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':
```

```
114                         evaluation[norm]['0']['gamma'] = float(x[start_pos:end_pos])
115                     else:
116                         evaluation[d_folder][norm]['gamma'] = float(x[start_pos:end_pos])
117 
118                 if dbindex_pos!=-1:
119                     start_pos = dbindex_pos+len('DB index is:')+1
120                     while x[start_pos]==' ':
121                         start_pos+=1
122                     end_pos=start_pos
123                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
124                         end_pos+=1
125                     val_str = x[start_pos:end_pos]
126                     if val_str !='-nan' and val_str !='inf':
127                         if norm=='PCA':
128                             evaluation[norm]['0']['db index'] = float(x[start_pos:end_pos])
129                         else:
130                             evaluation[d_folder][norm]['db index'] = float(x[start_pos:end_pos])
131                     norm_found = False
132 
133                 if validity_pos!=-1:
134                     start_pos = validity_pos+len('measure is:')+1
135                     while x[start_pos]==' ':
136                         start_pos+=1
137                     end_pos=start_pos
138                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
139                         end_pos+=1
140                     val_str = x[start_pos:end_pos]
141                     if val_str !='-nan' and val_str !='inf':
142                         if norm=='PCA':
143                             evaluation[norm]['0']['validity'] = float(x[start_pos:end_pos])/distance_range[
    norm]
144                         else:
145                             evaluation[d_folder][norm]['validity'] = float(x[start_pos:end_pos])/
    distance_range[norm]
146 
147                 elif measurement_pos!=-1:
148                     start_pos = measurement_pos+len('measurement is:')+1
149                     while x[start_pos]==' ':
150                         start_pos+=1
151                     end_pos=start_pos
152                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
153                         end_pos+=1
154                     val_str = x[start_pos:end_pos]
155                     if val_str !='-nan' and val_str !='inf':
156                         if norm=='PCA':
157                             evaluation[norm]['0']['validity'] = float(x[start_pos:end_pos])/distance_range[
    norm]
158                         else:
159                             evaluation[d_folder][norm]['validity'] = float(x[start_pos:end_pos])/
    distance_range[norm]
160 
161                 pca_time_tag = x.find('PCA+K_Means operation takes:')
162                 kmeans_time_tag = x.find('K-means on norm')
163                 kmedoid_time_tag = x.find('Direct K_Means operation time for norm')
164 
165                 if pca_time_tag!=-1 or kmeans_time_tag!=-1 or kmedoid_time_tag!=-1:
166                     takes = x.find('takes:')
167                     if takes==-1:
168                         raise ValueError('Error for time search!')
169                     start_pos = takes+len('takes:')
170                     while x[start_pos]==' ':
171                         start_pos+=1
172                     end_pos=start_pos
173                     while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' '
    and x[end_pos]!='s':
174                         end_pos+=1
175                     val_str = x[start_pos:end_pos]
176                     if norm=='PCA':
177                         evaluation[norm]['0']['time'] = float(x[start_pos:end_pos])
178                     else:
179                         evaluation[d_folder][norm]['time'] = float(x[start_pos:end_pos])
180                 else:
181                     takes = x.find('takes:')
182                     if takes!=-1:
183                         start_pos = takes+len('takes:')
184                         while x[start_pos]==' ':
185                             start_pos+=1
186                         end_pos=start_pos
187                         while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!='
    ' and x[end_pos]!='s':
188                             end_pos+=1
189                         val_str = x[start_pos:end_pos]
190                         if evaluation[d_folder][norm]['time']<=-9999.0:
191                             evaluation[d_folder][norm]['time'] = float(x[start_pos:end_pos])
192                         else:
193                             evaluation[d_folder][norm]['time'] += float(x[start_pos:end_pos])
194 
```

```
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
```

```
339                 end_pos=start_pos
340                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
341                     end_pos+=1
342                 val_str = x[start_pos:end_pos]
343                 if val_str !='-nan' and val_str !='inf':
344                     evaluation[data_folder][norm]['silhouette'] = float(x[start_pos:end_pos])
345 
346             if gamma_pos!=-1:
347                 start_pos = gamma_pos+len('statistic is:')+1
348                 while x[start_pos]==' ':
349                     start_pos+=1
350                 end_pos=start_pos
351                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
352                     end_pos+=1
353                 val_str = x[start_pos:end_pos]
354                 if val_str !='-nan' and val_str !='inf':
355                     evaluation[data_folder][norm]['gamma'] = float(x[start_pos:end_pos])
356 
357             if dbindex_pos!=-1:
358                 start_pos = dbindex_pos+len('DB index is:')+1
359                 while x[start_pos]==' ':
360                     start_pos+=1
361                 end_pos=start_pos
362                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
363                     end_pos+=1
364                 val_str = x[start_pos:end_pos]
365                 if val_str !='-nan' and val_str !='inf':
366                     evaluation[data_folder][norm]['db index'] = float(x[start_pos:end_pos])
367                 norm_found = False
368 
369             if validity_pos!=-1:
370                 start_pos = validity_pos+len('measure is:')+1
371                 while x[start_pos]==' ':
372                     start_pos+=1
373                 end_pos=start_pos
374                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
375                     end_pos+=1
376                 val_str = x[start_pos:end_pos]
377                 if val_str !='-nan' and val_str !='inf':
378                     evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[
    norm]
379 
380             elif measurement_pos!=-1:
381                 start_pos = measurement_pos+len('measurement is:')+1
382                 while x[start_pos]==' ':
383                     start_pos+=1
384                 end_pos=start_pos
385                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
386                     end_pos+=1
387                 val_str = x[start_pos:end_pos]
388                 if val_str !='-nan' and val_str !='inf':
389                     evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[
    norm]
390 
391             takes = x.find('takes:')
392             if takes!=-1:
393                 start_pos = takes+len('takes:')
394                 while x[start_pos]==' ':
395                     start_pos+=1
396                 end_pos=start_pos
397                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ' and x[
    end_pos]!='s':
398                     end_pos+=1
399                 val_str = x[start_pos:end_pos]
400                 if evaluation[data_folder][norm]['time']<=-9999.0:
401                     evaluation[data_folder][norm]['time'] = float(x[start_pos:end_pos])
402                 else:
403                     evaluation[data_folder][norm]['time'] += float(x[start_pos:end_pos])
404 
405     return evaluation
406 
407 
```

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']
```

```
202     for val in norm_list:
203         evaluation[data_folder][val] = {'silhouette':-10000.0, 'gamma':-10000.0, 'db index':-10000.0, '
    validity':-10000.0, 'time':-10000.0}
204 
205     readme = data_folder+'/README'
206     with open(readme) as r:
207         content = r.readlines()
208     norm_found = False
209     norm=None
210 
211     norm_found = False
212     for x in content:
213         if x=='' or x=='\n':
214             continue
215         if norm_found is False:
216             norm_pos = x.find('norm')
217             Norm_pos = x.find('Norm:')
218             if norm_pos==-1 and Norm_pos==-1:
219                 continue
220 
221             if norm_pos!=-1:
222                 end_pos = norm_pos+5
223                 while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
224                     end_pos+=1
225                 norm = x[norm_pos+5:end_pos]
226             elif Norm_pos!=-1:
227                 end_pos = Norm_pos+6
228                 while(x[end_pos]!=' ' and end_pos<=len(x)-1) and x[end_pos]!='\n':
229                     end_pos+=1
230                 norm = x[norm_pos+6:end_pos]
231             norm_found = True
232 
233         if norm_found is True and norm in norm_list:
234             sil_pos = x.find('silhouette:')
235             gamma_pos = x.find('statistic is:')
236             dbindex_pos = x.find('DB index is:')
237             validity_pos = x.find('measure is:')
238             measurement_pos = x.find('measurement is:')
239 
240             if sil_pos!=-1:
241                 start_pos = sil_pos+len('silhouette:')+1
242                 while x[start_pos]==' ':
243                     start_pos+=1
244                 end_pos=start_pos
245                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
246                     end_pos+=1
247                 val_str = x[start_pos:end_pos]
248                 if val_str !='-nan' and val_str !='inf':
249                     evaluation[data_folder][norm]['silhouette'] = float(x[start_pos:end_pos])
250 
251             if gamma_pos!=-1:
252                 start_pos = gamma_pos+len('statistic is:')+1
253                 while x[start_pos]==' ':
254                     start_pos+=1
255                 end_pos=start_pos
256                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
257                     end_pos+=1
258                 val_str = x[start_pos:end_pos]
259                 if val_str !='-nan' and val_str !='inf':
260                     evaluation[data_folder][norm]['gamma'] = float(x[start_pos:end_pos])
261 
262             if dbindex_pos!=-1:
263                 start_pos = dbindex_pos+len('DB index is:')+1
264                 while x[start_pos]==' ':
265                     start_pos+=1
266                 end_pos=start_pos
267                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
268                     end_pos+=1
269                 val_str = x[start_pos:end_pos]
270                 if val_str !='-nan' and val_str !='inf':
271                     evaluation[data_folder][norm]['db index'] = float(x[start_pos:end_pos])
272                 norm_found = False
273 
274             if validity_pos!=-1:
275                 start_pos = validity_pos+len('measure is:')+1
276                 while x[start_pos]==' ':
277                     start_pos+=1
278                 end_pos=start_pos
279                 while x[end_pos]!=',' and x[end_pos]!='\n' and end_pos<=len(x)-1 and x[end_pos]!=' ':
280                     end_pos+=1
281                 val_str = x[start_pos:end_pos]
282                 if val_str !='-nan' and val_str !='inf':
283                     evaluation[data_folder][norm]['validity'] = float(x[start_pos:end_pos])/distance_range[
    norm]
284 
285             elif measurement_pos!=-1:
286                 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!='':
```

```
17             norm_pos = x.find('norm')
18             pca_pos = x.find('PCA')
19 
20             if norm_pos==-1 and pca_pos==-1:
21                 continue
22             if norm_pos!=-1:
23                 start_pos = norm_pos+5
24                 end_pos = start_pos
25                 while x[end_pos]!=' ' and x[end_pos]!=',':
26                     end_pos+=1
27                 norm_str = x[start_pos:end_pos]
28             elif pca_pos!=-1:
29                 norm_str = 'PCA'
30 
31             range_pos = x.find('(max - min) is')
32             start_ = range_pos+len('(max - min) is')
33             while x[start_]==' ':
34                 start_+=1
35             distance_range[norm_str] = float(x[start_:])
36     return distance_range
37 
38 
39 # 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.

