HelperClass

The implmentation for blood flow resampling and posterior visualization and analysis

Generated by Doxygen 1.8.11

Contents

Chapter 1

HelpClass

Inside this folder, it includes many valuable implementations for pre-processing or post-processing of quantitatively concluding the clustering results on different simularity measures,

- The time-based sampling method for **blood flow** to make sure that all the points along blood flow pathlines are rigorously at the same time step starting from the beginning
- The blood flow data set after time-based sampling, which has exactly the same geometric information to the original pathlines
- C++ code to re-calculate the clustering evaluation metrics by reading from the clustering result
- The python script to fetch the evaluation metrics and calculation time
- The nonlinear code to map and calculate the clustering evaluation metrics
- The R visualization code for ranking-based circle mapping
- The Shell script to perform all the calculation, visualization and conclusion for the clustering evaluation
- The python script to conclude the finalized clustering results for all the streamlines/pathlines

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 average_ranking Namespace Reference

Variables

- list silhouette = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'AHC-single', 'BIRCH', 'AP', 'DB↩ SCAN', 'OPTICS', 'SC-eigen']
- list gamma = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'BIRCH', 'DBSCAN', 'AHC-single', 'OPTICS', 'SC-eigen', 'AP']
- list dbindex = ['PCA', 'kmeans', 'AHC-single', 'kmedoids', 'AHC-average', 'BIRCH', 'SC-eigen', 'SC-kmeans', 'DBSCAN', 'OPTICS', 'AP']
- list validity = ['DBSCAN', 'PCA', 'AHC-single', 'SC-kmeans', 'BIRCH', 'AHC-average', 'kmeans', 'kmedoids', 'AP', 'OPTICS', 'SC-eigen']
- list sil_norm = ['d_R', 'd_S', 'd_P', 'd_E', 'd_M', 'd_F', 'd_H', 'd_G']
- list gamma_norm = ['d_R', 'd_E', 'd_H', 'd_S', 'd_M', 'd_F', 'd_G', 'd_P']
- list db_norm = ['d_G', 'd_M', 'd_S', 'd_H', 'd_E', 'd_P', 'd_F', 'd_R']
- list validity_norm = ['d_M', 'd_R', 'd_P', 'd_F', 'd_H', 'd_S', 'd_E', 'd_G']
- dictionary average_ranking = {clustering:0 for clustering in silhouette}
- dictionary average_norm = {norm:0 for norm in sil_norm}
- list order = [100, None]

5.1.1 Variable Documentation

5.1.1.1 dictionary average_ranking.average_norm = {norm:0 for norm in sil_norm}

Definition at line 12 of file average_ranking.py.

5.1.1.2 dictionary average_ranking.average_ranking = {clustering:0 for clustering in silhouette}

Definition at line 11 of file average_ranking.py.

5.1.1.3 list average_ranking.db_norm = ['d_G', 'd_M', 'd_S', 'd_H', 'd_E', 'd_P', 'd_F', 'd_R']

Definition at line 8 of file average_ranking.py.

5.1.1.4 list average_ranking.dbindex = ['PCA', 'kmeans', 'AHC-single', 'kmedoids', 'AHC-average', 'BIRCH', 'SC-eigen', 'SC-kmeans', 'DBSCAN', 'OPTICS', 'AP']

Definition at line 3 of file average_ranking.py.

5.1.1.5 list average_ranking.gamma = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'BIRCH', 'DBSCAN', 'AHC-single', 'OPTICS', 'SC-eigen', 'AP']

Definition at line 2 of file average_ranking.py.

5.1.1.6 list average_ranking.gamma_norm = ['d_R', 'd_E', 'd_H', 'd_S', 'd_M', 'd_F', 'd_G', 'd_P']

Definition at line 7 of file average_ranking.py.

5.1.1.7 list average_ranking.order = [100, None]

Definition at line 19 of file average_ranking.py.

5.1.1.8 list average_ranking.sil_norm = ['d_R', 'd_S', 'd_P', 'd_E', 'd_M', 'd_F', 'd_H', 'd_G']

Definition at line 6 of file average_ranking.py.

5.1.1.9 list average_ranking.silhouette = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'AHC-single', 'BIRCH', 'AP', 'DBSCAN', 'OPTICS', 'SC-eigen']

Definition at line 1 of file average_ranking.py.

5.1.1.10 list average_ranking.validity = ['DBSCAN', 'PCA', 'AHC-single', 'SC-kmeans', 'BIRCH', 'AHC-average', 'kmeans', 'kmedoids', 'AP', 'OPTICS', 'SC-eigen']

Definition at line 4 of file average_ranking.py.

5.1.1.11 list average_ranking.validity_norm = ['d_M', 'd_R', 'd_P', 'd_F', 'd_H', 'd_S', 'd_E', 'd_G']

Definition at line 9 of file average_ranking.py.

5.2 fetch_data Namespace Reference

Functions

- def get_distance_limit (file_position)
- def extract_evaluation_data (distance_range, data_folder)
- def extract_single_readme (distance_range, data_folder)
- def extract_norm_readme (distance_range, data_folder)
- def get_average (lmethod_evaluation, sc_eigen_evaluation)
- def generate_text (evaluation_data, storage_name)
- def generate_time (evaluation_data, storage_name)
- def merge_two_dicts (first, second)
- def extract_full_data ()

5.2.1 Function Documentation

5.2.1.1 def fetch_data.extract_evaluation_data (*distance_range, data_folder*)

Definition at line 40 of file fetch_data.py.

```
40 def extract_evaluation_data(distance_range, data_folder):
41     evaluation = {}
42     norm_list = ['0','1','2','4','12','13','14','15']
43     for d_folder in listdir(data_folder):
44         readme = data_folder+'/'+d_folder+'/README'
45         with open(readme) as r:
46             content = r.readlines()
47         norm_found = False
48         norm=None
49         evaluation[d_folder] = {}
50         for val in norm_list:
51             evaluation[d_folder][val] = {'silhouette':-10000.0, 'gamma':-10000.0, 'db index':-10000.0, 'validity':-10000.0, 'time':-10000.0}
52 
53         if d_folder=='kmeans':
54             evaluation['PCA'] = {}
55             for val in norm_list:
56                 evaluation['PCA'][val] = {'silhouette':-10000.0, 'gamma':-10000.0, 'db index':-10000.0, 'validity':-10000.0, 'time':-10000.0}
57 
58         norm_found = False
59         for x in content:
60             if x=='' or x=='\n':
61                 continue
62             if norm_found is False:
63                 norm_pos = x.find('norm')
64                 Norm_pos = x.find('Norm:')
65                 pca_pos = x.find('PCA')
66                 if norm_pos==-1 and Norm_pos==-1 and pca_pos==-1:
67                     continue
68 
69                 if norm_pos!=-1:
70                     end_pos = norm_pos+5
71                     while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
72                         end_pos+=1
73                     norm = x[norm_pos+5:end_pos]
74                 elif Norm_pos!=-1:
75                     end_pos = Norm_pos+6
76                     while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
77                         end_pos+=1
78                     norm = x[norm_pos+6:end_pos]
79                 elif pca_pos!=-1:
80                     norm = 'PCA'
81                 norm_found = True
82 
83             if norm_found is True and (norm in norm_list or norm=='PCA'):
84                 sil_pos = x.find('silhouette:')
85                 gamma_pos = x.find('statistic is:')
86                 dbindex_pos = x.find('DB index is:')
87                 validity_pos = x.find('measure is:')
88                 measurement_pos = x.find('measurement is:')
89 
90                 if sil_pos!=-1:
91                     start_pos = sil_pos+len('silhouette:')+1
92                     while x[start_pos]==' ':
93                         start_pos+=1
94                     end_pos=start_pos
95                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
96                         end_pos+=1
97                     val_str = x[start_pos:end_pos]
98                     if val_str !='-nan' and val_str !='inf':
99                         if norm=='PCA':
100                             evaluation[norm]['0']['silhouette'] = float(x[start_pos:end_pos])
101                         else:
102                             evaluation[d_folder][norm]['silhouette'] = float(x[start_pos:end_pos])
103 
104                 if gamma_pos!=-1:
105                     start_pos = gamma_pos+len('statistic is:')+1
106                     while x[start_pos]==' ':
107                         start_pos+=1
108                     end_pos=start_pos
109                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
110                         end_pos+=1
111                     val_str = x[start_pos:end_pos]
112                     if val_str !='-nan' and val_str !='inf':
113                         if norm=='PCA':
```

```
                            evaluation[norm]['0']['gamma'] = float(x[start_pos:end_pos])
                        else:
                            evaluation[d_folder][norm]['gamma'] = float(x[start_pos:end_pos])

                if dbindex_pos!=-1:
                    start_pos = dbindex_pos+len('DB index is:')+1
                    while x[start_pos]==' ':
                        start_pos+=1
                    end_pos=start_pos
                    while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                        end_pos+=1
                    val_str = x[start_pos:end_pos]
                    if val_str !='-nan' and val_str !='inf':
                        if norm=='PCA':
                            evaluation[norm]['0']['db index'] = float(x[start_pos:end_pos])
                        else:
                            evaluation[d_folder][norm]['db index'] = float(x[start_pos:end_pos])
                    norm_found = False

                if validity_pos!=-1:
                    start_pos = validity_pos+len('measure is:')+1
                    while x[start_pos]==' ':
                        start_pos+=1
                    end_pos=start_pos
                    while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                        end_pos+=1
                    val_str = x[start_pos:end_pos]
                    if val_str !='-nan' and val_str !='inf':
                        if norm=='PCA':
                            evaluation[norm]['0']['validity'] = float(x[start_pos:end_pos])/distance_range[norm]
                        else:
                            evaluation[d_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]

                elif measurement_pos!=-1:
                    start_pos = measurement_pos+len('measurement is:')+1
                    while x[start_pos]==' ':
                        start_pos+=1
                    end_pos=start_pos
                    while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                        end_pos+=1
                    val_str = x[start_pos:end_pos]
                    if val_str !='-nan' and val_str !='inf':
                        if norm=='PCA':
                            evaluation[norm]['0']['validity'] = float(x[start_pos:end_pos])/distance_range[norm]
                        else:
                            evaluation[d_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]

                pca_time_tag = x.find('PCA+K_Means operation takes:')
                kmeans_time_tag = x.find('K-means on norm')
                kmedoid_time_tag = x.find('Direct K_Means operation time for norm')

                if pca_time_tag!=-1 or kmeans_time_tag!=-1 or kmedoid_time_tag!=-1:
                    takes = x.find('takes:')
                    if takes==-1:
                        raise ValueError('Error for time search!')
                    start_pos = takes+len('takes:')
                    while x[start_pos]==' ':
                        start_pos+=1
                    end_pos=start_pos
                    while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ' and x[end_pos]!='s':
                        end_pos+=1
                    val_str = x[start_pos:end_pos]
                    if norm=='PCA':
                        evaluation[norm]['0']['time'] = float(x[start_pos:end_pos])
                    else:
                        evaluation[d_folder][norm]['time'] = float(x[start_pos:end_pos])
                else:
                    takes = x.find('takes:')
                    if takes!=-1:
                        start_pos = takes+len('takes:')
                        while x[start_pos]==' ':
                            start_pos+=1
                        end_pos=start_pos
                        while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ' and x[end_pos]!='s':
                            end_pos+=1
                        val_str = x[start_pos:end_pos]
                        if evaluation[d_folder][norm]['time']<=-9999.0:
                            evaluation[d_folder][norm]['time'] = float(x[start_pos:end_pos])
                        else:
                            evaluation[d_folder][norm]['time'] += float(x[start_pos:end_pos])

```

```
195     return evaluation
196
197
198 # read AP and sc_eigen that only has one README file inside
```

