()/3-2;
385 
386     /* preset a priority_queue to get the sampled points in maximal curvatures */
387     priority_queue<CurvatureObject, std::vector<CurvatureObject>, CompareFunc> pQueue;
388 
389     std::vector<float> curvatureVec(segmentNum);
390 
391     Eigen::Vector3f firstSeg, secondSeg;
392 
393     /* discrete curvature */
394     float curva;
395 
396     int vecIndex = 0;
397     for(int i=0;i<segmentNum;++i)
398     {
399         for(int j=0;j<3;++j)
400         {
401             firstSeg(j)=array(3*i+3+j)-array(3*i+j);
402             secondSeg(j)=array(3*i+6+j)-array(3*i+3+j);
403         }
404 
405         float firstSegNorm = firstSeg.norm(), secondSegNorm = secondSeg.norm();
406         if(firstSegNorm<1.0e-8 || secondSegNorm<1.0e-8)
407             curva = 0.0;
408         else
409         {
410             curva = firstSeg.dot(secondSeg)/firstSegNorm/secondSegNorm;
411 
412             /* clip curvature into range [-1.0, 1.0] */
413             curva = std::min(float(1.0), curva);
414             curva = std::max(float(-1.0), curva);
415 
416             curva = acos(curva);
417         }
418         /* store in the vector */
419         curvatureVec[vecIndex++]=curva;
420 
421         /* push it into the priority queue */
422         pQueue.push(CurvatureObject(curva, i));
423 
424     }
425 
426     /* get the first binNum-1 object */
427     CurvatureObject top;
428 
429     /* use ordered_set to sort the index */
430     std::vector<int> indexVec;
431 
432     int indexNum = 0;
433     const int& requiredNum = binNum-1;
434     while(indexNum<requiredNum && !pQueue.empty())
435     {
436         top = pQueue.top();
437         indexVec.push_back(top.index);
438         pQueue.pop();
439         ++indexNum;
440     }
441 
442     assert(indexVec.size()==requiredNum);
443 
444     /* sort the vec */
445     std::sort(indexVec.begin(), indexVec.end());
446 
447     /* start sampling to make a curvature histogram */
448     float curvatureSum = 0.0;
449 
450     /* get accumulative curvature */
451     int left = 0, right;
452     for(int i=0;i<requiredNum;++i)
453     {
454         right = indexVec[i];
455 
```

```
456         /* sum up the curvature of left and right */
457         curvatureSum = 0.0;
458         for(int j=left;j<=right;++j)
459         {
460             curvatureSum+=curvatureVec[j];
461         }
462
463         histogram[i] = curvatureSum;
464
465         left = right+1;
466     }
467
468     /* add last element which is from left to last vertex */
469     curvatureSum = 0.0;
470     for(int i=left;i<segmentNum;++i)
471         curvatureSum+=curvatureVec[i];
472     histogram[requiredNum] = curvatureSum;
473 }
```

5.12.1.9 void getUnitDirection_byEach (const VectorXf & *array*, const int & *pointNum*, VectorXf & *direction*)

Definition at line 226 of file PreComputing.cpp.

```
229 {
230     Vector3f left;
231     float leftNorm;
232     for (int i = 0; i < pointNum; ++i)
233     {
234         left << array(3*i), array(3*i+1), array(3*i+2);
235         leftNorm = left.norm();
236         // I Know it's hardly possible to have smaller norm, but just in case
237         if(leftNorm>=1.0e-8)
238         {
239             for (int j = 0; j < 3; ++j)
240             {
241                 direction(3*i+j) = left(j)/leftNorm;
242             }
243         }
244         else
245         {
246             for (int j = 0; j < 3; ++j)
247             {
248                 direction(3*i+j) = 0;
249             }
250         }
251     }
252 }
```

5.12.1.10 void getUnnormalizedMultivariate (const VectorXf & *array*, const int & *size*, MultiVariate & *rowSequence*)

Definition at line 180 of file PreComputing.cpp.

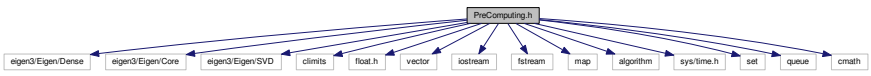
```
183 {
184     MatrixXf normalDirection(size,3);
185     float leftNorm;
186     Vector3f left;
187     VectorXf unitOne(size);
188     for (int j = 0; j < size; ++j)
189     {
190         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
191         leftNorm = left.norm();
192         if(leftNorm >= 1.0e-8)
193         {
194             for(int k=0;k<3;k++)
195                 /* record each line segment normal direction */
196                 normalDirection(j,k) = left(k);
197         }
198         else
199         {
200             for(int k=0;k<3;k++)
201                 /* if norm is small, mark them as zero to tell identical points */
202                 normalDirection(j,k) = 0.0;
```