```
1 def get_average_time(time_list):
2     data = []
3     for each in time_list:
4         data_set = []
5         with open(each) as f:
6             for line in f:
7                 if line!='' and line!='\n':
8                     line = line.strip()
9                     line = line.split()
10                     each_row = []
11                     for number in line:
12                         if number=='-':
13                             each_row.append(-10000.0)
14                         else:
15                             each_row.append(float(number))
16                     if each_row!=[]:
17                         data_set.append(each_row)
18         if data_set!=[]:
19             data.append(data_set)
20
21     row = len(data[0])
22     col = len(data[0][0])
23
24     result = []
25     for j in range(row):
26         col_data = []
27         for k in range(col):
28             summation = 0
29             effective = 0
30             for i in range(len(time_list)):
31                 if k<len(data[i][j]) and data[i][j][k]>=-9999.0:
32                     summation+=data[i][j][k]
33                     effective+=1
34             if effective==0:
35                 summation = '-'
36             else:
37                 summation=summation/effective
38
39             col_data.append(summation)
40
41         result.append(col_data)
42
43     storage = open('average_time', 'w')
44     for row in result:
45         for x in row:
46             storage.write('%s ' % x)
47         storage.write('\n')
48
49
50
```

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.

```
315 {
316     std::ofstream fout(fileName, ios::out);
317     if(fout.fail())
318     {
319         std::cout << "Error for creating ascii file in the folder!" << std::endl;
320         exit(1);
321     }
322     // get the how many pathlines
323     const int& lineSize = interpolatedLine.size();
324     Eigen::VectorXf pathlineData;
325     for(int i=0; i<lineSize; ++i)
326     {
327         pathlineData = interpolatedLine[i];
328 
329         for(int j=0; j<pathlineData.size(); ++j)
330         {
331             fout << pathlineData(j) << " ";
332         }
333         fout << std::endl;
334     }
335 
336     fout.close();
337 }
```

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

Definition at line 59 of file pathlineInterpolation.cpp.

```
60 {
61     if(argc!=2)
62     {
63         std::cout << "Error for argument count. Should be ./executable fileName" << std::endl;
64         exit(1);
65     }
66
67     // get the proper file name
68     auto pos = string(argv[1]).find(".vtk");
69     if(pos==std::string::npos)
70     {
71         std::cout << "Input file is not a .vtk file!" << std::endl;
72         exit(1);
73     }
74
75     std::vector<std::vector<PathlinePoint> > pathlineRaw;
76     std::tuple<float,float,float> timeRange;
77     // read raw pathlines with coordinates and time from .vtk file
78     readPathlineRaw(argv[1], pathlineRaw, timeRange);
79
80     // interpolate the pathlines so that points of different frames for pathlines reside the exactly same
       time slide
81     std::vector<Eigen::VectorXf> interpolatedLine;
82     performInterpolation(pathlineRaw, interpolatedLine, timeRange);
83
84     // write into txt file for clustering evaluation of blood flow
85     generateLineFile(interpolatedLine, string(argv[1]).substr(0,pos).c_str());
86
87     return 0;
88 }
```

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.

```
248 {
249     // use the time information of timeRange to perform time-based sampling for the pathlines
250     const float& starting = std::get<0>(timeRange);
251     const float& ending = std::get<1>(timeRange);
252     const float& aveSlice = MULTIPLIER*std::get<2>(timeRange);
253     interpolatedLine.resize(pathlines.size());
254
255 #pragma omp parallel for schedule(static) num_threads(8)
256     for(int i=0; i<pathlines.size(); ++i)
257     {
258         // for each pathlines, will interpolate it from 0, 1, 2, ...., currentEndingTime
259         const std::vector<PathlinePoint>& line = pathlines[i];
260         std::map<float, Eigen::Vector3f> timeCoordinates;
261         for(int j=0; j<line.size(); ++j)
262         {
263             timeCoordinates[line[j].time] = Eigen::Vector3f(line[j].coordinates[0], line[j].coordinates[1],
264                     line[j].coordinates[2]);
265         }
266         int numOfPoints = int((line.back().time-starting)/(aveSlice))+1;
267
268         Eigen::VectorXf& lineCoordinate = interpolatedLine[i];
269         lineCoordinate = Eigen::VectorXf(3*numOfPoints);
270
271         float current;
272         for(int j=0; j<numOfPoints; ++j)
273         {
274             current = starting + aveSlice*j;
275
276             // before the occuring time, should be directly repeating the first point
277             if(current<=line[0].time)
278             {
279                 for(int k=0; k<3; ++k)
280                 {
281                     lineCoordinate(3*j+k) = line[0].coordinates[k];
282                 }
283             }
```

```
284             // this time has been recorded in the map, directly load the data
285             else if(timeCoordinates.find(current)!=timeCoordinates.end())
286             {
287                 for(int k=0; k<3; ++k)
288                 {
289                     lineCoordinate(3*j+k) = timeCoordinates[current](k);
290                 }
291             }
292             // else, find the left and right time step and perform linear interpolation
293             else
294             {
295                 // find the clipping left and right time slices for interpolation
296                 auto right = timeCoordinates.upper_bound(current);
297                 auto left = std::prev(right);
298 
299                 float ratio = (current-left->first)/(right->first-left->first);
300                 Eigen::Vector3f currentPoint = (1.0-ratio)*left->second + ratio*right->second;
301                 for(int k=0; k<3; ++k)
302                 {
303                     lineCoordinate(3*j+k) = currentPoint(k);
304                 }
305             }
306         }
307     }
308 }
```

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.