5.2.1.2 def fetch_data.extract_full_data ( )

Definition at line 500 of file fetch_data.py.

```
500 def extract_full_data():
501     distance_range=get_distance_limit('dist_range')
502
503     print(distance_range)
504
505     lmethod_evaluation = extract_evaluation_data(distance_range, '
      optimal_clustering/lmethod')
506     sc_eigen_evaluation = extract_evaluation_data(distance_range, '
      optimal_clustering/sc_eigen_number')
507     average_evaluation = get_average(lmethod_evaluation, sc_eigen_evaluation)
508     print(average_evaluation['PCA']['0'])
509
510     ap_evaluation = extract_single_readme(distance_range, 'AP')
511     full_evaluation = merge_two_dicts(average_evaluation, ap_evaluation)
512
513     sc_eigen_evaluation = extract_single_readme(distance_range, 'sc_eigen')
514     full_evaluation = merge_two_dicts(full_evaluation, sc_eigen_evaluation)
515
516     birch_evaluation = extract_norm_readme(distance_range, 'birch')
517     full_evaluation = merge_two_dicts(full_evaluation, birch_evaluation)
518
519     dbscan_evaluation = extract_norm_readme(distance_range, 'dbscan')
520     full_evaluation = merge_two_dicts(full_evaluation, dbscan_evaluation)
521
522     optics_evaluation = extract_norm_readme(distance_range, 'optics')
523     full_evaluation = merge_two_dicts(full_evaluation, optics_evaluation)
524
525     generate_text(full_evaluation, 'evaluation')
526
527     generate_time(full_evaluation, 'time')
528
529
```

5.2.1.3 def fetch_data.extract_norm_readme (*distance_range, data_folder*)

Definition at line 314 of file fetch_data.py.

```
314 def extract_norm_readme(distance_range, data_folder):
315     evaluation = {data_folder:{}}
316     norm_list = ['0','1','2','4','12','13','14','15']
317     for val in norm_list:
318         evaluation[data_folder][val] = {'silhouette':-10000.0, 'gamma':-10000.0, 'db index':-10000.0, '
      validity':-10000.0, 'time':-10000.0}
319
320     for norm in listdir(data_folder):
321         readme = data_folder+'/'+norm+'/README'
322         with open(readme) as r:
323             content = r.readlines()
324
325         for x in content:
326             if x=='' or x=='\n':
327                 continue
328
329             sil_pos = x.find('silhouette:')
330             gamma_pos = x.find('statistic is:')
331             dbindex_pos = x.find('DB index is:')
332             validity_pos = x.find('measure is:')
333             measurement_pos = x.find('measurement is:')
334
335             if sil_pos!=-1:
336                 start_pos = sil_pos+len('silhouette:')+1
337                 while x[start_pos]==' ':
338                     start_pos+=1
```

```
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['silhouette'] = float(x[start_pos:end_pos])

            if gamma_pos!=-1:
                start_pos = gamma_pos+len('statistic is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['gamma'] = float(x[start_pos:end_pos])

            if dbindex_pos!=-1:
                start_pos = dbindex_pos+len('DB index is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['db index'] = float(x[start_pos:end_pos])
                norm_found = False

            if validity_pos!=-1:
                start_pos = validity_pos+len('measure is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]

            elif measurement_pos!=-1:
                start_pos = measurement_pos+len('measurement is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]

            takes = x.find('takes:')
            if takes!=-1:
                start_pos = takes+len('takes:')
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ' and x[end_pos]!='s':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if evaluation[data_folder][norm]['time']<=-9999.0:
                    evaluation[data_folder][norm]['time'] = float(x[start_pos:end_pos])
                else:
                    evaluation[data_folder][norm]['time'] += float(x[start_pos:end_pos])

    return evaluation


```

5.2.1.4 def fetch_data.extract_single_readme (*distance_range*, *data_folder*)

Definition at line 199 of file fetch_data.py.

```
199 def extract_single_readme(distance_range, data_folder):
200     evaluation = {data_folder:{}}
201     norm_list = ['0','1','2','4','12','13','14','15']
```

```
    for val in norm_list:
        evaluation[data_folder][val] = {'silhouette':-10000.0, 'gamma':-10000.0, 'db index':-10000.0, 'validity':-10000.0, 'time':-10000.0}

    readme = data_folder+'/README'
    with open(readme) as r:
        content = r.readlines()
    norm_found = False
    norm=None

    norm_found = False
    for x in content:
        if x=='' or x=='\n':
            continue
        if norm_found is False:
            norm_pos = x.find('norm')
            Norm_pos = x.find('Norm:')
            if norm_pos==-1 and Norm_pos==-1:
                continue

            if norm_pos!=-1:
                end_pos = norm_pos+5
                while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
                    end_pos+=1
                norm = x[norm_pos+5:end_pos]
            elif Norm_pos!=-1:
                end_pos = Norm_pos+6
                while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
                    end_pos+=1
                norm = x[norm_pos+6:end_pos]
            norm_found = True

        if norm_found is True and norm in norm_list:
            sil_pos = x.find('silhouette:')
            gamma_pos = x.find('statistic is:')
            dbindex_pos = x.find('DB index is:')
            validity_pos = x.find('measure is:')
            measurement_pos = x.find('measurement is:')

            if sil_pos!=-1:
                start_pos = sil_pos+len('silhouette:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['silhouette'] = float(x[start_pos:end_pos])

            if gamma_pos!=-1:
                start_pos = gamma_pos+len('statistic is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['gamma'] = float(x[start_pos:end_pos])

            if dbindex_pos!=-1:
                start_pos = dbindex_pos+len('DB index is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['db index'] = float(x[start_pos:end_pos])
                norm_found = False

            if validity_pos!=-1:
                start_pos = validity_pos+len('measure is:')+1
                while x[start_pos]==' ':
                    start_pos+=1
                end_pos=start_pos
                while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
                    end_pos+=1
                val_str = x[start_pos:end_pos]
                if val_str !='-nan' and val_str !='inf':
                    evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]

            elif measurement_pos!=-1:
                start_pos = measurement_pos+len('measurement is:')+1
```

```
287                 while x[start_pos]==' ':
288                     start_pos+=1
289                 end_pos=start_pos
290                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
291                     end_pos+=1
292                 val_str = x[start_pos:end_pos]
293                 if val_str !='-nan' and val_str !='inf':
294                     evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[norm]
295 
296             takes = x.find('takes:')
297             if takes!=-1:
298                 start_pos = takes+len('takes:')
299                 while x[start_pos]==' ':
300                     start_pos+=1
301                 end_pos=start_pos
302                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ' and x[end_pos]!='s':
303                     end_pos+=1
304                 val_str = x[start_pos:end_pos]
305                 if evaluation[data_folder][norm]['time']<=-9999.0:
306                     evaluation[data_folder][norm]['time'] = float(x[start_pos:end_pos])
307                 else:
308                     evaluation[data_folder][norm]['time'] += float(x[start_pos:end_pos])
309 
310     return evaluation
311 
312 
313 # read some data files like birch, dbscan, optics
```

5.2.1.5 def fetch_data.generate_text (*evaluation_data*, *storage_name*)

Definition at line 431 of file fetch_data.py.

```
431 def generate_text(evaluation_data, storage_name):
432     storage = open(storage_name, 'w')
433     clustering_algorithms = ['kmeans', 'kmedoids', 'AHC_single', 'AHC_average', 'birch', 'dbscan', 'optics', 'sc_kmeans', 'sc_eigen', 'AP', 'PCA']
434     norm_order = ['0', '1', '2', '4', '12', '13', '14', '15']
435     for clustering in clustering_algorithms:
436         if clustering in evaluation_data.keys():
437             first=[]
438             second=[]
439             for norm in norm_order:
440                 val = evaluation_data[clustering][norm]['silhouette']
441                 if val>=-9999.0:
442                     first.append(val)
443                 else:
444                     first.append('-')
445 
446                 val = evaluation_data[clustering][norm]['gamma']
447                 if val>=-9999.0:
448                     first.append(val)
449                 else:
450                     first.append('-')
451 
452                 val = evaluation_data[clustering][norm]['db index']
453                 if val>=-9999.0:
454                     second.append(val)
455                 else:
456                     second.append('-')
457 
458                 val = evaluation_data[clustering][norm]['validity']
459                 if val>=-9999.0:
460                     second.append(val)
461                 else:
462                     second.append('-')
463             for x in first:
464                 storage.write('%s ' % x)
465             storage.write('\n')
466             for x in second:
467                 storage.write('%s ' % x)
468             storage.write('\n')
469 
470 
```

5.2.1.6 def fetch_data.generate_time (*evaluation_data*, *storage_name*)

Definition at line 471 of file fetch_data.py.

```
471 def generate_time(evaluation_data, storage_name):
472     storage = open(storage_name, ’w’)
473     clustering_algorithms = [’kmeans’, ’kmedoids’, ’AHC_single’, ’AHC_average’, ’birch’, ’dbscan’, ’optics’
      , ’sc_kmeans’, ’sc_eigen’, ’AP’, ’PCA’]
474     norm_order = [’0’, ’1’, ’2’, ’4’, ’12’, ’13’, ’14’, ’15’]
475     for clustering in clustering_algorithms:
476         if clustering in evaluation_data.keys():
477             first=[]
478             second=[]
479             for norm in norm_order:
480                 val = evaluation_data[clustering][norm][’time’]
481                 if val>=-9999.0:
482                     first.append(val)
483                 else:
484                     first.append(’-’)
485 
486             for x in first:
487                 storage.write(’%s ’ % x)
488             storage.write(’\n’)
489             for x in second:
490                 storage.write(’%s ’ % x)
491             storage.write(’\n’)
492 
493 
```

5.2.1.7 def fetch_data.get_average (*lmethod_evaluation*, *sc_eigen_evaluation*)

Definition at line 408 of file fetch_data.py.

```
408 def get_average(lmethod_evaluation, sc_eigen_evaluation):
409     average_evaluation = {}
410     for clustering in lmethod_evaluation.keys():
411         average_evaluation[clustering] = {}
412         for norm in lmethod_evaluation[clustering].keys():
413             average_evaluation[clustering][norm] = {}
414             for eval_metric in lmethod_evaluation[clustering][norm].keys():
415                 average_val = 0.0
416                 effective = 0
417                 if lmethod_evaluation[clustering][norm][eval_metric]>=-9999.0:
418                     average_val+=lmethod_evaluation[clustering][norm][eval_metric]
419                     effective+=1
420                 if sc_eigen_evaluation[clustering][norm][eval_metric]>=-9999.0:
421                     average_val+=sc_eigen_evaluation[clustering][norm][eval_metric]
422                     effective+=1
423 
424                 if effective==0:
425                     average_evaluation[clustering][norm][eval_metric] = -10000.0
426                 else:
427                     average_evaluation[clustering][norm][eval_metric]=average_val/effective
428     return average_evaluation
429 
430 
```

5.2.1.8 def fetch_data.get_distance_limit (*file_position*)

Definition at line 9 of file fetch_data.py.

```
9 def get_distance_limit(file_position):
10     with open(file_position) as f:
11         content = f.readlines()
12     # you may also want to remove whitespace characters like ‘\n‘ at the end of each line
13     content = [x.strip() for x in content]
14     distance_range = {}
15     for x in content:
16         if x!=’’:
```

```
            norm_pos = x.find('norm')
            pca_pos = x.find('PCA')

            if norm_pos==-1 and pca_pos==-1:
                continue
            if norm_pos!=-1:
                start_pos = norm_pos+5
                end_pos = start_pos
                while x[end_pos]!=' ' and x[end_pos]!=',':
                    end_pos+=1
                norm_str = x[start_pos:end_pos]
            elif pca_pos!=-1:
                norm_str = 'PCA'

            range_pos = x.find('(max - min) is')
            start_ = range_pos+len('(max - min) is')
            while x[start_]==' ':
                start_+=1
            distance_range[norm_str] = float(x[start_:])
    return distance_range


# read optimal clustering and inside you've multiple clustering algorithm
```

5.2.1.9 def fetch_data.merge_two_dicts (*first,* *second*)

Definition at line 494 of file fetch_data.py.

```
494 def merge_two_dicts(first, second):
495     result = first.copy()
496     result.update(second)
497     return result
498
499
```

5.3 generate_ap Namespace Reference

Variables

- string streamline = 'birch/0/Crayfish_full.vtk'
- string ap_13 = 'AP/Crayfish_full.vtk'
- string result = 'ap_13.vtk'
- content = f.readlines()
- ap = f.readlines()
- storage = open(result, 'w')
- bool start = False
- int count = 0
- takes = x.find('SCALARS AP_norm13 int 1')

5.3.1 Variable Documentation

5.3.1.1 list generate_ap.ap = f.readlines()

Definition at line 11 of file generate_ap.py.