```
203         }
204         unitOne(j) = 1.0;
205     }
206
207     VectorXf meanNormal(3);
208     for (int i = 0; i < 3; ++i)
209     {
210         meanNormal(i) = normalDirection.transpose().row(i).mean();
211     }
212
213     MatrixXf tempMatrix = normalDirection-unitOne*meanNormal.transpose();
214     rowSequence.covariance = tempMatrix.transpose()*tempMatrix/(size-1);
215     rowSequence.meanVec = meanNormal;
216 }
```

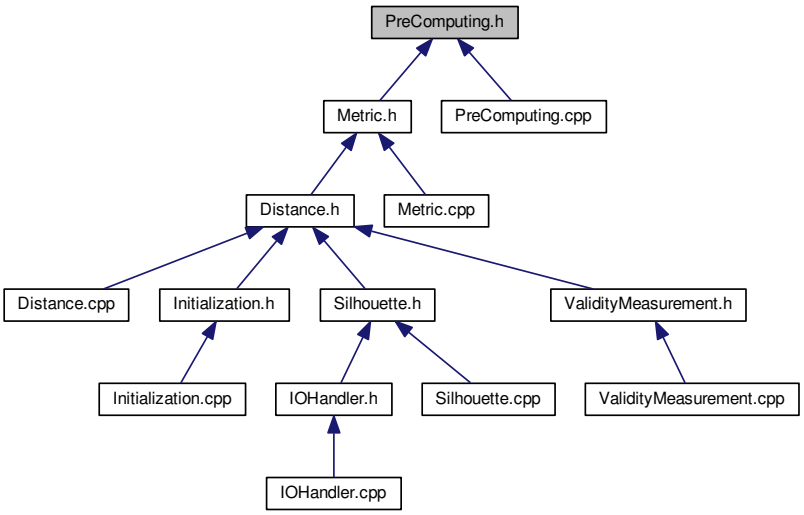
5.13 PreComputing.h File Reference

```
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/SVD>
#include <climits>
#include <float.h>
#include <vector>
#include <iostream>
#include <fstream>
#include <map>
#include <algorithm>
#include <sys/time.h>
#include <set>
#include <queue>
#include <cmath>
```

Include dependency graph for PreComputing.h:



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



Classes

- struct MultiVariate
- struct CurvatureObject
- class CompareFunc

Functions

- void getSequence (const VectorXf &array, const int &size, std::vector< float > &rowSequence)
- const float getRotation (const Eigen::VectorXf &array, const int &size)
- void getNormalMultivariate (const VectorXf &array, const int &size, MultiVariate &rowSequence)
- void getEachFixedSequence (const VectorXf &array, const int &size, std::vector< float > &rowSequence)
- void getUnnormalizedMultivariate (const VectorXf &array, const int &size, MultiVariate &rowSequence)
- void getUnitDirection_byEach (const VectorXf &array, const int &pointNum, VectorXf &direction)
- void getSignatureHist (const Eigen::VectorXf &array, const int &binSize, std::vector< float > &histogram)
- void getSignatureHistSampled (const Eigen::VectorXf &array, const int &binSize, std::vector< float > &histogram)
- void getLinearAngularEntropy (const Eigen::VectorXf &array, const int &bundleSize, std::vector< float > &histogram)
- void getPairWise_byEach (const VectorXf &data, const int &size, std::vector< float > &wiseVec, std::vector< float > &wiseNorm)
- template<typename _Matrix_Type_ >
  _Matrix_Type_ pseudoInverse (const _Matrix_Type_ &a, double epsilon=std::numeric_limits< double >::epsilon())

5.13.1 Function Documentation

5.13.1.1 void getEachFixedSequence (const VectorXf & *array*, const int & *size*, std::vector< float > & *rowSequence*)

Definition at line 135 of file PreComputing.cpp.