```
94 {
95     std::get<0>(timeRange) = FLT_MAX;
96     std::get<1>(timeRange) = -1.0;
97 
98     std::vector<PathlinePoint> pointCoordinateVec;
99 
100     std::ifstream fin(fileName, ios::in);
101 
102     if(fin.fail())
103     {
104         std::cout << "Error for opening file contents!" << std::endl;
105         exit(1);
106     }
107 
108     string line;
109 
110     // ignore the header file of vtk
111     for(int i=0; i<5; ++i)
112         getline(fin, line);
113 
114     stringstream ss(line);
115     ss >> line, ss >> line; // get number of points
116 
117     ss.clear();
118     ss.str("");
119 
120     // extract the number of points
121     const int& numOfPoints = std::atoi(line.c_str());
122     std::cout << "There are " << numOfPoints << " points read from the file!" << std::endl;
123 
124     // assign the memory for numOfPoints
125     pointCoordinateVec.resize(numOfPoints);
126 
127     // start traversal for point coordinates
128     int index = 0, separation = 0;
129     while(getline(fin, line) && line.size()!=0)
130     {
131         ss.str(line);
132         separation = 0;
133 
134         // read the one line of points into the cache
135         while(ss>>line)
136         {
137             pointCoordinateVec[index].coordinates[separation] = std::atof(line.c_str());
138             if(separation == 2)
139             {
140                 separation = 0;
141                 ++index;
```

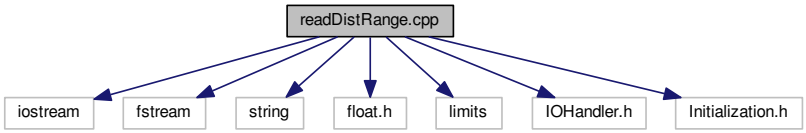
```
142             }
143             else
144                 ++separation;
145         }
146         ss.clear();
147         ss.str("");
148     }
149 
150     assert(index == numOfPoints);
151 
152     // read the number of lines from the txt file
153     for(int i=0; i<8; ++i)
154         getline(fin, line);
155 
156     ss.str(line);
157     ss >> line, ss >> line;
158     const int& numOfLines = std::atoi(line.c_str());
159     std::cout << "There are " << numOfLines << " pathlines read from the file!" << std::endl;
160     ss.clear();
161     ss.str("");
162 
163     // store the line to point arrays, e.g., one line has which indices of points
164     std::vector<vector<int> > pointsToLine(numOfLines);
165 
166     int numOfPointsForLine, pointIndex;
167     for(int i=0; i<numOfLines; ++i)
168     {
169         getline(fin, line);
170         ss.str(line);
171         ss >> line;
172 
173         // find how many points this line contain
174         numOfPointsForLine = std::atoi(line.c_str());
175 
176         std::vector<int>& currentLine = pointsToLine[i];
177         currentLine.resize(numOfPointsForLine);
178 
179         for(int j=0; j<numOfPointsForLine; ++j)
180         {
181             ss >> line;
182             pointIndex = std::atoi(line.c_str());
183             currentLine[j] = pointIndex;
184         }
185 
186         ss.clear();
187         ss.str("");
188     }
189 
190     while(getline(fin, line))
191     {
192         ss.str(line);
193         ss >> line;
194         ss.clear();
195         ss.str("");
196         if(strcmp(line.c_str(), "time")==0)
197             break;
198     }
199 
200     index = 0;
201     while(getline(fin, line) && line.size())
202     {
203         ss.str(line);
204         while(ss >> line)
205         {
206             pointCoordinateVec[index].time = std::atof(line.c_str());
207             ++index;
208         }
209         ss.clear();
210         ss.str("");
211     }
212     assert(index == numOfPoints);
213     fin.close();
214 
215     std::cout << "File content traversal completed!" << std::endl;
216 
217     pathlines.resize(numOfLines);
218     std::vector<int> lineIndex;
219 
220     float averageSlice = 0.0;
221     for(int i=0; i<numOfLines; ++i)
222     {
223         std::vector<PathlinePoint>& currentLine = pathlines[i];
224         lineIndex = pointsToLine[i];
225         currentLine.resize(lineIndex.size());
226 
227         for(int j=0; j<lineIndex.size(); ++j)
228         {
```

```
229             currentLine[j] = pointCoordinateVec[lineIndex[j]];
230         }
231         std::get<0>(timeRange) = std::min(std::get<0>(timeRange), currentLine[0].time);
232         std::get<1>(timeRange) = std::max(std::get<1>(timeRange), currentLine.back().time);
233 
234         averageSlice += (currentLine.back().time-currentLine[0].time)/float(lineIndex.size()-1);
235         std::cout << "Pathline " << i << " has starting time " << currentLine[0].time <<
236         " and ending time " << currentLine.back().time << std::endl;
237     }
238     pointCoordinateVec.clear();
239     std::get<2>(timeRange) = averageSlice/float(numOfLines);
240 
241     std::cout << "Starting time is " << std::get<0>(timeRange) << ", ending time is " << std::get<1>(
    timeRange) <<
242     ", and average time slice is " << std::get<2>(timeRange) << std::endl;
243 }
```

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.

```
54 {
55     for(int i=0; i<16; ++i)
56     {
57         if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
58             continue;
59         /* very hard to decide whether needed to perform such pre-processing */
60         MetricPreparation object = MetricPreparation(ds.dataMatrix.rows(), ds.
       dataMatrix.cols());
61         object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(), ds.
       dataMatrix.cols(), i);
62 
63         deleteDistanceMatrix(ds.dataMatrix.rows());
64         getDistanceMatrix(ds.dataMatrix, i, object);
65 
66         const int& Row = ds.dataMatrix.rows();
67         float min_dist = numeric_limits<float>::max(), max_dist = numeric_limits<float>::min();
68 
69     #pragma omp parallel for reduction(min:min_dist) num_threads(8)
70         for (int i = 0; i < Row; ++i)
71         {
72             for (int j = 0; j < Row; ++j)
73             {
74                 if(i==j)
75                     continue;
76                 min_dist = std::min(min_dist, distanceMatrix[i][j]);
77             }
78         }
79 
80     #pragma omp parallel for reduction(max:max_dist) num_threads(8)
81         for (int i = 0; i < Row; ++i)
82         {
83             for (int j = 0; j < Row; ++j)
84             {
85                 if(i==j)
86                     continue;
87                 max_dist = std::max(max_dist, distanceMatrix[i][j]);
88             }
89         }
90 
91         std::cout << "norm " << i << " has min " << min_dist << " and max " << max_dist << std::endl;
92 
93         std::ofstream readme("../dataset/dist_range", ios::app | ios::out);
94         if(readme.fail())
95         {
96             std::cout << "Error for opening readme!" << std::endl;
97             exit(1);
98         }
99 
100         readme << "For norm " << i << ", min is " << min_dist << ", max is " << max_dist << ", and (max -
       min) is " <<
101                 (max_dist-min_dist) << std::endl;
102         readme << std::endl;
103 
104         readme.close();
105     }
106 }
```

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

Definition at line 37 of file readDistRange.cpp.

```
38 {
39     if(argc!=3)
40     {
41         std::cout << "parameter option is not right!" << std::endl;
42         exit(1);
43     }
44 
45     Dataset ds;
46     setDataset(ds, argc, argv);
47 
48     getDistRange(ds);
49 
50     return 0;
51 }
```

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

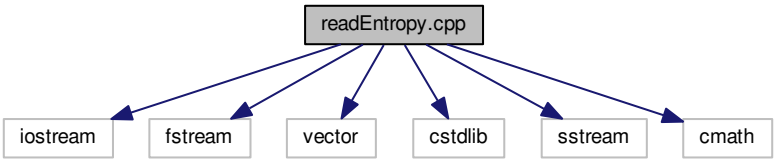
Definition at line 109 of file readDistRange.cpp.

```
110 {
111     if(argc!=3)
112     {
113         std::cout << "Input argument should have 3!" << endl
114                   << "./cluster inputFile_name(in dataset folder) "
115                   << "data_dimension(3)" << endl;
116         exit(1);
117     }
118     ds.strName = string("../dataset/")+string(argv[1]);
119     ds.dataName = string(argv[1]);
120     ds.dimension = atoi(argv[2]);
121 
122     int pathlineOption;
123     std::cout << "It is a pathline dataset? 1.Yes, 0.No" << std::endl;
124     std::cin >> pathlineOption;
125     assert(pathlineOption==1||pathlineOption==0);
126     bool isPathlines = (pathlineOption==1);
127 
128     int sampleOption;
129 
130     if(isPathlines)
131         sampleOption = 1;
132 
133     else
134     {
135         std::cout << "choose a sampling method for the dataset?" << std::endl
136                   << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
137         std::cin >> sampleOption;
138         assert(sampleOption==1||sampleOption==2);
139     }
140 
141     IOHandler::readFile(ds.strName,ds.dataVec,ds.
    vertexCount,ds.dimension,ds.maxElements);
142 
143     ds.fullName = ds.strName+"_full.vtk";
144     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.vertexCount, ds.
    dimension);
145 
146     if(sampleOption==1)
147         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
148     else if(sampleOption==2)
149         IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
150 }
```

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.