5.3.1.2 string generate_ap.ap_13 = 'AP/Crayfish_full.vtk'

Definition at line 2 of file generate_ap.py.

5.3.1.3 list generate_ap.content = f.readlines()

Definition at line 6 of file generate_ap.py.

5.3.1.4 int generate_ap.count = 0

Definition at line 25 of file generate_ap.py.

5.3.1.5 string generate_ap.result = 'ap_13.vtk'

Definition at line 3 of file generate_ap.py.

5.3.1.6 tuple generate_ap.start = False

Definition at line 24 of file generate_ap.py.

5.3.1.7 generate_ap.storage = open(result, 'w')

Definition at line 15 of file generate_ap.py.

5.3.1.8 string generate_ap.streamline = 'birch/0/Crayfish_full.vtk'

Definition at line 1 of file generate_ap.py.

5.3.1.9 generate_ap.takes = x.find('SCALARS AP_norm13 int 1')

Definition at line 28 of file generate_ap.py.

5.4 time_average Namespace Reference

Functions

- def get_average_time (time_list)

Variables

- list streamlines = ['bernard_time', 'crayfish_time', 'cylinder_time', 'hurricane_time', 'solar_plume_time', 'tornado_time']
- list pathlines = ['tub_pathlines_time', 'cylinder_pathlines_time', 'blood_flow_time']

5.4.1 Function Documentation

5.4.1.1 def time_average.get_average_time (*time_list*)

Definition at line 1 of file time_average.py.

```
def get_average_time(time_list):
    data = []
    for each in time_list:
        data_set = []
        with open(each) as f:
            for line in f:
                if line!='' and line!='\n':
                    line = line.strip()
                    line = line.split()
                     each_row = []
                     for number in line:
                         if number=='-':
                             each_row.append(-10000.0)
                         else:
                             each_row.append(float(number))
                     if each_row!=[]:
                         data_set.append(each_row)
         if data_set!=[]:
             data.append(data_set)

     row = len(data[0])
     col = len(data[0][0])

     result = []
     for j in range(row):
         col_data = []
         for k in range(col):
             summation = 0
             effective = 0
             for i in range(len(time_list)):
                 if k<len(data[i][j]) and data[i][j][k]>=-9999.0:
                     summation+=data[i][j][k]
                     effective+=1
             if effective==0:
                 summation = '-'
             else:
                 summation=summation/effective

             col_data.append(summation)

         result.append(col_data)

     storage = open('average_time', 'w')
     for row in result:
         for x in row:
             storage.write('%s ' % x)
         storage.write('\n')



```

5.4.2 Variable Documentation

5.4.2.1 list time_average.pathlines = ['tub_pathlines_time', 'cylinder_pathlines_time', 'blood_flow_time']

Definition at line 53 of file time_average.py.

5.4.2.2 list time_average.streamlines = ['bernard_time', 'crayfish_time', 'cylinder_time', 'hurricane_time', 'solar_plume_time', 'tornado_time']

Definition at line 52 of file time_average.py.

Chapter 6

Class Documentation

6.1 AverageClustering Struct Reference

Public Member Functions

- AverageClustering (const float &average, const int &index)
- AverageClustering ()

Public Attributes

- float average
- int originalIndex

6.1.1 Detailed Description

Definition at line 50 of file vtk_heatmap.cpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AverageClustering::AverageClustering (const float & *average,* const int & *index*) [inline]

Definition at line 54 of file vtk_heatmap.cpp.

```
54                                                              : average(
      average), originalIndex(index)
55     {}
```

6.1.2.2 AverageClustering::AverageClustering ( ) [inline]

Definition at line 57 of file vtk_heatmap.cpp.

```
57                      : average(0.0)
58     {}
```

6.1.3 Member Data Documentation

6.1.3.1 float AverageClustering::average

Definition at line 52 of file vtk_heatmap.cpp.

6.1.3.2 int AverageClustering::originalIndex

Definition at line 53 of file vtk_heatmap.cpp.

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

- vtk_heatmap.cpp

6.2 AverageColumn Struct Reference

Public Member Functions

- AverageColumn (const float &average, const string &name)
- AverageColumn ()

Public Attributes

- float average
- string name
- std::vector< float > valueVec
- std::vector< float > std_vec

6.2.1 Detailed Description

Definition at line 37 of file vtk_heatmap.cpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AverageColumn::AverageColumn (const float & *average*, const string & *name*) [inline]

Definition at line 43 of file vtk_heatmap.cpp.

```
43                                                            : average(
      average), name(name)
44     {}
```

6.2.2.2 AverageColumn::AverageColumn ( ) [inline]

Definition at line 46 of file vtk_heatmap.cpp.

```
46                      : average(0.0)
47      {}
```

6.2.3 Member Data Documentation

6.2.3.1 float AverageColumn::average

Definition at line 39 of file vtk_heatmap.cpp.

6.2.3.2 string AverageColumn::name

Definition at line 40 of file vtk_heatmap.cpp.

6.2.3.3 std::vector<float> AverageColumn::std_vec

Definition at line 42 of file vtk_heatmap.cpp.

6.2.3.4 std::vector<float> AverageColumn::valueVec

Definition at line 41 of file vtk_heatmap.cpp.

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

- vtk_heatmap.cpp

6.3 BestValue Struct Reference

Public Member Functions

- BestValue (const float &value)
- BestValue ()

Public Attributes

- float value
- int i
- int j

6.3.1 Detailed Description

Definition at line 62 of file vtk_heatmap.cpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BestValue::BestValue (const float & *value*) [inline]

Definition at line 67 of file vtk_heatmap.cpp.

```
67                                    : value(value), i(-1), j(-1)
68      {}
```

6.3.2.2 BestValue::BestValue ( ) [inline]

Definition at line 70 of file vtk_heatmap.cpp.

```
70                  : value(0.0), i(-1), j(-1)
71      {}
```

6.3.3 Member Data Documentation

6.3.3.1 int BestValue::i

Definition at line 65 of file vtk_heatmap.cpp.

6.3.3.2 int BestValue::j

Definition at line 65 of file vtk_heatmap.cpp.

6.3.3.3 float BestValue::value

Definition at line 64 of file vtk_heatmap.cpp.

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

- vtk_heatmap.cpp

6.4 Dataset Struct Reference

Public Attributes

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

6.4.1 Detailed Description

Definition at line 18 of file readDistRange.cpp.

6.4.2 Member Data Documentation

6.4.2.1 Eigen::MatrixXf Dataset::dataMatrix

Definition at line 21 of file readDistRange.cpp.

6.4.2.2 string Dataset::dataName

Definition at line 28 of file readDistRange.cpp.

6.4.2.3 vector<vector<float> > Dataset::dataVec

Definition at line 20 of file readDistRange.cpp.

6.4.2.4 int Dataset::dimension = -1

Definition at line 24 of file readDistRange.cpp.

6.4.2.5 string Dataset::fullName

Definition at line 27 of file readDistRange.cpp.

6.4.2.6 int Dataset::maxElements = -1

Definition at line 22 of file readDistRange.cpp.

6.4.2.7 string Dataset::strName

Definition at line 26 of file readDistRange.cpp.

6.4.2.8 int Dataset::vertexCount = -1

Definition at line 23 of file readDistRange.cpp.

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

- readDistRange.cpp

6.5 PathlinePoint Struct Reference

Public Member Functions

- PathlinePoint ()

Public Attributes

- float coordinates [3]
- float time

6.5.1 Detailed Description

Definition at line 36 of file pathlineInterpolation.cpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 PathlinePoint::PathlinePoint ( ) [inline]

Definition at line 41 of file pathlineInterpolation.cpp.

```
41                      : time(-1)
42     {
43         for(int i=0; i<3; ++i)
44             coordinates[i] = -1.0;
45     }
```

6.5.3 Member Data Documentation

6.5.3.1 float PathlinePoint::coordinates[3]

Definition at line 38 of file pathlineInterpolation.cpp.

6.5.3.2 float PathlinePoint::time

Definition at line 39 of file pathlineInterpolation.cpp.

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

- pathlineInterpolation.cpp

Chapter 7

File Documentation

7.1 average_ranking.py File Reference

Namespaces

- average_ranking

Variables

- list average_ranking.silhouette = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'AHC-single', 'BIRCH', 'AP', 'DBSCAN', 'OPTICS', 'SC-eigen']
- list average_ranking.gamma = ['AHC-average', 'PCA', 'kmeans', 'kmedoids', 'SC-kmeans', 'BIRCH', 'DBSC←↩AN', 'AHC-single', 'OPTICS', 'SC-eigen', 'AP']
- list average_ranking.dbindex = ['PCA', 'kmeans', 'AHC-single', 'kmedoids', 'AHC-average', 'BIRCH', 'SC-eigen', 'SC-kmeans', 'DBSCAN', 'OPTICS', 'AP']
- list average_ranking.validity = ['DBSCAN', 'PCA', 'AHC-single', 'SC-kmeans', 'BIRCH', 'AHC-average', 'kmeans', 'kmedoids', 'AP', 'OPTICS', 'SC-eigen']
- list average_ranking.sil_norm = ['d_R', 'd_S', 'd_P', 'd_E', 'd_M', 'd_F', 'd_H', 'd_G']
- list average_ranking.gamma_norm = ['d_R', 'd_E', 'd_H', 'd_S', 'd_M', 'd_F', 'd_G', 'd_P']
- list average_ranking.db_norm = ['d_G', 'd_M', 'd_S', 'd_H', 'd_E', 'd_P', 'd_F', 'd_R']
- list average_ranking.validity_norm = ['d_M', 'd_R', 'd_P', 'd_F', 'd_H', 'd_S', 'd_E', 'd_G']
- dictionary average_ranking.average_ranking = {clustering:0 for clustering in silhouette}
- dictionary average_ranking.average_norm = {norm:0 for norm in sil_norm}
- list average_ranking.order = [100, None]

7.2 fetch_data.py File Reference

Namespaces

- fetch_data

Functions

- def fetch_data.get_distance_limit (file_position)
- def fetch_data.extract_evaluation_data (distance_range, data_folder)
- def fetch_data.extract_single_readme (distance_range, data_folder)
- def fetch_data.extract_norm_readme (distance_range, data_folder)
- def fetch_data.get_average (lmethod_evaluation, sc_eigen_evaluation)
- def fetch_data.generate_text (evaluation_data, storage_name)
- def fetch_data.generate_time (evaluation_data, storage_name)
- def fetch_data.merge_two_dicts (first, second)
- def fetch_data.extract_full_data ()

7.3 generate_ap.py File Reference

Namespaces

- generate_ap

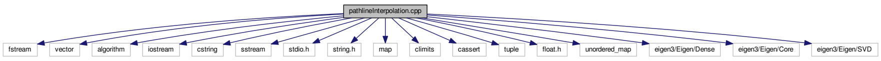
Variables

- string generate_ap.streamline = 'birch/0/Crayfish_full.vtk'
- string generate_ap.ap_13 = 'AP/Crayfish_full.vtk'
- string generate_ap.result = 'ap_13.vtk'
- generate_ap.content = f.readlines()
- generate_ap.ap = f.readlines()
- generate_ap.storage = open(result, 'w')
- bool generate_ap.start = False
- int generate_ap.count = 0
- generate_ap.takes = x.find('SCALARS AP_norm13 int 1')

7.4 pathlineInterpolation.cpp File Reference

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

Include dependency graph for pathlineInterpolation.cpp:



Classes

- struct PathlinePoint

Macros

- #define MULTIPLIER 8.0

Functions

- void readPathlineRaw (const char ∗fileName, std::vector< std::vector< PathlinePoint > > &pathlines, std::tuple< float, float, float > &timeRange)
- void performInterpolation (const std::vector< std::vector< PathlinePoint > > &pathlines, std::vector< Eigen::VectorXf > &interpolatedLine, const std::tuple< float, float, float > &timeRange)
- void generateLineFile (const std::vector< Eigen::VectorXf > &interpolatedLine, const char ∗fileName)
- int main (int argc, char ∗argv[ ])

7.4.1 Macro Definition Documentation

7.4.1.1 #define MULTIPLIER 8.0

Definition at line 33 of file pathlineInterpolation.cpp.