```
138 {
139     rowSequence = std::vector<float>(2);
140     float dotValue, leftNorm, meanRotation = 0.0, deviation = 0.0, angle, result;
141     Vector3f left, xRay;
142     xRay << 1.0,0.0,0.0;
143     for (int j = 0; j < size; ++j)
144     {
145         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
146         dotValue = left.dot(xRay);
147         leftNorm = left.norm();
148         if(leftNorm >= 1.0e-8)
149         {
150             result = dotValue/leftNorm;
151             result = min(1.0,(double)result);
152             result = max(-1.0,(double)result);
153             angle = acos(result);
154             meanRotation += angle;
155             deviation += angle*angle;
156         }
157         else
158         {
159             angle = M_PI;
160             meanRotation += angle;
161             deviation += angle*angle;
162         }
163     }
164     meanRotation /= size;
165     rowSequence[0] = meanRotation;
166     int stdDevia = deviation/size-(meanRotation*meanRotation);
167     if(stdDevia<0)
168         stdDevia = 1.0e-8;
169     rowSequence[1] = sqrt(stdDevia);
170 }
```

5.13.1.2 void getLinearAngularEntropy (const Eigen::VectorXf & *array*, const int & *bundleSize*, std::vector< float > & *histogram*)

Definition at line 483 of file PreComputing.cpp.

```
486 {
487     /* if empty vector, should allocate memory ahead of time */
488     if(histogram.empty())
489         histogram = std::vector<float>(2);
490
491     /* get how many vertices you'll have */
492     const int& segmentNum = array.size()/3-1;
493
494     const int& curvatureNum = segmentNum-1;
495
496     /* should partition the whole streamlines into bunleSize segments, and compute the entropy */
497
498     std::vector<float> segmentVec(segmentNum), curvatureVec(curvatureNum);
499
500     Eigen::Vector3f firstSeg, secondSeg;
501
502     /* discrete curvature */
503     float curva;
504
505     float lengthSum = 0.0, curveSum = 0.0;
506     int vecIndex = 0;
507     for(int i=0;i<curvatureNum;++i)
508     {
509         for(int j=0;j<3;++j)
510         {
511             firstSeg(j)=array(3*i+3+j)-array(3*i+j);
512             secondSeg(j)=array(3*i+6+j)-array(3*i+3+j);
513         }
514
```