```
57 {
58     int total = 0;
59     for(int i=0;i<groupSize.size();++i)
60         total+=groupSize[i];
61 
62     float prob, result = 0.0;
63     for(int i=0;i<groupSize.size();++i)
64     {
65         prob = float(groupSize[i])/float(total);
66         result+=prob*log2f(prob);
67     }
68 
69     result = -result/log2f(groupSize.size());
70     std::cout << "Entropy is " << result << std::endl;
71 }
```

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

Definition at line 14 of file readEntropy.cpp.

```
15 {
16     if(argc!=2)
17     {
18         std::cout << "Error for argument input!" << std::endl;
19         exit(1);
20     }
21 
22     std::vector<int> storage;
23     readFile(storage, argv[1]);
24 
25     computeEntropy(storage);
26 
27     return 0;
28 }
```

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

Definition at line 31 of file readEntropy.cpp.

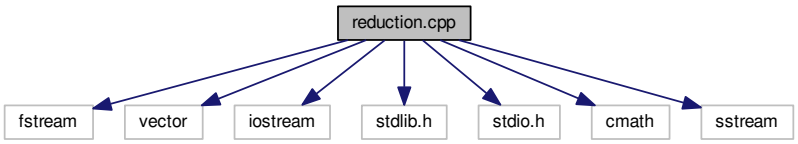
```
32 {
33     ifstream fin(fileName, ios::in);
34     if(!fin)
35     {
36         std::cout << "Error for reading a file!" << std::endl;
37         exit(1);
38     }
39 
40     string line;
41 
42     int num;
43     while(getline(fin,line))
44     {
45         num = 0;
46         stringstream ss(line);
47         while(ss>>line)
48             ++num;
49         if(num > 0)
50             groupSize.push_back(num);
51     }
52 
53     fin.close();
54 }
```

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.

```
51 {
52     std::ifstream fin(fileName, ios::in);
53     if(!fin)
54     {
55         std::cout << "Error creating files!" << std::endl;
56         exit(1);
57     }
58     stringstream ss;
59     std::vector<float> tempVec;
60 
61     string line, part;
62 
63     std::vector<float> vec(3);
64     float temp;
65 
66     int lineTag = 0;
67     while(getline(fin, line) /* && currentNumber < MAXNUMBER*/)
68     {
69         //currentDimensions = 0;
70         if(lineTag==1)
71         {
72             lineTag = (lineTag+1)%lineNumber;
73             continue;
74         }
75 
76         int tag = 0, count = 0;
77         bool isNext = false;
78         ss.str(line);
79         while(ss>>part /*&& currentDimensions<3*MAXDIMENSION*/)
80         {
81             /* operations below would remove duplicate vertices because that would damage our computation
    */
82             if(tag>=3)
83             {
84                 isNext = !isNext;
85                 tag = (tag+1)%(vertexNumber*3);
86                 continue;
87             }
88             temp = atof(part.c_str());
89             if(isNext)
90             {
91                 if(count<3)
92                 {
93                     vec[count] = temp;
94                     tag = (tag+1)%(vertexNumber*3);
95                     ++count;
96                 }
97                 if(count==3)
98                 {
99                     int size = tempVec.size();
100                     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))
101                     {
102                         tempVec.push_back(vec[0]);
103                         tempVec.push_back(vec[1]);
104                         tempVec.push_back(vec[2]);
105                     }
106                     count = 0;
107                 }
108                 continue;
109             }
110             tempVec.push_back(temp);
111             tag = (tag+1)%(vertexNumber*3);
112             //currentDimensions++;
113         }
114         /* accept only streamlines with at least three vertices */
115         if(tempVec.size()/3>2)
116         {
117             dataVec.push_back(tempVec);
118         }
119         tempVec.clear();
120         ss.clear();
121         ss.str("");
122         //currentNumber++;
123 
124         lineTag = (lineTag+1)%lineNumber;
125     }
126     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)

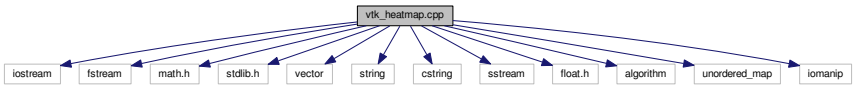
Variables

- 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.

```
336 {
337     /* get the limit range of four scalar values */
338     const int& rows = dataVec.size();
339     const int& cols = dataVec[0].size();
340 
341     float value;
342     int num;
343     /* generate the assembled four-scalar normalized for R visualization */
344     std::ofstream normalized("assembled", ios::out);
345     if(normalized.fail())
346     {
347         std::cout << "Error for creating wrong files!" << std::endl;
348         exit(1);
349     }
350 
351     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_",
352                          "d_T", "d_T"};
353     for (int i = 0; i < cols; ++i)
354     {
355         normalized << rownames[i] << "\t";
356         for (int j = 0; j < rows;++j)
357         {
358             if(dataVec[j][i]<=-9999.0)
359             {
360                 value = 0.0;
361                 normalized << value << "\t";
362                 continue;
363             }
364             num = (j%2)*2+i%2;
365             if(num<=1)
366                 value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-
    range_start)+range_start;
367             else if(num==2)
368             {
369                 if(dataVec[j][i]>=max_db_index)
370                     value = 1.0;
371                 else
372                     value = (dataVec[j][i]-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
373             }
374             else if(num==3)
375                 value = (log10(dataVec[j][i])-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
376 
377             if(num==0||num==1)
378                 normalized << value << "\t";
379             else if(num==2||num==3)
380                 normalized << (1.0+range_start)-value << "\t";
381         }
382         normalized << std::endl;
383     }
384 
385     normalized.close();
386 }
```

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

Definition at line 713 of file vtk_heatmap.cpp.

```
714 {
715     const int& rows = dataVec.size();
716     const int& cols = dataVec[0].size();
717     std::ofstream latex_table("latex_table", ios::out);
718     if(latex_table.fail())
719     {
720         std::cout << "Error for creating latex table w.r.t. data vec!" << std::endl;
721         exit(1);
722     }
```

```
723
724     /* get the best respective value */
725     string clustering[] = {"\\textbf{K-means}", "\\textbf{K-medoids}", "\\textbf{AHC}-single", "\\
    textbf{AHC}-average", "\\textbf{BIRCH}",
726                            "\\textbf{DBSCAN}", "\\textbf{OPTICS}", "\\textbf{SC}-kmeans", "\\
    textbf{SC}-eigen", "\\textbf{AP}", "\\textbf{PCA}"};
727     BestValue ***best_value = new BestValue**[cols/2];
728     for (int i = 0; i < cols/2; ++i)
729     {
730         best_value[i] = new BestValue*[2];
731         for(int j=0; j<2; ++j)
732         {
733             if(j==0)
734             {
735                 best_value[i][j] = new BestValue[2];
736                 best_value[i][j][0].value = -10000.0;
737                 best_value[i][j][1].value = -10000.0;
738             }
739             else if(j==1)
740             {
741                 best_value[i][j] = new BestValue[2];
742                 best_value[i][j][0].value = FLT_MAX;
743                 best_value[i][j][1].value = FLT_MAX;
744             }
745         }
746     }
747
748     int col_num, index;
749     for (int i = 0; i < cols; ++i)
750     {
751         col_num = i/2;
752         index = i%2;
753         for (int j = 0; j < rows; j+=2)
754         {
755             if (dataVec[j][i]<=-9999.0)
756                 continue;
757
758             if (dataVec[j][i]>best_value[col_num][0][index].value)
759             {
760                 best_value[col_num][0][index].value = dataVec[j][i];
761                 best_value[col_num][0][index].i = i;
762                 best_value[col_num][0][index].j = j;
763             }
764         }
765
766         for (int j = 1; j < rows; j+=2)
767         {
768             if (dataVec[j][i]<=-9999.0)
769                 continue;
770
771             if (dataVec[j][i]<best_value[col_num][1][index].value)
772             {
773                 best_value[col_num][1][index].value = dataVec[j][i];
774                 best_value[col_num][1][index].i = i;
775                 best_value[col_num][1][index].j = j;
776             }
777         }
778     }
779
780
781     string tag[] = {"\\dashuline", "\\underline", "*", "\\textbf"};
782
783     for (int j = 0; j<rows; ++j)
784     {
785         latex_table << "\\multirow{2}{*}{" << clustering[j/2] << "} ";
786         for (int i = 0; i < cols; ++i)
787         {
788             latex_table << "&";
789             if (i==best_value[i/2][0][i%2].i && j==best_value[i/2][0][i%2].j)
790                 latex_table << tag[i%2] << "{";
791
792             if (dataVec[j][i]<=-9999.0)
793                 latex_table << "-";
794             else
795                 latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
796             if (i==best_value[i/2][0][i%2].i && j==best_value[i/2][0][i%2].j)
797                 latex_table << "}";
798         }
799         latex_table << "\\\\" << std::endl;
800
801         j+=1;
802         for (int i = 0; i < cols; ++i)
803         {
804             if(i%2==0)
805             {
806                 latex_table << "& \\multicolumn{1}{l}{";
807
```

```
808                 if (dataVec[j][i]<=-9999.0)
809                     latex_table << "\\hspace{0.23cm}" << "-";
810                 else if (dataVec[j][i]>=0.01 && dataVec[j][i]<1000)
811                     latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
812                 else
813                     latex_table << std::scientific << std::setprecision(1) << dataVec[j][i];
814                 if (i==best_value[i/2][1][0].i && j==best_value[i/2][1][0].j)
815                     latex_table << tag[2];
816 
817                 latex_table << "}";
818             }
819             else
820             {
821                 latex_table << "&";
822                 if (i==best_value[i/2][1][1].i && j==best_value[i/2][1][1].j)
823                     latex_table << tag[3] << "{";
824 
825                 if (dataVec[j][i]<=-9999.0)
826                     latex_table << "-";
827                 else if (dataVec[j][i]>=0.01)
828                     latex_table << std::fixed << std::setprecision(3) << dataVec[j][i];
829                 else
830                     latex_table << std::scientific << std::setprecision(1) << dataVec[j][i];
831                 if (i==best_value[i/2][1][1].i && j==best_value[i/2][1][1].j)
832                     latex_table << "}";
833             }
834         }
835         latex_table << "\\\\" << std::endl;
836 
837         latex_table << "\\hline" << std::endl;
838     }
839 
840     latex_table.close();
841 
842 }
```