7.4.2 Function Documentation

7.4.2.1 void generateLineFile (const std::vector< Eigen::VectorXf > & *interpolatedLine,* const char ∗ *fileName*)

Definition at line 314 of file pathlineInterpolation.cpp.

```
{
    std::ofstream fout(fileName, ios::out);
    if(fout.fail())
    {
        std::cout << "Error for creating ascii file in the folder!" << std::endl;
        exit(1);
    }
    // get the how many pathlines
    const int& lineSize = interpolatedLine.size();
    Eigen::VectorXf pathlineData;
    for(int i=0; i<lineSize; ++i)
    {
        pathlineData = interpolatedLine[i];

        for(int j=0; j<pathlineData.size(); ++j)
        {
            fout << pathlineData(j) << " ";
        }
        fout << std::endl;
    }

    fout.close();
}
```

7.4.2.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 59 of file pathlineInterpolation.cpp.

```
{
    if(argc!=2)
    {
        std::cout << "Error for argument count. Should be ./executable fileName" << std::endl;
        exit(1);
    }

    // get the proper file name
    auto pos = string(argv[1]).find(".vtk");
    if(pos==std::string::npos)
    {
        std::cout << "Input file is not a .vtk file!" << std::endl;
        exit(1);
    }

    std::vector<std::vector<PathlinePoint> > pathlineRaw;
    std::tuple<float,float,float> timeRange;
    // read raw pathlines with coordinates and time from .vtk file
    readPathlineRaw(argv[1], pathlineRaw, timeRange);

    // interpolate the pathlines so that points of different frames for pathlines reside the exactly same time slide
    std::vector<Eigen::VectorXf> interpolatedLine;
    performInterpolation(pathlineRaw, interpolatedLine, timeRange);

    // write into txt file for clustering evaluation of blood flow
    generateLineFile(interpolatedLine, string(argv[1]).substr(0,pos).c_str());

    return 0;
}
```

7.4.2.3 void performInterpolation (const std::vector< std::vector< PathlinePoint > > & *pathlines,* std::vector< Eigen::VectorXf > & *interpolatedLine,* const std::tuple< float, float, float > & *timeRange*)

Definition at line 246 of file pathlineInterpolation.cpp.

```
{
    // use the time information of timeRange to perform time-based sampling for the pathlines
    const float& starting = std::get<0>(timeRange);
    const float& ending = std::get<1>(timeRange);
    const float& aveSlice = MULTIPLIER*std::get<2>(timeRange);
    interpolatedLine.resize(pathlines.size());

#pragma omp parallel for schedule(static) num_threads(8)
    for(int i=0; i<pathlines.size(); ++i)
    {
        // for each pathlines, will interpolate it from 0, 1, 2, ...., currentEndingTime
        const std::vector<PathlinePoint>& line = pathlines[i];
        std::map<float, Eigen::Vector3f> timeCoordinates;
        for(int j=0; j<line.size(); ++j)
        {
            timeCoordinates[line[j].time] = Eigen::Vector3f(line[j].coordinates[0], line[j].coordinates[1],
                    line[j].coordinates[2]);
        }
        int numOfPoints = int((line.back().time-starting)/(aveSlice))+1;

        Eigen::VectorXf& lineCoordinate = interpolatedLine[i];
        lineCoordinate = Eigen::VectorXf(3*numOfPoints);

        float current;
        for(int j=0; j<numOfPoints; ++j)
        {
            current = starting + aveSlice*j;

            // before the occuring time, should be directly repeating the first point
            if(current<=line[0].time)
            {
                for(int k=0; k<3; ++k)
                {
                    lineCoordinate(3*j+k) = line[0].coordinates[k];
                }
            }
            // this time has been recorded in the map, directly load the data
            else if(timeCoordinates.find(current)!=timeCoordinates.end())
            {
                for(int k=0; k<3; ++k)
                {
                    lineCoordinate(3*j+k) = timeCoordinates[current](k);
                }
            }
            // else, find the left and right time step and perform linear interpolation
            else
            {
                // find the clipping left and right time slices for interpolation
                auto right = timeCoordinates.upper_bound(current);
                auto left = std::prev(right);

                float ratio = (current-left->first)/(right->first-left->first);
                Eigen::Vector3f currentPoint = (1.0-ratio)*left->second + ratio*right->second;
                for(int k=0; k<3; ++k)
                {
                    lineCoordinate(3*j+k) = currentPoint(k);
                }
            }
        }
    }
}
```

7.4.2.4 void readPathlineRaw (const char * *fileName*, std::vector< std::vector< PathlinePoint > > & *pathlines*, std::tuple< float, float, float > & *timeRange*)

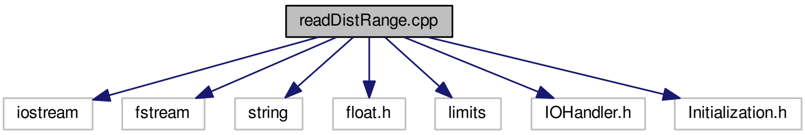
Definition at line 92 of file pathlineInterpolation.cpp.

```
{
    std::get<0>(timeRange) = FLT_MAX;
    std::get<1>(timeRange) = -1.0;

    std::vector<PathlinePoint> pointCoordinateVec;

    std::ifstream fin(fileName, ios::in);

    if(fin.fail())
    {
        std::cout << "Error for opening file contents!" << std::endl;
        exit(1);
    }

    string line;

    // ignore the header file of vtk
    for(int i=0; i<5; ++i)
        getline(fin, line);

    stringstream ss(line);
    ss >> line, ss >> line; // get number of points

    ss.clear();
    ss.str("");

    // extract the number of points
    const int& numOfPoints = std::atoi(line.c_str());
    std::cout << "There are " << numOfPoints << " points read from the file!" << std::endl;

    // assign the memory for numOfPoints
    pointCoordinateVec.resize(numOfPoints);

    // start traversal for point coordinates
    int index = 0, separation = 0;
    while(getline(fin, line) && line.size()!=0)
    {
        ss.str(line);
        separation = 0;

        // read the one line of points into the cache
        while(ss>>line)
        {
            pointCoordinateVec[index].coordinates[separation] = std::atof(line.c_str());
            if(separation == 2)
            {
                separation = 0;
                ++index;
            }
            else
                ++separation;
        }
        ss.clear();
        ss.str("");
    }

    assert(index == numOfPoints);

    // read the number of lines from the txt file
    for(int i=0; i<8; ++i)
        getline(fin, line);

    ss.str(line);
    ss >> line, ss >> line;
    const int& numOfLines = std::atoi(line.c_str());
    std::cout << "There are " << numOfLines << " pathlines read from the file!" << std::endl;
    ss.clear();
    ss.str("");

    // store the line to point arrays, e.g., one line has which indices of points
    std::vector<vector<int> > pointsToLine(numOfLines);

    int numOfPointsForLine, pointIndex;
    for(int i=0; i<numOfLines; ++i)
    {
        getline(fin, line);
        ss.str(line);
        ss >> line;

        // find how many points this line contain
        numOfPointsForLine = std::atoi(line.c_str());

        std::vector<int>& currentLine = pointsToLine[i];
        currentLine.resize(numOfPointsForLine);

        for(int j=0; j<numOfPointsForLine; ++j)
        {
            ss >> line;
            pointIndex = std::atoi(line.c_str());
            currentLine[j] = pointIndex;
        }

        ss.clear();
        ss.str("");
    }

    while(getline(fin, line))
    {
        ss.str(line);
        ss >> line;
        ss.clear();
        ss.str("");
        if(strcmp(line.c_str(), "time")==0)
            break;
    }

    index = 0;
    while(getline(fin, line) && line.size())
    {
        ss.str(line);
        while(ss >> line)
        {
            pointCoordinateVec[index].time = std::atof(line.c_str());
            ++index;
        }
        ss.clear();
        ss.str("");
    }
    assert(index == numOfPoints);
    fin.close();

    std::cout << "File content traversal completed!" << std::endl;

    pathlines.resize(numOfLines);
    std::vector<int> lineIndex;

    float averageSlice = 0.0;
    for(int i=0; i<numOfLines; ++i)
    {
        std::vector<PathlinePoint>& currentLine = pathlines[i];
        lineIndex = pointsToLine[i];
        currentLine.resize(lineIndex.size());

        for(int j=0; j<lineIndex.size(); ++j)
        {
            currentLine[j] = pointCoordinateVec[lineIndex[j]];
        }
        std::get<0>(timeRange) = std::min(std::get<0>(timeRange), currentLine[0].time);
        std::get<1>(timeRange) = std::max(std::get<1>(timeRange), currentLine.back().time);

        averageSlice += (currentLine.back().time-currentLine[0].time)/float(lineIndex.size()-1);
        std::cout << "Pathline " << i << " has starting time " << currentLine[0].time <<
        " and ending time " << currentLine.back().time << std::endl;
    }
    pointCoordinateVec.clear();
    std::get<2>(timeRange) = averageSlice/float(numOfLines);

    std::cout << "Starting time is " << std::get<0>(timeRange) << ", ending time is " << std::get<1>(
  timeRange) <<
    ", and average time slice is " << std::get<2>(timeRange) << std::endl;
}
```

7.5 readDistRange.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <float.h>
#include <limits>
#include "IOHandler.h"
#include "Initialization.h"
```

Include dependency graph for readDistRange.cpp:



Classes

- struct Dataset

Functions

- void getDistRange (const Dataset &ds)
- void setDataset (Dataset &ds, const int &argc, char ∗∗argv)
- int main (int argc, char ∗argv[ ])

7.5.1 Function Documentation

7.5.1.1 void getDistRange (const Dataset & *ds*)

Definition at line 53 of file readDistRange.cpp.