```
515         if(firstSeg.norm()<1.0e-6 || secondSeg.norm()<1.0e-6)
516         {
517             segmentVec[i] = 0.0;
518             curvatureVec[i]= 0.0;
519             continue;
520         }
521 
522         float firstSegNorm = firstSeg.norm(), secondSegNorm = secondSeg.norm();
523         if(firstSegNorm<1.0e-8 || secondSegNorm<1.0e-8)
524             curva = 0.0;
525         else
526         {
527             curva = firstSeg.dot(secondSeg)/firstSegNorm/secondSegNorm;
528 
529             /* clip curvature into range [-1.0, 1.0] */
530             curva = std::min(float(1.0), curva);
531             curva = std::max(float(-1.0), curva);
532 
533             curva = acos(curva);
534         }
535 
536         /* store in the vector */
537         curvatureVec[i]=curva;
538         curveSum+=curva;
539 
540         /* store path */
541         segmentVec[i] = firstSeg.norm();
542         lengthSum+=segmentVec[i];
543     }
544 
545     int i = curvatureNum;
546     for(int j=0;j<3;++j)
547     {
548         firstSeg(j)=array(3*i+3+j)-array(3*i+j);
549     }
550     segmentVec[i] = firstSeg.norm();
551     lengthSum+=segmentVec[i];
552 
553 
554     /* should deal with exceptional case if lengthSum == 0 or curveSum == 0 */
555     if(lengthSum<1.0e-6)
556     {
557         histogram[0] = 1.0;
558     }
559     else
560     {
561         /* get ratio for the vec */
562         const int& segmentQuotient = segmentNum/bundleSize;
563         const int& segmentResidue = segmentNum%bundleSize;
564 
565         /* get the vec for bundleSize */
566         std::vector<float> lengthVec(bundleSize);
567 
568         float tempLength, linearEntropy = 0.0, prob;
569         int left, right;
570         for(int k = 0;k<bundleSize-1;++k)
571         {
572             tempLength = 0.0;
573             left = k*segmentQuotient, right = (k+1)*segmentQuotient;
574             for(int i = left;i<right;++i)
575                 tempLength+=segmentVec[i];
576 
577             prob = tempLength/lengthSum;
578 
579             if(prob>1.0e-6)
580                 linearEntropy += prob*log2f(prob);
581         }
582 
583         left = (bundleSize-1)*segmentQuotient, right = segmentNum;
584         tempLength = 0.0;
585         for(int i=left;i<right;++i)
586         {
587             tempLength+=segmentVec[i];
588         }
589         if(prob>1.0e-6)
590             prob = tempLength/lengthSum;
591         linearEntropy += prob*log2f(prob);
592 
593         linearEntropy = -linearEntropy/log2f(float(bundleSize));
594         histogram[0] = linearEntropy;
595     }
596 
597     /* deal with curveSum == 1.0 */
598     if(curveSum<1.0e-6)
599     {
600         histogram[1] = 1.0;
601     }
```

```
602     else
603     {
604         const int& curvatureQuotient = curvatureNum/bundleSize;
605         const int& curvatureResidue = curvatureNum%bundleSize;
606 
607         /* get the vec for bundleSize */
608         std::vector<float> curveVec(bundleSize);
609 
610         float tempCurve, angularEntropy = 0.0, prob;
611         int left, right;
612         for(int k = 0;k<bundleSize-1;++k)
613         {
614             tempCurve = 0.0;
615             left = k*curvatureQuotient, right = (k+1)*curvatureQuotient;
616             for(int i=left;i<right;++i)
617                 tempCurve+=curvatureVec[i];
618 
619             prob = tempCurve/curveSum;
620             if(prob>1.0e-6)
621                 angularEntropy += prob*log2f(prob);
622         }
623 
624         left = (bundleSize-1)*curvatureQuotient, right = curvatureNum;
625         tempCurve = 0.0;
626         for(int i=left;i<right;++i)
627         {
628             tempCurve+=curvatureVec[i];
629         }
630         prob = tempCurve/curveSum;
631         if(prob>1.0e-6)
632             angularEntropy += prob*log2f(prob);
633 
634         angularEntropy = -angularEntropy/log2f(float(bundleSize));
635         histogram[1] = angularEntropy;
636     }
637 }
```

5.13.1.3 void getNormalMultivariate (const VectorXf & *array,* const int & *size,* MultiVariate & *rowSequence*)

Definition at line 89 of file PreComputing.cpp.

```
92 {
93     MatrixXf normalDirection(size,3);
94     float leftNorm;
95     Vector3f left;
96     VectorXf unitOne(size);
97     for (int j = 0; j < size; ++j)
98     {
99         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
100         leftNorm = left.norm();
101         if(leftNorm >= 1.0e-8)
102         {
103             for(int k=0;k<3;k++)
104                 /* record each line segment normal direction */
105                 normalDirection(j,k) = left(k)/leftNorm;
106         }
107         else
108         {
109             for(int k=0;k<3;k++)
110                 /* if norm is small, mark them as zero to tell identical points */
111                 normalDirection(j,k) = 0.0;
112         }
113         unitOne(j) = 1.0;
114     }
115 
116     VectorXf meanNormal(3);
117     for (int i = 0; i < 3; ++i)
118     {
119         meanNormal(i) = normalDirection.transpose().row(i).mean();
120     }
121 
122     MatrixXf tempMatrix = normalDirection-unitOne*meanNormal.transpose();
123     rowSequence.covariance = tempMatrix.transpose()*tempMatrix/(size-1);
124     rowSequence.meanVec = meanNormal;
125 }
```

5.13.1.4 void getPairWise_byEach (const VectorXf & *data,* const int & *size,* std::vector< float > & *wiseVec,* std::vector< float > & *wiseNorm*)

Definition at line 263 of file PreComputing.cpp.