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

Definition at line 488 of file vtk_heatmap.cpp.

```
489 {
490     /* get the limit range of four scalar values */
491     const int& rows = dataVec.size();
492     const int& cols = dataVec[0].size();
493 
494     float value;
495     int num;
496 
497     unordered_map<int, std::vector<AverageColumn> > rankMap;
498 
499     for (int i = 0; i < 4; ++i)
500     {
501         rankMap.insert(make_pair(i, std::vector<AverageColumn>()));
502     }
503 
504     string metric[]={"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};
505 
506     int effective[2];
507     for (int i = 0; i < cols; ++i)
508     {
509         AverageColumn ac[2];
510         ac[0].name = ac[1].name = metric[i/2];
511         effective[0] = effective[1] = 0;
512         for (int j = 0; j < rows;j+=2)
513         {
514             num = i%2;
515             ac[0].valueVec.push_back(dataVec[j][i]);
516             if (dataVec[j][i]>-9999.0)
517             {
518                 ++effective[0];
519                 ac[0].average+=dataVec[j][i];
520             }
521 
522             num = i%2+2;
523             ac[1].valueVec.push_back(dataVec[j+1][i]);
524             if (dataVec[j+1][i]>-9999.0)
525             {
526                 ++effective[1];
527                 ac[1].average+=dataVec[j+1][i];
528             }
```

```
529         }
530         ac[0].average/=effective[0];
531         ac[1].average/=effective[1];
532 
533         rankMap[i%2].push_back(ac[0]);
534         rankMap[i%2+2].push_back(ac[1]);
535     }
536 
537     string clustering[]={"K-means", "K-medoids", "AHC-single", "AHC-average", "BIRCH", "DBSCAN", "OPTICS",
    "SC-kmeans", "SC-eigen", "AP", "PCA"};
538     unordered_map<int, std::vector<AverageClustering> > clusteringMap;
539     for (int j = 0; j < rows; ++j)
540     {
541         AverageClustering ac[2];
542         ac[0].originalIndex = ac[1].originalIndex = j/2;
543         effective[0] = effective[1] = 0;
544         for (int i = 0; i < cols; ++i)
545         {
546             if (dataVec[j][i]>-9999.0)
547             {
548                 ++effective[i%2];
549                 ac[i%2].average+=dataVec[j][i];
550             }
551         }
552         ac[0].average/=effective[0];
553         ac[1].average/=effective[1];
554         clusteringMap[(j%2)*2].push_back(ac[0]);
555         clusteringMap[(j%2)*2+1].push_back(ac[1]);
556     }
557 
558 
559     for (int i = 0; i < 4; ++i)
560     {
561         /* ranking silhouette and gamma from largest to smallest */
562         if(i<=1)
563         {
564             std::sort(rankMap[i].begin(), rankMap[i].end(), [](const
    AverageColumn& a, const AverageColumn& b)
565             {return a.average>b.average;});
566 
567             std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const
    AverageClustering& a, const AverageClustering& b)
568             {return a.average>b.average;});
569         }
570         /* ranking db index and validity from smallest to largest */
571         else
572         {
573             std::sort(rankMap[i].begin(), rankMap[i].end(), [](const
    AverageColumn& a, const AverageColumn& b)
574             {return a.average<b.average;});
575             std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const
    AverageClustering& a, const AverageClustering& b)
576             {return a.average<b.average;});
577         }
578     }
579     std::unordered_map<int, std::vector<int> > verticalOrder;
580     for (int i = 0; i < 4; ++i)
581     {
582         verticalOrder[i] = std::vector<int>(clusteringMap[i].size());
583         for (int j=0; j<clusteringMap[i].size(); ++j)
584         {
585             verticalOrder[i][j] = clusteringMap[i][j].originalIndex;
586         }
587     }
588 
589     float realValue;
590     /* generate the assembled four-scalar normalized for R visualization */
591     string file_names[] = {"silhouette_ranking", "gamma_ranking", "dbindex_ranking", "validity_ranking"};
592 
593     string best_names[] = {"silhouette_best", "gamma_best", "dbindex_best", "validity_best"};
594 
595     BestValue best_value[4];
596     best_value[0].value = best_value[1].value = -FLT_MAX;
597     best_value[2].value = best_value[3].value = FLT_MAX;
598 
599     std::cout << "Ranking started..." << std::endl;
600     for (int i = 0; i < 4; ++i)
601     {
602         std::ofstream colnames((to_string(i)+"_colnames").c_str(), ios::out);
603         if(colnames.fail())
604         {
605             std::cout << "Error for creating files!" << std::endl;
606             exit(1);
607         }
608 
609         std::cout << file_names[i] << std::endl;
610         for (int j = 0; j < verticalOrder[i].size(); ++j)
```

```
611         {
612             std::cout << "(" << clustering[verticalOrder[i][j]] << "," << clusteringMap[i][j].average << "
    ), ";;
613             colnames << clustering[verticalOrder[i][j]] << " ";
614         }
615         colnames << std::endl;
616         colnames.close();
617 
618         std::ofstream ranked_file(file_names[i].c_str(), ios::out);
619         if (ranked_file.fail())
620         {
621             std::cout << "Error for creating file!" << std::endl;
622             exit(1);
623         }
624 
625         const std::vector<AverageColumn>& element = rankMap[i];
626         for (int j = 0; j < element.size(); ++j)
627         {
628             ranked_file << element[j].name << "\t";
629             for (int k = 0; k<element[j].valueVec.size(); ++k)
630             {
631 
632                 realValue = (element[j].valueVec)[verticalOrder[i][k]];
633 
634                 if(realValue<=-9999.0)
635                 {
636                     ranked_file << 0.0 << "\t";
637                     continue;
638                 }
639                 else
640                 {
641                     if (i<=1)
642                     {
643                         if(realValue>=best_value[i].value)
644                         {
645                             best_value[i].value=realValue;
646                             best_value[i].i = k;
647                             best_value[i].j = j;
648                         }
649                     }
650                     else if (i>=2)
651                     {
652                         if(realValue<=best_value[i].value)
653                         {
654                             best_value[i].value=realValue;
655                             best_value[i].i = k;
656                             best_value[i].j = j;
657                         }
658                     }
659                 }
660 
661                 if(i<=1)
662                     value = (realValue-data_range[i][0])/data_range[i][2]*(1.0-
    range_start)+range_start;
663                 else if(i==2)
664                 {
665                     if(realValue>=max_db_index)
666                         value = 1.0;
667                     else
668                         value = (realValue-data_range[i][0])/
    data_range[i][2]*(1.0-range_start)+range_start;
669                 }
670                 else
671                     value = (log10(realValue)-data_range[i][0])/
    data_range[i][2]*(1.0-range_start)+range_start;
672 
673                 if(i<=1)
674                     ranked_file << value << "\t";
675                 else
676                     ranked_file << (1.0+range_start)-value << "\t";
677             }
678             ranked_file << std::endl;
679         }
680         std::cout << std::endl;
681         ranked_file.close();
682     }
683 
684     std::cout << std::endl;
685     for (int i = 0; i < 4; ++i)
686     {
687         std::ofstream best_file(best_names[i].c_str(), ios::out);
688         if(best_file.fail())
689         {
690             std::cout << "Error for creating files!" << std::endl;
691             exit(1);
692         }
693 
```

```
694         const std::vector<AverageColumn>& element = rankMap[i];
695         std::cout << best_value[i].value << std::endl;
696         for (int j = 0; j < element.size(); ++j)
697         {
698             best_file << element[j].name << "\t";
699             for (int k = 0; k < element[j].valueVec.size(); ++k)
700             {
701                 if (best_value[i].i==k && best_value[i].j==j)
702                     best_file << 1.0 << "\t";
703                 else
704                     best_file << -1.0 << "\t";
705             }
706             best_file << std::endl;
707         }
708         best_file.close();
709     }
710 }
```