```
{
    for(int i=0; i<16; ++i)
    {
        if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
            continue;
        /* very hard to decide whether needed to perform such pre-processing */
        MetricPreparation object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
        object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(), ds.dataMatrix.cols(), i);

        deleteDistanceMatrix(ds.dataMatrix.rows());
        getDistanceMatrix(ds.dataMatrix, i, object);

        const int& Row = ds.dataMatrix.rows();
        float min_dist = numeric_limits<float>::max(), max_dist = numeric_limits<float>::min();

    #pragma omp parallel for reduction(min:min_dist) num_threads(8)
        for (int i = 0; i < Row; ++i)
        {
            for (int j = 0; j < Row; ++j)
            {
                if(i==j)
                    continue;
                min_dist = std::min(min_dist, distanceMatrix[i][j]);
            }
        }

    #pragma omp parallel for reduction(max:max_dist) num_threads(8)
        for (int i = 0; i < Row; ++i)
        {
            for (int j = 0; j < Row; ++j)
            {
                if(i==j)
                    continue;
                max_dist = std::max(max_dist, distanceMatrix[i][j]);
            }
        }

        std::cout << "norm " << i << " has min " << min_dist << " and max " << max_dist << std::endl;

        std::ofstream readme("../dataset/dist_range", ios::app | ios::out);
        if(readme.fail())
        {
            std::cout << "Error for opening readme!" << std::endl;
            exit(1);
        }

        readme << "For norm " << i << ", min is " << min_dist << ", max is " << max_dist << ", and (max - min) is " <<
                (max_dist-min_dist) << std::endl;
        readme << std::endl;

        readme.close();
    }
}
```

7.5.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 37 of file readDistRange.cpp.

```
{
    if(argc!=3)
    {
        std::cout << "parameter option is not right!" << std::endl;
        exit(1);
    }

    Dataset ds;
    setDataset(ds, argc, argv);

    getDistRange(ds);

    return 0;
}
```

7.5.1.3 void setDataset (Dataset & *ds*, const int & *argc*, char ∗∗ *argv*)

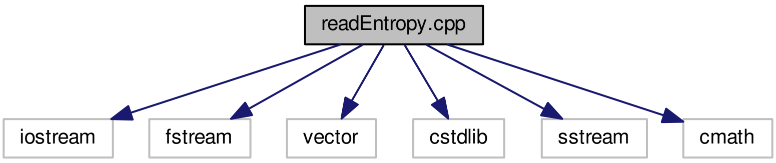
Definition at line 109 of file readDistRange.cpp.

```
{
    if(argc!=3)
    {
        std::cout << "Input argument should have 3!" << endl
                  << "./cluster inputFile_name(in dataset folder) "
                  << "data_dimension(3)" << endl;
        exit(1);
    }
    ds.strName = string("../dataset/")+string(argv[1]);
    ds.dataName = string(argv[1]);
    ds.dimension = atoi(argv[2]);

    int pathlineOption;
    std::cout << "It is a pathline dataset? 1.Yes, 0.No" << std::endl;
    std::cin >> pathlineOption;
    assert(pathlineOption==1||pathlineOption==0);
    bool isPathlines = (pathlineOption==1);

    int sampleOption;

    if(isPathlines)
        sampleOption = 1;

    else
    {
        std::cout << "choose a sampling method for the dataset?" << std::endl
                  << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
        std::cin >> sampleOption;
        assert(sampleOption==1||sampleOption==2);
    }

    IOHandler::readFile(ds.strName,ds.dataVec,ds.
  vertexCount,ds.dimension,ds.maxElements);

    ds.fullName = ds.strName+"_full.vtk";
    IOHandler::printVTK(ds.fullName, ds.dataVec, ds.vertexCount, ds.
  dimension);

    if(sampleOption==1)
        IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
  dimension,ds.maxElements);
    else if(sampleOption==2)
        IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.
  dimension,ds.maxElements);
}
```

7.6 readEntropy.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
#include <sstream>
#include <cmath>
```

Include dependency graph for readEntropy.cpp:



Functions

- void readFile (std::vector< int > &groupSize, const char ∗fileName)
- void computeEntropy (const std::vector< int > &groupSize)
- int main (int argc, char ∗argv[ ])

7.6.1 Function Documentation

7.6.1.1 void computeEntropy (const std::vector< int > & *groupSize*)

Definition at line 56 of file readEntropy.cpp.

```
{
    int total = 0;
    for(int i=0;i<groupSize.size();++i)
        total+=groupSize[i];

    float prob, result = 0.0;
    for(int i=0;i<groupSize.size();++i)
    {
        prob = float(groupSize[i])/float(total);
        result+=prob*log2f(prob);
    }

    result = -result/log2f(groupSize.size());
    std::cout << "Entropy is " << result << std::endl;
}
```

7.6.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 14 of file readEntropy.cpp.

```
{
    if(argc!=2)
    {
        std::cout << "Error for argument input!" << std::endl;
        exit(1);
    }

    std::vector<int> storage;
    readFile(storage, argv[1]);

    computeEntropy(storage);

    return 0;
}
```

7.6.1.3 void readFile (std::vector< int > & *groupSize,* const char ∗ *fileName*)

Definition at line 31 of file readEntropy.cpp.

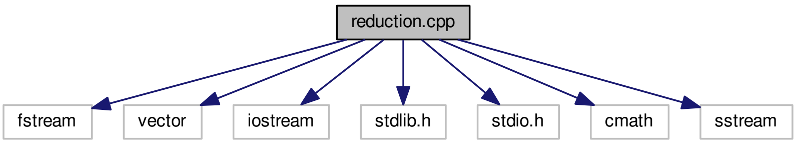
```
{
    ifstream fin(fileName, ios::in);
    if(!fin)
    {
        std::cout << "Error for reading a file!" << std::endl;
        exit(1);
    }

    string line;

    int num;
    while(getline(fin,line))
    {
        num = 0;
        stringstream ss(line);
        while(ss>>line)
            ++num;
        if(num > 0)
            groupSize.push_back(num);
    }

    fin.close();
}
```

7.7 README.md File Reference

7.8 reduction.cpp File Reference

```
#include <fstream>
#include <vector>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <cmath>
#include <sstream>
```

Include dependency graph for reduction.cpp:



Functions

- void readFile (const char ∗fileName, std::vector< std::vector< float > > &dataVec, const int &lineNumber, const int &vertexNumber)
- void writeFile (const char ∗fileName, const std::vector< std::vector< float > > &dataVec)
- int main (int argc, char ∗argv[ ])

7.8.1 Function Documentation

7.8.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 21 of file reduction.cpp.

```
22 {
23     if(argc!=3)
24     {
25         std::cout << "Error for argument input! Should be ./main originalFile newFile " << std::endl;
26         exit(1);
27     }
28 
29 
30     std::vector<std::vector<float> > dataVec;
31     const char* fileName =  argv[1];
32 
33     std::cout << "input the number among which one is chosen for streamlines? " << std::endl;
34     int lineNumber;
35     std::cin >> lineNumber;
36 
37     std::cout << "input the number among which one is chosen for vertex? " << std::endl;
38     int vertexNumber;
39     std::cin >> vertexNumber;
40 
41     readFile(fileName, dataVec, lineNumber, vertexNumber);
42     writeFile(argv[2], dataVec);
43     return 0;
44 }
```

7.8.1.2 void readFile (const char ∗ *fileName*, std::vector< std::vector< float > > & *dataVec*, const int & *lineNumber*, const int & *vertexNumber*)

Definition at line 47 of file reduction.cpp.

```
{
    std::ifstream fin(fileName, ios::in);
    if(!fin)
    {
        std::cout << "Error creating files!" << std::endl;
        exit(1);
    }
    stringstream ss;
    std::vector<float> tempVec;

    string line, part;

    std::vector<float> vec(3);
    float temp;

    int lineTag = 0;
    while(getline(fin, line) /* && currentNumber < MAXNUMBER*/)
    {
        //currentDimensions = 0;
        if(lineTag==1)
        {
            lineTag = (lineTag+1)%lineNumber;
            continue;
        }

        int tag = 0, count = 0;
        bool isNext = false;
        ss.str(line);
        while(ss>>part /*&& currentDimensions<3*MAXDIMENSION*/)
        {
            /* operations below would remove duplicate vertices because that would damage our computation */
            if(tag>=3)
            {
                isNext = !isNext;
                tag = (tag+1)%(vertexNumber*3);
                continue;
            }
            temp = atof(part.c_str());
            if(isNext)
            {
                if(count<3)
                {
                    vec[count] = temp;
                    tag = (tag+1)%(vertexNumber*3);
                    ++count;
                }
                if(count==3)
                {
                    int size = tempVec.size();
                    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))
                    {
                        tempVec.push_back(vec[0]);
                        tempVec.push_back(vec[1]);
                        tempVec.push_back(vec[2]);
                    }
                    count = 0;
                }
                continue;
            }
            tempVec.push_back(temp);
            tag = (tag+1)%(vertexNumber*3);
            //currentDimensions++;
        }
        /* accept only streamlines with at least three vertices */
        if(tempVec.size()/3>2)
        {
            dataVec.push_back(tempVec);
        }
        tempVec.clear();
        ss.clear();
        ss.str("");
        //currentNumber++;

        lineTag = (lineTag+1)%lineNumber;
    }
    fin.close();
127
128     std::cout << "Finished reading file!" << std::endl;
129 }
```

7.8.1.3 void writeFile (const char ∗ *fileName,* const std::vector< std::vector< float > > & *dataVec*)

Definition at line 131 of file reduction.cpp.

```
133 {
134     std::ofstream fout(fileName, ios::out);
135     if(!fout)
136     {
137         std::cout << "Cannot create a file!" << std::endl;
138         exit(1);
139     }
140
141     const int& vecSize = dataVec.size();
142     std::vector<float> tempVec;
143     int tempVecSize;
144
145     for (int i = 0; i < vecSize; ++i)
146     {
147         tempVec = dataVec[i];
148         tempVecSize = tempVec.size();
149         for (int j = 0; j < tempVecSize; ++j)
150         {
151             fout << tempVec[j] << " ";
152         }
153         fout << std::endl;
154     }
155     fout.close();
156
157     std::cout << "Finished writing file!" << std::endl;
158 }
```

7.9 time_average.py File Reference

Namespaces

- time_average

Functions

- def time_average.get_average_time (time_list)

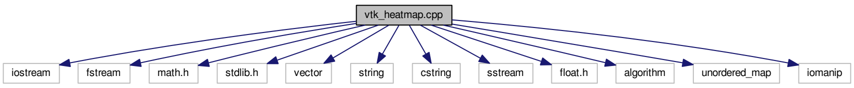
Variables

- list time_average.streamlines = ['bernard_time', 'crayfish_time', 'cylinder_time', 'hurricane_time', 'solar_↩ plume_time', 'tornado_time']
- list time_average.pathlines = ['tub_pathlines_time', 'cylinder_pathlines_time', 'blood_flow_time']

7.10 vtk_heatmap.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include <vector>
#include <string>
#include <cstring>
#include <sstream>
#include <float.h>
#include <algorithm>
#include <unordered_map>
#include <iomanip>
```

Include dependency graph for vtk_heatmap.cpp:



Classes

- struct AverageColumn
- struct AverageClustering
- struct BestValue

Functions

- void readData (std::vector< std::vector< float > > &dataVec, const char ∗fileName)
- void createHeatMap (const std::vector< std::vector< float > > &dataVec)
- int make_index (const int &i, const int &j, const int &col)
- void create_assemble (const std::vector< std::vector< float > > &dataVec)
- void create_separate (const std::vector< std::vector< float > > &dataVec)
- void create_ranking (const std::vector< std::vector< float > > &dataVec)
- void create_latex_table (const std::vector< std::vector< float > > &dataVec)
- void get_average_value (std::vector< std::vector< float > > &averageValue, std::vector< std::vector< float > > &standardDeviation, string file_list[ ], const int &file_size)
- void create_std_ranking (const std::vector< std::vector< float > > &averageValue, const std::vector< std↩ ::vector< float > > &standardDeviation, const int &file_size)
- int main (int argc, char ∗argv[ ])

Variables

- const float & range_start = 0.1
- const float & max_db_index = 5.0
- float data_range [4][3]

7.10.1 Function Documentation

7.10.1.1 void create_assemble (const std::vector< std::vector< float > > & *dataVec*)

Definition at line 335 of file vtk_heatmap.cpp.