```
267 {
268     if(wiseVec.empty())
269         wiseVec = std::vector<float>(3*size);
270
271     if(wiseNorm.empty())
272         wiseNorm = std::vector<float>(size);
273
274     for (int i = 0; i < size; ++i)
275     {
276         float leftNorm;
277         Vector3f left;
278         left << data(3*i+3)-data(3*i),data(3*i+4)-data(3*i+1),data(3*i+5)-data(3*i+2);
279         leftNorm = left.norm();
280         if(leftNorm >= 1.0e-8)
281         {
282             for (int j = 0; j < 3; ++j)
283             {
284                 wiseVec[3*i+j] = left(j)/leftNorm;
285             }
286             wiseNorm[i] = leftNorm;
287         }
288         else
289         {
290             for (int j = 0; j < 3; ++j)
291             {
292                 wiseVec[3*i+j] = 0.0;
293             }
294             wiseNorm[i] = 0.0;
295         }
296     }
297 }
```

5.13.1.5 const float getRotation (const Eigen::VectorXf & *array,* const int & *size*)

5.13.1.6 void getSequence (const VectorXf & *array,* const int & *size,* std::vector< float > & *rowSequence*)

Definition at line 17 of file PreComputing.cpp.

```
20 {
21     rowSequence = std::vector<float>(2);
22     float dotValue, leftNorm, rightNorm, meanRotation = 0.0, deviation = 0.0, angle, result;
23     Vector3f left, right;
24     for (int j = 0; j < size; ++j)
25     {
26         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
27         right << array(j*3+6)-array(j*3+3), array(j*3+7)-array(j*3+4), array(j*3+8)-array(j*3+5);
28         dotValue = left.dot(right);
29         leftNorm = left.norm();
30         rightNorm = right.norm();
31         if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
32         {
33             result = dotValue/leftNorm/rightNorm;
34             result = min(1.0,(double)result);
35             result = max(-1.0,(double)result);
36             angle = acos(result);
37             meanRotation += angle;
38             deviation += angle*angle;
39         }
40     }
41     meanRotation /= size;
42     rowSequence[0] = meanRotation;
43     int stdDevia = deviation/size-(meanRotation*meanRotation);
44     if(stdDevia<0)
45         stdDevia = 1.0e-8;
46     rowSequence[1] = sqrt(stdDevia);
47 }
```

5.13.1.7 void getSignatureHist (const Eigen::VectorXf & *array*, const int & *binSize*, std::vector< float > & *histogram*)

Definition at line 307 of file PreComputing.cpp.