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

Definition at line 388 of file vtk_heatmap.cpp.

```
389 {
390     /* get the limit range of four scalar values */
391     const int& rows = dataVec.size();
392     const int& cols = dataVec[0].size();
393 
394     float value;
395     int num;
396     /* generate the assembled four-scalar normalized for R visualization */
397     std::ofstream silouette("Silhouette", ios::out), similarity("Gamma", ios::out),
    dbindex("DBindex", ios::out), validity("Validity", ios::out);
398     if(silouette.fail() || similarity.fail() || dbindex.fail() ||
    validity.fail())
399     {
400         std::cout << "Error for creating wrong files!" << std::endl;
401         exit(1);
402     }
403 
404     string metric[] = {"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};
405     for (int i = 0; i < cols; ++i)
406     {
407         if(i%2==0)
408         {
409             silouette << metric[i/2] << "\t";
410             dbindex << metric[i/2] << "\t";
411         }
412         else
413         {
414             similarity << metric[i/2] << "\t";
415             validity << metric[i/2] << "\t";
416         }
417         for (int j = 0; j < rows;++j)
418         {
419             num = (j%2)*2+i%2;
420             if(dataVec[j][i]<=-9999.0)
421             {
422                 value = 0.0;
423                 switch(num)
424                 {
425                     case 0:
426                         silouette << value << "\t";
427                         break;
428 
429                     case 1:
430                         similarity << value << "\t";
431                         break;
432 
433                     case 2:
434                         dbindex << value << "\t";
435                         break;
436 
437                     case 3:
438                         validity << value << "\t";
439                         break;
440                 }
441 
442                 continue;
443             }
444             if(num<=1)
```

```
445                 value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-
    range_start)+range_start;
446             else if(num==2)
447             {
448                 if(dataVec[j][i]>=max_db_index)
449                     value = 1.0;
450                 else
451                     value = (dataVec[j][i]-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
452             }
453             else if(num==3)
454                 value = (log10(dataVec[j][i])-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
455
456             if(num==0||num==1)
457             {
458                 if(num==0)
459                     silouette << value << "\t";
460                 else if(num==1)
461                     similarity << value << "\t";
462             }
463             else if(num==2||num==3)
464             {
465                 if(num==2)
466                     dbindex << (1.0+range_start)-value << "\t";
467                 else if(num==3)
468                     validity << (1.0+range_start)-value << "\t";
469             }
470         }
471         if(i%2==0)
472         {
473             silouette << std::endl;
474             dbindex << std::endl;
475         }
476         else
477         {
478             similarity << std::endl;
479             validity << std::endl;
480         }
481     }
482     silouette.close();
483     similarity.close();
484     dbindex.close();
485     validity << std::endl;
486 }
```