```
{
    /* get the limit range of four scalar values */
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();

    float value;
    int num;
    /* generate the assembled four-scalar normalized for R visualization */
    std::ofstream normalized("assembled", ios::out);
    if(normalized.fail())
    {
        std::cout << "Error for creating wrong files!" << std::endl;
        exit(1);
    }

    string rownames[] = {"d_E", "d_E_", "d_F", "d_F_", "d_G", "d_G_", "d_R", "d_R_", "d_M", "d_M_", "d_H", "d_H_", "d_S", "d_S_", "d_P", "d_P_",
                         "d_T", "d_T"};
    for (int i = 0; i < cols; ++i)
    {
        normalized << rownames[i] << "\t";
        for (int j = 0; j < rows;++j)
        {
            if(dataVec[j][i]<=-9999.0)
            {
                value = 0.0;
                normalized << value << "\t";
                continue;
            }
            num = (j%2)*2+i%2;
            if(num<=1)
                value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            else if(num==2)
            {
                if(dataVec[j][i]>=max_db_index)
                    value = 1.0;
                else
                    value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            }
            else if(num==3)
                value = (log10(dataVec[j][i])-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;

            if(num==0||num==1)
                normalized << value << "\t";
            else if(num==2||num==3)
                normalized << (1.0+range_start)-value << "\t";
        }
        normalized << std::endl;
    }

    normalized.close();
}
```

7.10.1.2 void create_latex_table (const std::vector< std::vector< float > > & *dataVec*)

Definition at line 713 of file vtk_heatmap.cpp.

```
{
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();
    std::ofstream latex_table("latex_table", ios::out);
    if(latex_table.fail())
    {
        std::cout << "Error for creating latex table w.r.t. data vec!" << std::endl;
        exit(1);
    }
```

```

  /* get the best respective value */
  string clustering[] = {"\\textbf{K-means}", "\\textbf{K-medoids}", "\\textbf{AHC}-single", "\\textbf{AHC}-average", "\\textbf{BIRCH}",
                          "\\textbf{DBSCAN}", "\\textbf{OPTICS}", "\\textbf{SC}-kmeans", "\\textbf{SC}-eigen", "\\textbf{AP}", "\\textbf{PCA}"};
  BestValue ***best_value = new BestValue**[cols/2];
  for (int i = 0; i < cols/2; ++i)
  {
      best_value[i] = new BestValue*[2];
      for(int j=0; j<2; ++j)
      {
          if(j==0)
          {
              best_value[i][j] = new BestValue[2];
              best_value[i][j][0].value = -10000.0;
              best_value[i][j][1].value = -10000.0;
          }
          else if(j==1)
          {
              best_value[i][j] = new BestValue[2];
              best_value[i][j][0].value = FLT_MAX;
              best_value[i][j][1].value = FLT_MAX;
          }
      }
  }

  int col_num, index;
  for (int i = 0; i < cols; ++i)
  {
      col_num = i/2;
      index = i%2;
      for (int j = 0; j < rows; j+=2)
      {
          if (dataVec[j][i]<=-9999.0)
              continue;

          if (dataVec[j][i]>best_value[col_num][0][index].value)
          {
              best_value[col_num][0][index].value = dataVec[j][i];
              best_value[col_num][0][index].i = i;
              best_value[col_num][0][index].j = j;
          }
      }

      for (int j = 1; j < rows; j+=2)
      {
          if (dataVec[j][i]<=-9999.0)
              continue;

          if (dataVec[j][i]<best_value[col_num][1][index].value)
          {
              best_value[col_num][1][index].value = dataVec[j][i];
              best_value[col_num][1][index].i = i;
              best_value[col_num][1][index].j = j;
          }
      }
  }


  string tag[] = {"\\dashuline", "\\underline", "*", "\\textbf"};

  for (int j = 0; j<rows; ++j)
  {
      latex_table << "\\multirow{2}{*}{" << clustering[j/2] << "} ";
      for (int i = 0; i < cols; ++i)
      {
          latex_table << "&";
          if (i==best_value[i/2][0][i%2].i && j==best_value[i/2][0][i%2].j)
              latex_table << tag[i%2] << "{";

          if (dataVec[j][i]<=-9999.0)
              latex_table << "-";
          else
              latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
          if (i==best_value[i/2][0][i%2].i && j==best_value[i/2][0][i%2].j)
              latex_table << "}";
      }
      latex_table << "\\\\" << std::endl;

      j+=1;
      for (int i = 0; i < cols; ++i)
      {
          if(i%2==0)
          {
              latex_table << "& \\multicolumn{1}{l}{";

```

```
                    if (dataVec[j][i]<=-9999.0)
                        latex_table << "\\hspace{0.23cm}" << "-";
                    else if (dataVec[j][i]>=0.01 && dataVec[j][i]<1000)
                        latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
                    else
                        latex_table << std::scientific << std::setprecision(1) << dataVec[j][i];
                    if (i==best_value[i/2][1][0].i && j==best_value[i/2][1][0].j)
                        latex_table << tag[2];

                    latex_table << "}";
                }
                else
                {
                    latex_table << "&";
                    if (i==best_value[i/2][1][1].i && j==best_value[i/2][1][1].j)
                        latex_table << tag[3] << "{";

                    if (dataVec[j][i]<=-9999.0)
                        latex_table << "-";
                    else if (dataVec[j][i]>=0.01)
                        latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
                    else
                        latex_table << std::scientific << std::setprecision(1) << dataVec[j][i];
                    if (i==best_value[i/2][1][1].i && j==best_value[i/2][1][1].j)
                        latex_table << "}";
                }
            }
            latex_table << "\\\\" << std::endl;

            latex_table << "\\hline" << std::endl;
        }

        latex_table.close();

}
```

7.10.1.3 void create_ranking (const std::vector< std::vector< float > > & *dataVec*)

Definition at line 488 of file vtk_heatmap.cpp.

```
{
    /* get the limit range of four scalar values */
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();

    float value;
    int num;

    unordered_map<int, std::vector<AverageColumn> > rankMap;

    for (int i = 0; i < 4; ++i)
    {
        rankMap.insert(make_pair(i, std::vector<AverageColumn>()));
    }

    string metric[]={"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};

    int effective[2];
    for (int i = 0; i < cols; ++i)
    {
        AverageColumn ac[2];
        ac[0].name = ac[1].name = metric[i/2];
        effective[0] = effective[1] = 0;
        for (int j = 0; j < rows;j+=2)
        {
            num = i%2;
            ac[0].valueVec.push_back(dataVec[j][i]);
            if (dataVec[j][i]>-9999.0)
            {
                ++effective[0];
                ac[0].average+=dataVec[j][i];
            }

            num = i%2+2;
            ac[1].valueVec.push_back(dataVec[j+1][i]);
            if (dataVec[j+1][i]>-9999.0)
            {
                ++effective[1];
                ac[1].average+=dataVec[j+1][i];
            }
```

```
      }
      ac[0].average/=effective[0];
      ac[1].average/=effective[1];

      rankMap[i%2].push_back(ac[0]);
      rankMap[i%2+2].push_back(ac[1]);
  }

  string clustering[]={"K-means", "K-medoids", "AHC-single", "AHC-average", "BIRCH", "DBSCAN", "OPTICS", "SC-kmeans", "SC-eigen", "AP", "PCA"};
  unordered_map<int, std::vector<AverageClustering> > clusteringMap;
  for (int j = 0; j < rows; ++j)
  {
      AverageClustering ac[2];
      ac[0].originalIndex = ac[1].originalIndex = j/2;
      effective[0] = effective[1] = 0;
      for (int i = 0; i < cols; ++i)
      {
          if (dataVec[j][i]>-9999.0)
          {
              ++effective[i%2];
              ac[i%2].average+=dataVec[j][i];
          }
      }
      ac[0].average/=effective[0];
      ac[1].average/=effective[1];
      clusteringMap[(j%2)*2].push_back(ac[0]);
      clusteringMap[(j%2)*2+1].push_back(ac[1]);
  }


  for (int i = 0; i < 4; ++i)
  {
      /* ranking silhouette and gamma from largest to smallest */
      if(i<=1)
      {
          std::sort(rankMap[i].begin(), rankMap[i].end(), [](const AverageColumn& a, const AverageColumn& b)
          {return a.average>b.average;});

          std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const AverageClustering& a, const AverageClustering& b)
          {return a.average>b.average;});
      }
      /* ranking db index and validity from smallest to largest */
      else
      {
          std::sort(rankMap[i].begin(), rankMap[i].end(), [](const AverageColumn& a, const AverageColumn& b)
          {return a.average<b.average;});
          std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const AverageClustering& a, const AverageClustering& b)
          {return a.average<b.average;});
      }
  }
  std::unordered_map<int, std::vector<int> > verticalOrder;
  for (int i = 0; i < 4; ++i)
  {
      verticalOrder[i] = std::vector<int>(clusteringMap[i].size());
      for (int j=0; j<clusteringMap[i].size(); ++j)
      {
          verticalOrder[i][j] = clusteringMap[i][j].originalIndex;
      }
  }

  float realValue;
  /* generate the assembled four-scalar normalized for R visualization */
  string file_names[] = {"silhouette_ranking", "gamma_ranking", "dbindex_ranking", "validity_ranking"};

  string best_names[] = {"silhouette_best", "gamma_best", "dbindex_best", "validity_best"};

  BestValue best_value[4];
  best_value[0].value = best_value[1].value = -FLT_MAX;
  best_value[2].value = best_value[3].value = FLT_MAX;

  std::cout << "Ranking started..." << std::endl;
  for (int i = 0; i < 4; ++i)
  {
      std::ofstream colnames((to_string(i)+"_colnames").c_str(), ios::out);
      if(colnames.fail())
      {
          std::cout << "Error for creating files!" << std::endl;
          exit(1);
      }

      std::cout << file_names[i] << std::endl;
      for (int j = 0; j < verticalOrder[i].size(); ++j)
      {
          std::cout << "(" << clustering[verticalOrder[i][j]] << "," << clusteringMap[i][j].average << "), ";;
          colnames << clustering[verticalOrder[i][j]] << " ";
      }
      colnames << std::endl;
      colnames.close();

      std::ofstream ranked_file(file_names[i].c_str(), ios::out);
      if (ranked_file.fail())
      {
          std::cout << "Error for creating file!" << std::endl;
          exit(1);
      }

      const std::vector<AverageColumn>& element = rankMap[i];
      for (int j = 0; j < element.size(); ++j)
      {
          ranked_file << element[j].name << "\t";
          for (int k = 0; k<element[j].valueVec.size(); ++k)
          {

              realValue = (element[j].valueVec)[verticalOrder[i][k]];

              if(realValue<=-9999.0)
              {
                  ranked_file << 0.0 << "\t";
                  continue;
              }
              else
              {
                  if (i<=1)
                  {
                      if(realValue>=best_value[i].value)
                      {
                          best_value[i].value=realValue;
                          best_value[i].i = k;
                          best_value[i].j = j;
                      }
                  }
                  else if (i>=2)
                  {
                      if(realValue<=best_value[i].value)
                      {
                          best_value[i].value=realValue;
                          best_value[i].i = k;
                          best_value[i].j = j;
                      }
                  }
              }

              if(i<=1)
                  value = (realValue-data_range[i][0])/data_range[i][2]*(1.0-range_start)+range_start;
              else if(i==2)
              {
                  if(realValue>=max_db_index)
                      value = 1.0;
                  else
                      value = (realValue-data_range[i][0])/data_range[i][2]*(1.0-range_start)+range_start;
              }
              else
                  value = (log10(realValue)-data_range[i][0])/data_range[i][2]*(1.0-range_start)+range_start;

              if(i<=1)
                  ranked_file << value << "\t";
              else
                  ranked_file << (1.0+range_start)-value << "\t";
          }
          ranked_file << std::endl;
      }
      std::cout << std::endl;
      ranked_file.close();
  }

  std::cout << std::endl;
  for (int i = 0; i < 4; ++i)
  {
      std::ofstream best_file(best_names[i].c_str(), ios::out);
      if(best_file.fail())
      {
          std::cout << "Error for creating files!" << std::endl;
          exit(1);
      }

```

```
        const std::vector<AverageColumn>& element = rankMap[i];
        std::cout << best_value[i].value << std::endl;
        for (int j = 0; j < element.size(); ++j)
        {
            best_file << element[j].name << "\t";
            for (int k = 0; k < element[j].valueVec.size(); ++k)
            {
                if (best_value[i].i==k && best_value[i].j==j)
                    best_file << 1.0 << "\t";
                else
                    best_file << -1.0 << "\t";
            }
            best_file << std::endl;
        }
        best_file.close();
    }
}
```

7.10.1.4 void create_separate (const std::vector< std::vector< float > > & *dataVec*)

Definition at line 388 of file vtk_heatmap.cpp.