```
310 {
311     /* if empty vector, should allocate memory ahead of time */
312     if(histogram.empty())
313         histogram = std::vector<float>(binNum);
314 
315     /* get how many vertices you'll have */
316     const int& segmentNum = array.size()/3-1;
317 
318     /* how many vertices on each bin on average */
319     const int& binSize = segmentNum/binNum;
320 
321     /* first several has binSize+1 vertices, while the rest have binSize vertices */
322     const int& residueNum = segmentNum%binNum;
323 
324     if(binSize<1)
325     {
326         std::cout << "Error for bin size calculation!" << std::endl;
327         exit(1);
328     }
329 
330     int totalVertexOnBin = binSize+1, index = 0;
331     float dotValue, leftNorm, rightNorm, meanRotation = 0.0, result, rotationSum;
332     Vector3f left, right;
333     for (int i = 0; i < binNum; ++i)
334     {
335         /* would reduce that to binSize if i>=residueNum */
336         if(i==residueNum)
337             totalVertexOnBin = binSize;
338 
339         /* reset the rotationSum */
340         rotationSum = 0.0;
341         for(int j=0;j<totalVertexOnBin;++j)
342         {
343             left << array(index*3+3)-array(index*3),
344                     array(index*3+4)-array(index*3+1),
345                     array(index*3+5)-array(index*3+2);
346             right << array(index*3+6)-array(index*3+3),
347                      array(index*3+7)-array(index*3+4),
348                      array(index*3+8)-array(index*3+5);
349             dotValue = left.dot(right);
350             leftNorm = left.norm();
351             rightNorm = right.norm();
352             if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
353             {
354                 result = dotValue/leftNorm/rightNorm;
355                 result = min(1.0,(double)result);
356                 result = max(-1.0,(double)result);
357                 rotationSum += acos(result);
358             }
359             ++index;
360         }
361 
362         histogram[i] = rotationSum;
363     }
364     assert(index==segmentNum);
365 }
```

5.13.1.8 void getSignatureHistSampled (const Eigen::VectorXf & *array*, const int & *binSize*, std::vector< float > & *histogram*)

Definition at line 375 of file PreComputing.cpp.

```
378 {
379     /* if empty vector, should allocate memory ahead of time */
380     if(histogram.empty())
381         histogram = std::vector<float>(binNum);
382 
383     /* get how many vertices you'll have */
384     const int& segmentNum = array.size()/3-2;
385 
386     /* preset a priority_queue to get the sampled points in maximal curvatures */
387     priority_queue<CurvatureObject, std::vector<CurvatureObject>, CompareFunc> pQueue;
388 
```

```
389     std::vector<float> curvatureVec(segmentNum);
390
391     Eigen::Vector3f firstSeg, secondSeg;
392
393     /* discrete curvature */
394     float curva;
395
396     int vecIndex = 0;
397     for(int i=0;i<segmentNum;++i)
398     {
399         for(int j=0;j<3;++j)
400         {
401             firstSeg(j)=array(3*i+3+j)-array(3*i+j);
402             secondSeg(j)=array(3*i+6+j)-array(3*i+3+j);
403         }
404
405         float firstSegNorm = firstSeg.norm(), secondSegNorm = secondSeg.norm();
406         if(firstSegNorm<1.0e-8 || secondSegNorm<1.0e-8)
407             curva = 0.0;
408         else
409         {
410             curva = firstSeg.dot(secondSeg)/firstSegNorm/secondSegNorm;
411
412             /* clip curvature into range [-1.0, 1.0] */
413             curva = std::min(float(1.0), curva);
414             curva = std::max(float(-1.0), curva);
415
416             curva = acos(curva);
417         }
418         /* store in the vector */
419         curvatureVec[vecIndex++]=curva;
420
421         /* push it into the priority queue */
422         pQueue.push(CurvatureObject(curva, i));
423
424     }
425
426     /* get the first binNum-1 object */
427     CurvatureObject top;
428
429     /* use ordered_set to sort the index */
430     std::vector<int> indexVec;
431
432     int indexNum = 0;
433     const int& requiredNum = binNum-1;
434     while(indexNum<requiredNum && !pQueue.empty())
435     {
436         top = pQueue.top();
437         indexVec.push_back(top.index);
438         pQueue.pop();
439         ++indexNum;
440     }
441
442     assert(indexVec.size()==requiredNum);
443
444     /* sort the vec */
445     std::sort(indexVec.begin(), indexVec.end());
446
447     /* start sampling to make a curvature histogram */
448     float curvatureSum = 0.0;
449
450     /* get accumulative curvature */
451     int left = 0, right;
452     for(int i=0;i<requiredNum;++i)
453     {
454         right = indexVec[i];
455
456         /* sum up the curvature of left and right */
457         curvatureSum = 0.0;
458         for(int j=left;j<=right;++j)
459         {
460             curvatureSum+=curvatureVec[j];
461         }
462
463         histogram[i] = curvatureSum;
464
465         left = right+1;
466     }
467
468     /* add last element which is from left to last vertex */
469     curvatureSum = 0.0;
470     for(int i=left;i<segmentNum;++i)
471         curvatureSum+=curvatureVec[i];
472     histogram[requiredNum] = curvatureSum;
473 }
```

5.13.1.9 void getUnitDirection_byEach (const VectorXf & *array*, const int & *pointNum*, VectorXf & *direction*)

Definition at line 226 of file PreComputing.cpp.