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.

```
910 {
911     /* get the limit range of four scalar values */
912     const int& rows = dataVec.size();
913     const int& cols = dataVec[0].size();
914
915     float value;
916     int num;
917
918     float value_option[] = {FLT_MAX, -FLT_MAX, 0};
919     for (int i = 0; i < 3; ++i)
920     {
921         for (int j=0;j<4;++j)
922             data_range[j][i] = value_option[i];
923     }
924
925     for (int i = 0; i < rows; ++i)
926     {
927         for (int j = 0; j < cols; ++j)
928         {
929             if(dataVec[i][j]<=-9999.0)
930                 continue;
931             num = (i%2)*2+j%2;
932             data_range[num][0] = std::min(data_range[num][0], dataVec[i][j]);
933             data_range[num][1] = std::max(data_range[num][1], dataVec[i][j]);
934
935         }
936     }
937     for (int i = 0; i < 4; ++i)
```

```
938     {
939         std::cout << data_range[i][0] << " " << data_range[i][1] << std::endl;
940         if(i==2)
941         {
942             if(data_range[i][1]>=max_db_index)
943                 data_range[i][1]=max_db_index;
944         }
945         else if(i==3)
946         {
947             data_range[i][0] = log10(data_range[i][0]);
948             data_range[i][1] = log10(data_range[i][1]);
949         }
950         data_range[i][2] = data_range[i][1]-data_range[i][0];
951     }
952 
953 
954     unordered_map<int, std::vector<AverageColumn> > rankMap;
955 
956     for (int i = 0; i < 4; ++i)
957     {
958         rankMap.insert(make_pair(i, std::vector<AverageColumn>()));
959     }
960 
961     string metric[]={"d_E", "d_F", "d_G", "d_R", "d_M", "d_H", "d_S", "d_P", "d_T"};
962 
963     int effective[2];
964     for (int i = 0; i < cols; ++i)
965     {
966         AverageColumn ac[2];
967         ac[0].name = ac[1].name = metric[i/2];
968         effective[0] = effective[1] = 0;
969         for (int j = 0; j < rows;j+=2)
970         {
971             num = i%2;
972             ac[0].valueVec.push_back(dataVec[j][i]);
973             if(file_size>=4)
974                 ac[0].std_vec.push_back(standardDeviation[j][i]);
975             if (dataVec[j][i]>-9999.0)
976             {
977                 ++effective[0];
978                 ac[0].average+=dataVec[j][i];
979             }
980 
981             num = i%2+2;
982             ac[1].valueVec.push_back(dataVec[j+1][i]);
983             if(file_size>=4)
984                 ac[1].std_vec.push_back(standardDeviation[j+1][i]);
985             if (dataVec[j+1][i]>-9999.0)
986             {
987                 ++effective[1];
988                 ac[1].average+=dataVec[j+1][i];
989             }
990         }
991         ac[0].average/=effective[0];
992         ac[1].average/=effective[1];
993 
994         rankMap[i%2].push_back(ac[0]);
995         rankMap[i%2+2].push_back(ac[1]);
996     }
997 
998     string clustering[]={"K-means", "K-medoids", "AHC-single", "AHC-average", "BIRCH", "DBSCAN", "OPTICS",
    "SC-kmeans", "SC-eigen", "AP", "PCA"};
999     unordered_map<int, std::vector<AverageClustering> > clusteringMap;
1000     for (int j = 0; j < rows; ++j)
1001     {
1002         AverageClustering ac[2];
1003         ac[0].originalIndex = ac[1].originalIndex = j/2;
1004         effective[0] = effective[1] = 0;
1005         for (int i = 0; i < cols; ++i)
1006         {
1007             if (dataVec[j][i]>-9999.0)
1008             {
1009                 ++effective[i%2];
1010                 ac[i%2].average+=dataVec[j][i];
1011             }
1012         }
1013         ac[0].average/=effective[0];
1014         ac[1].average/=effective[1];
1015         clusteringMap[(j%2)*2].push_back(ac[0]);
1016         clusteringMap[(j%2)*2+1].push_back(ac[1]);
1017     }
1018 
1019     for (int i = 0; i < 4; ++i)
1020     {
1021         /* ranking silhouette and gamma from largest to smallest */
1022         if(i<=1)
1023         {
```

```
1024            std::sort(rankMap[i].begin(), rankMap[i].end(), [](const
    AverageColumn& a, const AverageColumn& b)
1025            {return a.average>b.average;});
1026
1027            std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const
    AverageClustering& a, const AverageClustering& b)
1028            {return a.average>b.average;});
1029        }
1030        /* ranking db index and validity from smallest to largest */
1031        else
1032        {
1033            std::sort(rankMap[i].begin(), rankMap[i].end(), [](const
    AverageColumn& a, const AverageColumn& b)
1034            {return a.average<b.average;});
1035            std::sort(clusteringMap[i].begin(), clusteringMap[i].end(), [](const
    AverageClustering& a, const AverageClustering& b)
1036            {return a.average<b.average;});
1037        }
1038    }
1039    std::unordered_map<int, std::vector<int> > verticalOrder;
1040    for (int i = 0; i < 4; ++i)
1041    {
1042        verticalOrder[i] = std::vector<int>(clusteringMap[i].size());
1043        for (int j=0; j<clusteringMap[i].size(); ++j)
1044        {
1045            verticalOrder[i][j] = clusteringMap[i][j].originalIndex;
1046        }
1047    }
1048
1049    float realValue;
1050    /* generate the assembled four-scalar normalized for R visualization */
1051    string file_names[] = {"silhouette_ranking", "gamma_ranking", "dbindex_ranking", "validity_ranking"};
1052
1053    string best_names[] = {"silhouette_best", "gamma_best", "dbindex_best", "validity_best"};
1054
1055    BestValue best_value[4];
1056    best_value[0].value = best_value[1].value = -FLT_MAX;
1057    best_value[2].value = best_value[3].value = FLT_MAX;
1058
1059    for (int i = 0; i < 4; ++i)
1060    {
1061        std::ofstream colnames((to_string(i)+"_colnames").c_str(), ios::out);
1062        if(colnames.fail())
1063        {
1064            std::cout << "Error for creating files!" << std::endl;
1065            exit(1);
1066        }
1067        for (int j = 0; j < verticalOrder[i].size(); ++j)
1068        {
1069            colnames << clustering[verticalOrder[i][j]] << " ";
1070        }
1071        colnames << std::endl;
1072        colnames.close();
1073
1074        std::ofstream ranked_file(file_names[i].c_str(), ios::out);
1075        if (ranked_file.fail())
1076        {
1077            std::cout << "Error for creating file!" << std::endl;
1078            exit(1);
1079        }
1080
1081        const std::vector<AverageColumn>& element = rankMap[i];
1082        for (int j = 0; j < element.size(); ++j)
1083        {
1084            ranked_file << element[j].name << "\t";
1085            for (int k = 0; k<element[j].valueVec.size(); ++k)
1086            {
1087                realValue = (element[j].valueVec)[verticalOrder[i][k]];
1088                if(realValue<=-9999.0)
1089                {
1090                    ranked_file << 0.0 << "\t";
1091                    continue;
1092                }
1093                else
1094                {
1095                    if (i<=1)
1096                    {
1097                        if(realValue>=best_value[i].value)
1098                        {
1099                            best_value[i].value=realValue;
1100                            best_value[i].i = k;
1101                            best_value[i].j = j;
1102                        }
1103                    }
1104                    else if (i>=2)
1105                    {
1106                        if(realValue<=best_value[i].value)
```

```
1107                        {
1108                            best_value[i].value=realValue;
1109                            best_value[i].i = k;
1110                            best_value[i].j = j;
1111                        }
1112                    }
1113                }
1114
1115                if(i<=1)
1116                    value = (realValue-data_range[i][0])/data_range[i][2]*(1.0-
    range_start)+range_start;
1117                if(i==2)
1118                {
1119                    if(realValue>=max_db_index)
1120                        value = 1.0;
1121                    else
1122                        value = (realValue-data_range[i][0])/
    data_range[i][2]*(1.0-range_start)+range_start;
1123                }
1124                else if(i==3)
1125                    value = (log10(realValue)-data_range[i][0])/
    data_range[i][2]*(1.0-range_start)+range_start;
1126
1127                if(i<=1)
1128                    ranked_file << value << "\t";
1129                else
1130                    ranked_file << (1.0+range_start)-value << "\t";
1131            }
1132            ranked_file << std::endl;
1133        }
1134        ranked_file.close();
1135    }
1136
1137
1138    std::cout << std::endl;
1139    for (int i = 0; i < 4; ++i)
1140    {
1141        std::ofstream best_file(best_names[i].c_str(), ios::out);
1142        if(best_file.fail())
1143        {
1144            std::cout << "Error for creating files!" << std::endl;
1145            exit(1);
1146        }
1147
1148        const std::vector<AverageColumn>& element = rankMap[i];
1149        for (int j = 0; j < element.size(); ++j)
1150        {
1151            best_file << element[j].name << "\t";
1152            for (int k = 0; k < element[j].valueVec.size(); ++k)
1153            {
1154                if (best_value[i].i==k && best_value[i].j==j)
1155                    best_file << 1.0 << "\t";
1156                else
1157                    best_file << -1.0 << "\t";
1158            }
1159            best_file << std::endl;
1160        }
1161        best_file.close();
1162    }
1163
1164    if(file_size>=4)
1165    {
1166
1167        float std_range[4][3]=
1168        {
1169            FLT_MAX, -FLT_MAX, 0,
1170            FLT_MAX, -FLT_MAX, 0,
1171            FLT_MAX, -FLT_MAX, 0,
1172            FLT_MAX, -FLT_MAX, 0
1173        };
1174
1175        for (int i = 0; i < rows; ++i)
1176        {
1177            for (int j = 0; j < cols; ++j)
1178            {
1179                if(standardDeviation[i][j]<=-9999.0)
1180                    continue;
1181                num = (i%2)*2+j%2;
1182                std_range[num][0] = std::min(std_range[num][0], standardDeviation[i][j]);
1183                std_range[num][1] = std::max(std_range[num][1], standardDeviation[i][j]);
1184
1185            }
1186        }
1187        for (int i = 0; i < 4; ++i)
1188        {
1189            std_range[i][2] = std_range[i][1]-std_range[i][0];
1190            std::cout << "[" << std_range[i][0] << "," << std_range[i][1] << "]: " << std_range[i][2] <<
```

```
    std::endl;
1191        }
1192
1193
1194        string std_names[] = {"silhouette_std", "gamma_std", "dbindex_std", "validity_std"};
1195
1196        for (int i = 0; i < 4; ++i)
1197        {
1198            std::ofstream ranked_file(std_names[i].c_str(), ios::out);
1199            if (ranked_file.fail())
1200            {
1201                std::cout << "Error for creating file!" << std::endl;
1202                exit(1);
1203            }
1204
1205            const std::vector<AverageColumn>& element = rankMap[i];
1206            for (int j = 0; j < element.size(); ++j)
1207            {
1208                ranked_file << element[j].name << "\t";
1209                for (int k = 0; k<element[j].std_vec.size(); ++k)
1210                {
1211                    realValue = (element[j].std_vec)[verticalOrder[i][k]];
1212                    if(realValue<=-9999.0)
1213                    {
1214                        ranked_file << 0.0 << "\t";
1215                        continue;
1216                    }
1217                    value = (realValue-std_range[i][0])/std_range[i][2]*(1.0-
    range_start)+range_start;
1218
1219                    ranked_file << value << "\t";
1220                }
1221                ranked_file << std::endl;
1222            }
1223            ranked_file.close();
1224        }
1225    }
1226
1227 }
```