```
{
    /* get the limit range of four scalar values */
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();

    float value;
    int num;
    /* generate the assembled four-scalar normalized for R visualization */
    std::ofstream silouette("Silhouette", ios::out), similarity("Gamma", ios::out), dbindex("DBindex", ios::out), validity("Validity", ios::out);
    if(silouette.fail() || similarity.fail() || dbindex.fail() || validity.fail())
    {
        std::cout << "Error for creating wrong files!" << std::endl;
        exit(1);
    }

    string metric[] = {"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};
    for (int i = 0; i < cols; ++i)
    {
        if(i%2==0)
        {
            silouette << metric[i/2] << "\t";
            dbindex << metric[i/2] << "\t";
        }
        else
        {
            similarity << metric[i/2] << "\t";
            validity << metric[i/2] << "\t";
        }
        for (int j = 0; j < rows;++j)
        {
            num = (j%2)*2+i%2;
            if(dataVec[j][i]<=-9999.0)
            {
                value = 0.0;
                switch(num)
                {
                    case 0:
                        silouette << value << "\t";
                        break;

                    case 1:
                        similarity << value << "\t";
                        break;

                    case 2:
                        dbindex << value << "\t";
                        break;

                    case 3:
                        validity << value << "\t";
                        break;
                }

                continue;
            }
            if(num<=1)
```

```
                value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            else if(num==2)
            {
                if(dataVec[j][i]>=max_db_index)
                    value = 1.0;
                else
                    value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            }
            else if(num==3)
                value = (log10(dataVec[j][i])-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;

            if(num==0||num==1)
            {
                if(num==0)
                    silouette << value << "\t";
                else if(num==1)
                    similarity << value << "\t";
            }
            else if(num==2||num==3)
            {
                if(num==2)
                    dbindex << (1.0+range_start)-value << "\t";
                else if(num==3)
                    validity << (1.0+range_start)-value << "\t";
            }
        }
        if(i%2==0)
        {
            silouette << std::endl;
            dbindex << std::endl;
        }
        else
        {
            similarity << std::endl;
            validity << std::endl;
        }
    }
    silouette.close();
    similarity.close();
    dbindex.close();
    validity << std::endl;
}
```

7.10.1.5 void create_std_ranking (const std::vector< std::vector< float > > & *averageValue*, const std::vector< std::vector< float > > & *standardDeviation*, const int & *file_size*)

Definition at line 908 of file vtk_heatmap.cpp.

```
{
    /* get the limit range of four scalar values */
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();

    float value;
    int num;

    float value_option[] = {FLT_MAX, -FLT_MAX, 0};
    for (int i = 0; i < 3; ++i)
    {
        for (int j=0;j<4;++j)
            data_range[j][i] = value_option[i];
    }

    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            if(dataVec[i][j]<=-9999.0)
                continue;
            num = (i%2)*2+j%2;
            data_range[num][0] = std::min(data_range[num][0], dataVec[i][j]);
            data_range[num][1] = std::max(data_range[num][1], dataVec[i][j]);

        }
    }
    for (int i = 0; i < 4; ++i)
    {
        std::cout << data_range[i][0] << " " << data_range[i][1] << std::endl;
        if(i==2)
        {
            if(data_range[i][1]>=max_db_index)
                data_range[i][1]=max_db_index;
        }
        else if(i==3)
        {
            data_range[i][0] = log10(data_range[i][0]);
            data_range[i][1] = log10(data_range[i][1]);
        }
        data_range[i][2] = data_range[i][1]-data_range[i][0];
    }


    unordered_map<int, std::vector<AverageColumn> > rankMap;

    for (int i = 0; i < 4; ++i)
    {
        rankMap.insert(make_pair(i, std::vector<AverageColumn>()));
    }

    string metric[]={"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};

    int effective[2];
    for (int i = 0; i < cols; ++i)
    {
        AverageColumn ac[2];
        ac[0].name = ac[1].name = metric[i/2];
        effective[0] = effective[1] = 0;
        for (int j = 0; j < rows;j+=2)
        {
            num = i%2;
            ac[0].valueVec.push_back(dataVec[j][i]);
            if(file_size>=4)
                ac[0].std_vec.push_back(standardDeviation[j][i]);
            if (dataVec[j][i]>-9999.0)
            {
                ++effective[0];
                ac[0].average+=dataVec[j][i];
            }

            num = i%2+2;
            ac[1].valueVec.push_back(dataVec[j+1][i]);
            if(file_size>=4)
                ac[1].std_vec.push_back(standardDeviation[j+1][i]);
            if (dataVec[j+1][i]>-9999.0)
            {
                ++effective[1];
                ac[1].average+=dataVec[j+1][i];
            }
        }
        ac[0].average/=effective[0];
        ac[1].average/=effective[1];

        rankMap[i%2].push_back(ac[0]);
        rankMap[i%2+2].push_back(ac[1]);
    }

    string clustering[]={"K-means", "K-medoids", "AHC-single", "AHC-average", "BIRCH", "DBSCAN", "OPTICS", "SC-kmeans", "SC-eigen", "AP", "PCA"};
    unordered_map<int, std::vector<AverageClustering> > clusteringMap;
    for (int j = 0; j < rows; ++j)
    {
        AverageClustering ac[2];
        ac[0].originalIndex = ac[1].originalIndex = j/2;
        effective[0] = effective[1] = 0;
        for (int i = 0; i < cols; ++i)
        {
            if (dataVec[j][i]>-9999.0)
            {
                ++effective[i%2];
                ac[i%2].average+=dataVec[j][i];
            }
        }
        ac[0].average/=effective[0];
        ac[1].average/=effective[1];
        clusteringMap[(j%2)*2].push_back(ac[0]);
        clusteringMap[(j%2)*2+1].push_back(ac[1]);
    }

    for (int i = 0; i < 4; ++i)
    {
        /* ranking silhouette and gamma from largest to smallest */
        if(i<=1)
        {
            std::sort(rankMap[i].begin(), rankMap[i].end(), [](const AverageColumn& a, const AverageColumn& b)
            {return a.average>b.average;});

            std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const AverageClustering& a, const AverageClustering& b)
            {return a.average>b.average;});
        }
        /* ranking db index and validity from smallest to largest */
        else
        {
            std::sort(rankMap[i].begin(), rankMap[i].end(), [](const AverageColumn& a, const AverageColumn& b)
            {return a.average<b.average;});
            std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const AverageClustering& a, const AverageClustering& b)
            {return a.average<b.average;});
        }
    }
    std::unordered_map<int, std::vector<int> > verticalOrder;
    for (int i = 0; i < 4; ++i)
    {
        verticalOrder[i] = std::vector<int>(clusteringMap[i].size());
        for (int j=0; j<clusteringMap[i].size(); ++j)
        {
            verticalOrder[i][j] = clusteringMap[i][j].originalIndex;
        }
    }

    float realValue;
    /* generate the assembled four-scalar normalized for R visualization */
    string file_names[] = {"silhouette_ranking", "gamma_ranking", "dbindex_ranking", "validity_ranking"};

    string best_names[] = {"silhouette_best", "gamma_best", "dbindex_best", "validity_best"};

    BestValue best_value[4];
    best_value[0].value = best_value[1].value = -FLT_MAX;
    best_value[2].value = best_value[3].value = FLT_MAX;

    for (int i = 0; i < 4; ++i)
    {
        std::ofstream colnames((to_string(i)+"_colnames").c_str(), ios::out);
        if(colnames.fail())
        {
            std::cout << "Error for creating files!" << std::endl;
            exit(1);
        }
        for (int j = 0; j < verticalOrder[i].size(); ++j)
        {
            colnames << clustering[verticalOrder[i][j]] << " ";
        }
        colnames << std::endl;
        colnames.close();

        std::ofstream ranked_file(file_names[i].c_str(), ios::out);
        if (ranked_file.fail())
        {
            std::cout << "Error for creating file!" << std::endl;
            exit(1);
        }

        const std::vector<AverageColumn>& element = rankMap[i];
        for (int j = 0; j < element.size(); ++j)
        {
            ranked_file << element[j].name << "\t";
            for (int k = 0; k<element[j].valueVec.size(); ++k)
            {
                realValue = (element[j].valueVec)[verticalOrder[i][k]];
                if(realValue<=-9999.0)
                {
                    ranked_file << 0.0 << "\t";
                    continue;
                }
                else
                {
                    if (i<=1)
                    {
                        if(realValue>=best_value[i].value)
                        {
                            best_value[i].value=realValue;
                            best_value[i].i = k;
                            best_value[i].j = j;
                        }
                    }
                    else if (i>=2)
                    {
                        if(realValue<=best_value[i].value)
                        {
                            best_value[i].value=realValue;
                            best_value[i].i = k;
                            best_value[i].j = j;
                        }
                    }
                }

                if(i<=1)
                    value = (realValue-data_range[i][0])/data_range[i][2]*(1.0-
range_start)+range_start;
                if(i==2)
                {
                    if(realValue>=max_db_index)
                        value = 1.0;
                    else
                        value = (realValue-data_range[i][0])/
data_range[i][2]*(1.0-range_start)+range_start;
                }
                else if(i==3)
                    value = (log10(realValue)-data_range[i][0])/
data_range[i][2]*(1.0-range_start)+range_start;

                if(i<=1)
                    ranked_file << value << "\t";
                else
                    ranked_file << (1.0+range_start)-value << "\t";
            }
            ranked_file << std::endl;
        }
        ranked_file.close();
    }


    std::cout << std::endl;
    for (int i = 0; i < 4; ++i)
    {
        std::ofstream best_file(best_names[i].c_str(), ios::out);
        if(best_file.fail())
        {
            std::cout << "Error for creating files!" << std::endl;
            exit(1);
        }

        const std::vector<AverageColumn>& element = rankMap[i];
        for (int j = 0; j < element.size(); ++j)
        {
            best_file << element[j].name << "\t";
            for (int k = 0; k < element[j].valueVec.size(); ++k)
            {
                if (best_value[i].i==k && best_value[i].j==j)
                    best_file << 1.0 << "\t";
                else
                    best_file << -1.0 << "\t";
            }
            best_file << std::endl;
        }
        best_file.close();
    }

    if(file_size>=4)
    {

        float std_range[4][3]=
        {
            FLT_MAX, -FLT_MAX, 0,
            FLT_MAX, -FLT_MAX, 0,
            FLT_MAX, -FLT_MAX, 0,
            FLT_MAX, -FLT_MAX, 0
        };

        for (int i = 0; i < rows; ++i)
        {
            for (int j = 0; j < cols; ++j)
            {
                if(standardDeviation[i][j]<=-9999.0)
                    continue;
                num = (i%2)*2+j%2;
                std_range[num][0] = std::min(std_range[num][0], standardDeviation[i][j]);
                std_range[num][1] = std::max(std_range[num][1], standardDeviation[i][j]);