```
229 {
230     Vector3f left;
231     float leftNorm;
232     for (int i = 0; i < pointNum; ++i)
233     {
234         left << array(3*i), array(3*i+1), array(3*i+2);
235         leftNorm = left.norm();
236         // I Know it's hardly possible to have smaller norm, but just in case
237         if(leftNorm>=1.0e-8)
238         {
239             for (int j = 0; j < 3; ++j)
240             {
241                 direction(3*i+j) = left(j)/leftNorm;
242             }
243         }
244         else
245         {
246             for (int j = 0; j < 3; ++j)
247             {
248                 direction(3*i+j) = 0;
249             }
250         }
251     }
252 }
```

5.13.1.10 void getUnnormalizedMultivariate (const VectorXf & *array*, const int & *size*, MultiVariate & *rowSequence*)

Definition at line 180 of file PreComputing.cpp.

```
183 {
184     MatrixXf normalDirection(size,3);
185     float leftNorm;
186     Vector3f left;
187     VectorXf unitOne(size);
188     for (int j = 0; j < size; ++j)
189     {
190         left << array(j*3+3)-array(j*3), array(j*3+4)-array(j*3+1), array(j*3+5)-array(j*3+2);
191         leftNorm = left.norm();
192         if(leftNorm >= 1.0e-8)
193         {
194             for(int k=0;k<3;k++)
195                 /* record each line segment normal direction */
196                 normalDirection(j,k) = left(k);
197         }
198         else
199         {
200             for(int k=0;k<3;k++)
201                 /* if norm is small, mark them as zero to tell identical points */
202                 normalDirection(j,k) = 0.0;
203         }
204         unitOne(j) = 1.0;
205     }
206
207     VectorXf meanNormal(3);
208     for (int i = 0; i < 3; ++i)
209     {
210         meanNormal(i) = normalDirection.transpose().row(i).mean();
211     }
212
213     MatrixXf tempMatrix = normalDirection-unitOne*meanNormal.transpose();
214     rowSequence.covariance = tempMatrix.transpose()*tempMatrix/(size-1);
215     rowSequence.meanVec = meanNormal;
216 }
```

5.13.1.11 template<typename _Matrix_Type_ > _Matrix_Type_ pseudoInverse (const _Matrix_Type_ & *a,* double *epsilon* = std::numeric_limits<double>::epsilon())

Definition at line 200 of file PreComputing.h.

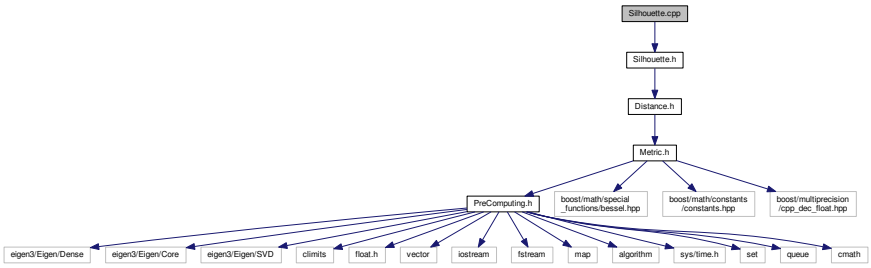
```
202 {
203     Eigen::JacobiSVD< _Matrix_Type_ > svd(a ,Eigen::ComputeThinU | Eigen::ComputeThinV);
204     double tolerance = epsilon * std::max(a.cols(), a.rows())
205                       *svd.singularValues().array().abs()(0);
206     return svd.matrixV() *  (svd.singularValues().array().abs() > tolerance).select
207            (svd.singularValues().array().inverse(), 0).matrix().asDiagonal()
208            * svd.matrixU().adjoint();
209 }
```

5.14 README.md File Reference

5.15 Silhouette.cpp File Reference