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

Definition at line 186 of file vtk_heatmap.cpp.

```
187 {
188     /* get the limit range of four scalar values */
189     const int& rows = dataVec.size();
190     const int& cols = dataVec[0].size();
191
192     for(int i=0; i<4; ++i)
193     {
194         data_range[i][0] = FLT_MAX;
195         data_range[i][1] = -FLT_MAX;
196         data_range[i][2] = 0;
197     }
198
199     float value;
200     int num;
201     int nonZero = 0;
202     for (int i = 0; i < rows; ++i)
203     {
204         for (int j = 0; j < cols; ++j)
205         {
206             if(dataVec[i][j]<=-9999.0)
207                 continue;
208             num = (i%2)*2+j%2;
209             ++nonZero;
210             data_range[num][0] = std::min(data_range[num][0], dataVec[i][j]);
211             data_range[num][1] = std::max(data_range[num][1], dataVec[i][j]);
212
213         }
214     }
215     for (int i = 0; i < 4; ++i)
216     {
217         std::cout << data_range[i][0] << " " << data_range[i][1] << std::endl;
218         if(i==3)
219         {
220             data_range[i][0] = log10(data_range[i][0]);
221             data_range[i][1] = log10(data_range[i][1]);
222         }
```

```
223         else if(i==2)
224         {
225             if(data_range[i][1]>=max_db_index)
226                 data_range[i][1]=max_db_index;
227         }
228         data_range[i][2] = data_range[i][1]-data_range[i][0];
229     }
230 
231     /* generate heatmap values */
232     std::ofstream ofs("heatmap.vtk", ios::out);
233     if(ofs.fail())
234     {
235         std::cout << "Error for creating vtk file!" << std::endl;
236         exit(1);
237     }
238     ofs << "# vtk DataFile Version 3.0\n"
239         << "matrix_vis" << "\n"
240         << "ASCII\n\n"
241         << "DATASET POLYDATA\n";
242     ofs << "POINTS " << (rows+1)*(cols+1) << " float\n";
243     const float& x_step = 0.1;
244     const float& y_step = 0.1;
245     for (int j = 0; j < rows+1; ++j)
246     {
247         for (int i = 0; i < cols+1; ++i)
248         {
249             ofs << i*x_step << " " << j*y_step << " " << 0 << std::endl;
250         }
251     }
252     ofs << "POLYGONS " << nonZero << " " << 5*nonZero << "\n";
253     int x, y;
254     for (int j = rows-1; j >=0; --j)
255     {
256         for (int i = 0; i<cols; ++i)
257         {
258             if(dataVec[j][i]<=-9999.0)
259                 continue;
260             ofs << 4 << " " << make_index(i,rows-1-j,cols+1) << " " <<
    make_index(i+1,rows-1-j,cols+1) << " "
261                 << make_index(i+1,rows-j,cols+1) << " " << make_index(i,rows-j,cols+1)
    << std::endl;
262         }
263     }
264     ofs << "CELL_DATA " << nonZero << "\n" << "SCALARS " << "label" << " float 1\n" << "LOOKUP_TABLE
     default\n";
265     for (int j = rows-1; j >=0; --j)
266     {
267         for (int i = 0; i < cols; ++i)
268         {
269             if(dataVec[j][i]<=-9999.0)
270                 continue;
271             num = (j%2)*2+i%2;
272 
273             if(num<=1)
274             {
275                 value = (dataVec[j][i]-data_range[num][0])/data_range[num][2]*(1.0-
    range_start)+range_start;
276             }
277             else if(num==3)
278             {
279                 value = (log10(dataVec[j][i])-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
280             }
281             else if(num==2)
282             {
283                 if(dataVec[j][i]>=max_db_index)
284                     value = 1.0;
285                 else
286                     value = (dataVec[j][i]-data_range[num][0])/
    data_range[num][2]*(1.0-range_start)+range_start;
287             }
288             if(num==0 || num==1)
289                 ofs << value << std::endl;
290             else
291                 ofs << (1.0+range_start)-value << std::endl;
292 
293         }
294     }
295     ofs.close();
296 
297     /* generate boundary grids for 2X2 */
298     std::ofstream grid("grid.vtk", ios::out);
299     if(grid.fail())
300     {
301         std::cout << "Error for creating file!" << std::endl;
302         exit(1);
303     }
```

```
304     grid << "# vtk DataFile Version 3.0\n"
305         << "matrix_vis" << "\n"
306         << "ASCII\n\n"
307         << "DATASET POLYDATA\n";
308     grid << "POINTS " << (rows/2+1)*(cols/2+1) << " float\n";
309     for (int j = 0; j < rows/2+1; ++j)
310     {
311         for (int i = 0; i < cols/2+1; ++i)
312         {
313             grid << i*2.0*x_step << " " << j*2.0*y_step << " " << 0 << std::endl;
314         }
315     }
316     const int& line_number = rows/2*(cols/2+1)+(rows/2+1)*cols/2;
317     grid << "LINES " << line_number << " " << 3*line_number << std::endl;
318     for (int j = 0; j < rows/2; ++j)
319     {
320         for (int i = 0; i < cols/2; ++i)
321         {
322             grid << 2 << " " << make_index(i,j,cols/2+1) << " " <<
    make_index(i+1,j,cols/2+1) << std::endl;
323             grid << 2 << " " << make_index(i,j,cols/2+1) << " " <<
    make_index(i,j+1,cols/2+1) << std::endl;
324         }
325         grid << 2 << " " << make_index(cols/2,j,cols/2+1) << " " <<
    make_index(cols/2,j+1,cols/2+1) << std::endl;
326     }
327     for (int i = 0; i < cols/2; ++i)
328     {
329         grid << 2 << " " << make_index(i,rows/2,cols/2+1) << " " <<
    make_index(i+1,rows/2,cols/2+1) << std::endl;
330     }
331     grid.close();
332
333 }
```

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.

```
847 {
848     std::vector<std::vector<float> > totalValue[file_size];
849
850     for (int i = 0; i < file_size; ++i)
851     {
852         readData(totalValue[i], filenames[i].c_str());
853     }
854
855     const int& rows = totalValue[0].size();
856     const int& cols = totalValue[0][0].size();
857
858     averageValue = std::vector< std::vector<float> >(rows, std::vector<float>(cols));
859
860     if(file_size>=4)
861         standardDeviation = std::vector< std::vector<float> >(rows, std::vector<float>(cols));
862
863     float average, stdeviation, value;
864     int effective;
865     for (int i = 0; i < cols; ++i)
866     {
867         for (int j = 0; j < rows; ++j)
868         {
869             average = stdeviation = 0.0;
870             effective = 0;
871             for (int k = 0; k < file_size; ++k)
872             {
873                 value = totalValue[k][j][i];
874                 if (value<=-9999.0)
875                     continue;
876                 average+=value;
877                 if(file_size>=4)
878                     stdeviation+=value*value;
879                 ++effective;
880             }
881             if(effective==0)
882             {
883                 averageValue[j][i] = -10000.0;
884                 if(file_size>=4)
885                     standardDeviation[j][i] = -10000.0;
```

```
886             }
887             else
888             {
889                 average/=float(effective);
890                 averageValue[j][i] = average;
891                 if(file_size>=4)
892                 {
893                     stdeviation = stdeviation/float(effective)-average*average;
894                     if(stdeviation<0)
895                     {
896                         std::cout << "Error for one-pass standard deviation computation!" << std::endl;
897                         exit(1);
898                     }
899                     standardDeviation[j][i] = sqrt(stdeviation);
900                 }
901             }
902         }
903     }
904
905 }
```

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

Definition at line 96 of file vtk_heatmap.cpp.

```
97 {
98     // if(argc!=2)
99     // {
100     //  std::cout << "Error for argument count!" << std::endl;
101     //  exit(1);
102     // }
103
104     // std::vector<std::vector<float> > dataVec;
105
106     // readData(dataVec, argv[1]);
107
108     // createHeatMap(dataVec);
109
110     // create_assemble(dataVec);
111
112     // //create_separate(dataVec);
113
114     // create_ranking(dataVec);
115
116     // create_latex_table(dataVec);
117
118
119     if(argc!=1)
120     {
121         std::cout << "Get average and std so no need for argument!" << std::endl;
122         exit(1);
123     }
124     std::vector<std::vector<float> > averageValue, standardDeviation;
125
126     //string filenames[] = {"bernard_evaluation", "crayfish_evaluation", "cylinder_evaluation",
      "hurricane_evaluation", "solar_plume_evaluation", "tornado_evaluation"};
127
128     string filenames[] = {"cylinder_pathlines_evaluation", "tub_pathlines_evaluation", "
     blood_flow_evaluation"};
129
130     get_average_value(averageValue, standardDeviation, filenames, sizeof(filenames)/sizeof
     (string));
131
132     create_std_ranking(averageValue, standardDeviation, sizeof(filenames)/sizeof(string))
     ;
133
134     create_assemble(averageValue);
135
136     create_latex_table(averageValue);
137
138     createHeatMap(averageValue);
139
140     return 0;
141 }
```

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.

```
181 {
182     return j*col+i;
183 }
```

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

Definition at line 144 of file vtk_heatmap.cpp.

```
145 {
146     std::ifstream fin(fileName, ios::in);
147     if(fin.fail())
148     {
149         std::cout << "Error for reading data from existing file!" << std::endl;
150         exit(1);
151     }
152 
153     stringstream ss;
154     string line;
155     std::vector<float> row;
156     while(getline(fin, line))
157     {
158         ss.str(std::string());
159         ss.clear();
160         ss << line;
161         while(ss>>line)
162         {
163             if(strcmp(line.c_str(), "-")==0)
164             {
165                 row.push_back(-10000.0);
166             }
167             else
168             {
169                 row.push_back(std::atof(line.c_str()));
170             }
171         }
172         dataVec.push_back(row);
173         row.clear();
174     }
175 
176     fin.close();
177 }
```

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