            }
        }
        for (int i = 0; i < 4; ++i)
        {
            std_range[i][2] = std_range[i][1]-std_range[i][0];
            std::cout << "[" << std_range[i][0] << "," << std_range[i][1] << "]: " << std_range[i][2] <<
```

```
 std::endl;
        }


        string std_names[] = {"silhouette_std", "gamma_std", "dbindex_std", "validity_std"};

        for (int i = 0; i < 4; ++i)
        {
            std::ofstream ranked_file(std_names[i].c_str(), ios::out);
            if (ranked_file.fail())
            {
                std::cout << "Error for creating file!" << std::endl;
                exit(1);
            }

            const std::vector<AverageColumn>& element = rankMap[i];
            for (int j = 0; j < element.size(); ++j)
            {
                ranked_file << element[j].name << "\t";
                for (int k = 0; k<element[j].std_vec.size(); ++k)
                {
                    realValue = (element[j].std_vec)[verticalOrder[i][k]];
                    if(realValue<=-9999.0)
                    {
                        ranked_file << 0.0 << "\t";
                        continue;
                    }
                    value = (realValue-std_range[i][0])/std_range[i][2]*(1.0-
 range_start)+range_start;

                    ranked_file << value << "\t";
                }
                ranked_file << std::endl;
            }
            ranked_file.close();
        }
    }

}
```

7.10.1.6 void createHeatMap (const std::vector< std::vector< float > > & *dataVec*)

Definition at line 186 of file vtk_heatmap.cpp.

```
{
    /* get the limit range of four scalar values */
    const int& rows = dataVec.size();
    const int& cols = dataVec[0].size();

    for(int i=0; i<4; ++i)
    {
        data_range[i][0] = FLT_MAX;
        data_range[i][1] = -FLT_MAX;
        data_range[i][2] = 0;
    }

    float value;
    int num;
    int nonZero = 0;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            if(dataVec[i][j]<=-9999.0)
                continue;
            num = (i%2)*2+j%2;
            ++nonZero;
            data_range[num][0] = std::min(data_range[num][0], dataVec[i][j]);
            data_range[num][1] = std::max(data_range[num][1], dataVec[i][j]);

        }
    }
    for (int i = 0; i < 4; ++i)
    {
        std::cout << data_range[i][0] << " " << data_range[i][1] << std::endl;
        if(i==3)
        {
            data_range[i][0] = log10(data_range[i][0]);
            data_range[i][1] = log10(data_range[i][1]);
        }
        else if(i==2)
        {
            if(data_range[i][1]>=max_db_index)
                data_range[i][1]=max_db_index;
        }
        data_range[i][2] = data_range[i][1]-data_range[i][0];
    }

    /* generate heatmap values */
    std::ofstream ofs("heatmap.vtk", ios::out);
    if(ofs.fail())
    {
        std::cout << "Error for creating vtk file!" << std::endl;
        exit(1);
    }
    ofs << "# vtk DataFile Version 3.0\n"
        << "matrix_vis" << "\n"
        << "ASCII\n\n"
        << "DATASET POLYDATA\n";
    ofs << "POINTS " << (rows+1)*(cols+1) << " float\n";
    const float& x_step = 0.1;
    const float& y_step = 0.1;
    for (int j = 0; j < rows+1; ++j)
    {
        for (int i = 0; i < cols+1; ++i)
        {
            ofs << i*x_step << " " << j*y_step << " " << 0 << std::endl;
        }
    }
    ofs << "POLYGONS " << nonZero << " " << 5*nonZero << "\n";
    int x, y;
    for (int j = rows-1; j >=0; --j)
    {
        for (int i = 0; i<cols; ++i)
        {
            if(dataVec[j][i]<=-9999.0)
                continue;
            ofs << 4 << " " << make_index(i,rows-1-j,cols+1) << " " << make_index(i+1,rows-1-j,cols+1) << " "
                << make_index(i+1,rows-j,cols+1) << " " << make_index(i,rows-j,cols+1) << std::endl;
        }
    }
    ofs << "CELL_DATA " << nonZero << "\n" << "SCALARS " << "label" << " float 1\n" << "LOOKUP_TABLE default\n";
    for (int j = rows-1; j >=0; --j)
    {
        for (int i = 0; i < cols; ++i)
        {
            if(dataVec[j][i]<=-9999.0)
                continue;
            num = (j%2)*2+i%2;

            if(num<=1)
            {
                value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            }
            else if(num==3)
            {
                value = (log10(dataVec[j][i])-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            }
            else if(num==2)
            {
                if(dataVec[j][i]>=max_db_index)
                    value = 1.0;
                else
                    value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-range_start)+range_start;
            }
            if(num==0 || num==1)
                ofs << value << std::endl;
            else
                ofs << (1.0+range_start)-value << std::endl;

        }
    }
    ofs.close();

    /* generate boundary grids for 2X2 */
    std::ofstream grid("grid.vtk", ios::out);
    if(grid.fail())
    {
        std::cout << "Error for creating file!" << std::endl;
        exit(1);
    }
    grid << "# vtk DataFile Version 3.0\n"
        << "matrix_vis" << "\n"
        << "ASCII\n\n"
        << "DATASET POLYDATA\n";
    grid << "POINTS " << (rows/2+1)*(cols/2+1) << " float\n";
    for (int j = 0; j < rows/2+1; ++j)
    {
        for (int i = 0; i < cols/2+1; ++i)
        {
            grid << i*2.0*x_step << " " << j*2.0*y_step << " " << 0 << std::endl;
        }
    }
    const int& line_number = rows/2*(cols/2+1)+(rows/2+1)*cols/2;
    grid << "LINES " << line_number << " " << 3*line_number << std::endl;
    for (int j = 0; j < rows/2; ++j)
    {
        for (int i = 0; i < cols/2; ++i)
        {
            grid << 2 << " " << make_index(i,j,cols/2+1) << " " <<
  make_index(i+1,j,cols/2+1) << std::endl;
            grid << 2 << " " << make_index(i,j,cols/2+1) << " " <<
  make_index(i,j+1,cols/2+1) << std::endl;
        }
        grid << 2 << " " << make_index(cols/2,j,cols/2+1) << " " <<
  make_index(cols/2,j+1,cols/2+1) << std::endl;
    }
    for (int i = 0; i < cols/2; ++i)
    {
        grid << 2 << " " << make_index(i,rows/2,cols/2+1) << " " <<
  make_index(i+1,rows/2,cols/2+1) << std::endl;
    }
    grid.close();

}
```

7.10.1.7 void get_average_value (std::vector< std::vector< float > > & *averageValue*, std::vector< std::vector< float > > & *standardDeviation*, string *file_list[ ]*, const int & *file_size*)

Definition at line 845 of file vtk_heatmap.cpp.

```
{
    std::vector<std::vector<float> > totalValue[file_size];

    for (int i = 0; i < file_size; ++i)
    {
        readData(totalValue[i], filenames[i].c_str());
    }

    const int& rows = totalValue[0].size();
    const int& cols = totalValue[0][0].size();

    averageValue = std::vector< std::vector<float> >(rows, std::vector<float>(cols));

    if(file_size>=4)
        standardDeviation = std::vector< std::vector<float> >(rows, std::vector<float>(cols));

    float average, stdeviation, value;
    int effective;
    for (int i = 0; i < cols; ++i)
    {
        for (int j = 0; j < rows; ++j)
        {
            average = stdeviation = 0.0;
            effective = 0;
            for (int k = 0; k < file_size; ++k)
            {
                value = totalValue[k][j][i];
                if (value<=-9999.0)
                    continue;
                average+=value;
                if(file_size>=4)
                    stdeviation+=value*value;
                ++effective;
            }
            if(effective==0)
            {
                averageValue[j][i] = -10000.0;
                if(file_size>=4)
                    standardDeviation[j][i] = -10000.0;
            }
            else
            {
                average/=float(effective);
                averageValue[j][i] = average;
                if(file_size>=4)
                {
                    stdeviation = stdeviation/float(effective)-average*average;
                    if(stdeviation<0)
                    {
                        std::cout << "Error for one-pass standard deviation computation!" << std::endl;
                        exit(1);
                    }
                    standardDeviation[j][i] = sqrt(stdeviation);
                }
            }
        }
    }

}
```

7.10.1.8 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 96 of file vtk_heatmap.cpp.

```
{
    // if(argc!=2)
    // {
    //  std::cout << "Error for argument count!" << std::endl;
    //  exit(1);
    // }

    // std::vector<std::vector<float> > dataVec;

    // readData(dataVec, argv[1]);

    // createHeatMap(dataVec);

    // create_assemble(dataVec);

    // //create_separate(dataVec);

    // create_ranking(dataVec);

    // create_latex_table(dataVec);


    if(argc!=1)
    {
        std::cout << "Get average and std so no need for argument!" << std::endl;
        exit(1);
    }
    std::vector<std::vector<float> > averageValue, standardDeviation;

    //string filenames[] = {"bernard_evaluation", "crayfish_evaluation", "cylinder_evaluation", "hurricane_evaluation", "solar_plume_evaluation", "tornado_evaluation"};

    string filenames[] = {"cylinder_pathlines_evaluation", "tub_pathlines_evaluation", "blood_flow_evaluation"};

    get_average_value(averageValue, standardDeviation, filenames, sizeof(filenames)/sizeof(string));

    create_std_ranking(averageValue, standardDeviation, sizeof(filenames)/sizeof(string));

    create_assemble(averageValue);

    create_latex_table(averageValue);

    createHeatMap(averageValue);

    return 0;
}
```

7.10.1.9 int make_index (const int & *i*, const int & *j*, const int & *col*)

Definition at line 180 of file vtk_heatmap.cpp.

```
{
    return j*col+i;
}
```

7.10.1.10 void readData (std::vector< std::vector< float > > & *dataVec*, const char * *fileName*)

Definition at line 144 of file vtk_heatmap.cpp.

```
{
    std::ifstream fin(fileName, ios::in);
    if(fin.fail())
    {
        std::cout << "Error for reading data from existing file!" << std::endl;
        exit(1);
    }

    stringstream ss;
    string line;
    std::vector<float> row;
    while(getline(fin, line))
    {
        ss.str(std::string());
        ss.clear();
        ss << line;
        while(ss>>line)
        {
            if(strcmp(line.c_str(), "-")==0)
            {
                row.push_back(-10000.0);
            }
            else
            {
                row.push_back(std::atof(line.c_str()));
            }
        }
        dataVec.push_back(row);
        row.clear();
    }

    fin.close();
}
```

7.10.2 Variable Documentation

7.10.2.1 float data_range[4][3]

Initial value:

```
=
{
    FLT_MAX, -FLT_MAX, 0,
    FLT_MAX, -FLT_MAX, 0,
    FLT_MAX, -FLT_MAX, 0,
    FLT_MAX, -FLT_MAX, 0
}
```

Definition at line 29 of file vtk_heatmap.cpp.

7.10.2.2 const float& max_db_index = 5.0

Definition at line 19 of file vtk_heatmap.cpp.

7.10.2.3 const float& range_start = 0.1

Definition at line 16 of file vtk_heatmap.cpp.

Index