```
#include "Silhouette.h"
```

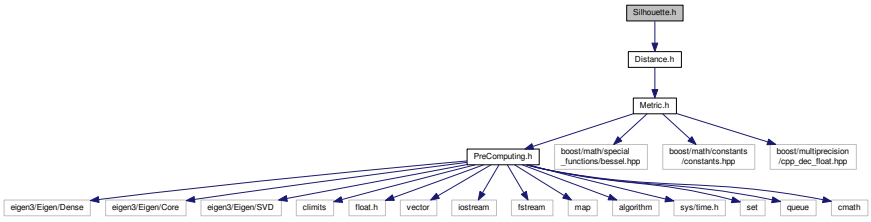
Include dependency graph for Silhouette.cpp:



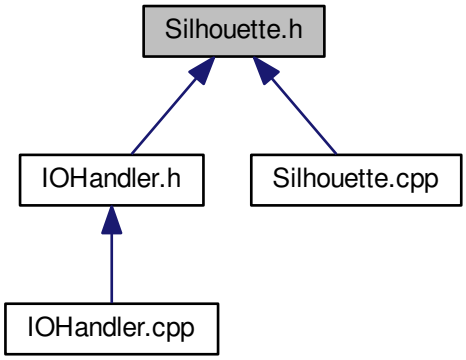
5.16 Silhouette.h File Reference

```
#include "Distance.h"
```

Include dependency graph for Silhouette.h:



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



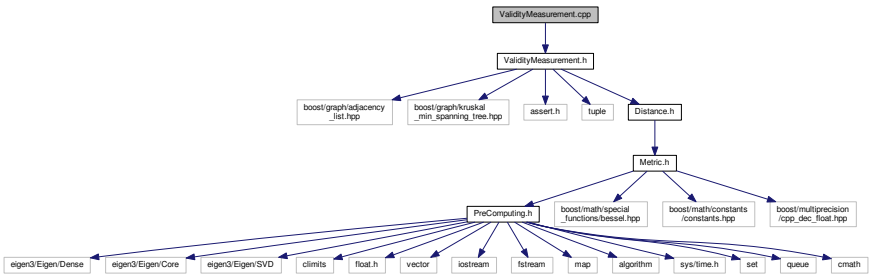
Classes

- class Silhouette

5.17 ValidityMeasurement.cpp File Reference

```
#include "ValidityMeasurement.h"
```

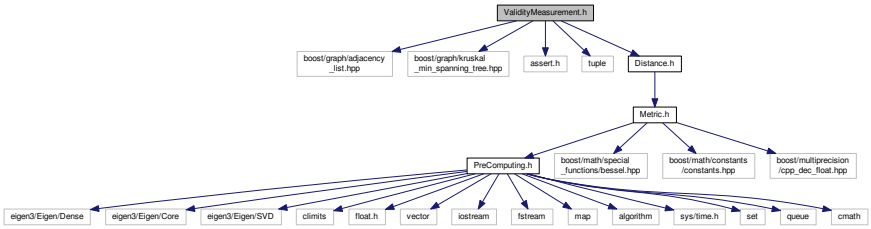
Include dependency graph for ValidityMeasurement.cpp:



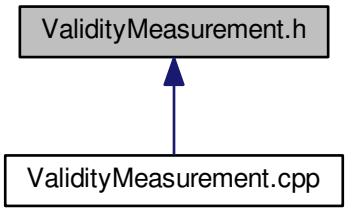
5.18 ValidityMeasurement.h File Reference

```
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/kruskal_min_spanning_tree.hpp>
#include <assert.h>
#include <tuple>
#include "Distance.h"
```

Include dependency graph for ValidityMeasurement.h:



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



Classes

- class ValidityMeasurement

Index