Clustering methods

v1.0

Generated by Doxygen 1.8.9.1

Tue Mar 10 2015 17:08:36

Contents

1 Deprecated List 1

2 Modules Index 3

2.1 Modules List . . . 3

3 Data Type Index 5

3.1 Data Types List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Module Documentation 9

5.1 module_calcul Module Reference . . . 9

5.1.1 Detailed Description . . . 9

5.1.2 Function/Subroutine Documentation . . . 9

5.1.2.1 apply_kernel_k_means . . . 10

5.1.2.2 apply_spectral_clustering . . . 10

5.1.2.3 gaussian_kernel . . . 10

5.1.2.4 get_sigma . . . 11

5.1.2.5 get_sigma_interface . . . 11

5.1.2.6 mean_shift . . . 12

5.1.2.7 poly_kernel . . . 12

5.2 module_decoupe Module Reference . . . 12

5.2.1 Detailed Description . . . 13

5.2.2 Function/Subroutine Documentation . . . 13

5.2.2.1 define_bounds . . . 13

5.2.2.2 define_domains . . . 14

5.2.2.3 group_clusters . . . 14

5.2.2.4 partition_data . . . 15

5.2.2.5 partition_with_interface . . . 16

5.2.2.6 partition_with_overlapping . . . 17

5.3 module_embed Module Reference . . . 17

5.3.1 Detailed Description . . . 18

5.3.2 Function/Subroutine Documentation . . . 18
5.3.2.1 apply_kmeans . . . 18
5.3.2.2 apply_spectral_embedding . . . 19
5.4 module_entree Module Reference . . . 20
5.4.1 Detailed Description . . . 21
5.4.2 Function/Subroutine Documentation . . . 21
5.4.2.1 assign_picture_array . . . 21
5.4.2.2 help . . . 21
5.4.2.3 read_coordinates_data . . . 22
5.4.2.4 read_geometric_data . . . 22
5.4.2.5 read_params . . . 23
5.4.2.6 read_picture_data . . . 23
5.4.2.7 read_threshold_data . . . 24
5.5 module_mpi Module Reference . . . 25
5.5.1 Detailed Description . . . 25
5.5.2 Function/Subroutine Documentation . . . 25
5.5.2.1 receive_clusters . . . 25
5.5.2.2 receive_number_clusters . . . 26
5.5.2.3 receive_partitioning . . . 26
5.5.2.4 send_clusters . . . 27
5.5.2.5 send_number_clusters . . . 27
5.5.2.6 send_partitioning . . . 27
5.6 module_solve Module Reference . . . 28
5.6.1 Detailed Description . . . 28
5.6.2 Function/Subroutine Documentation . . . 28
5.6.2.1 solve_dgeev . . . 28
5.6.2.2 solve_dgeevx . . . 29
5.6.2.3 solve_dsyev . . . 29
5.6.2.4 solve_dsyevr . . . 29
5.6.2.5 solve_dsyevx . . . 29
5.7 module_sortie Module Reference . . . 30
5.7.1 Detailed Description . . . 30
5.7.2 Function/Subroutine Documentation . . . 30
5.7.2.1 write_domains . . . 30
5.7.2.2 write_final_clusters . . . 31
5.7.2.3 write_metadata . . . 31
5.7.2.4 write_partial_clusters . . . 32
5.7.2.5 write_partitioning . . . 32
5.8 module_sparse Module Reference . . . 33
5.8.1 Function/Subroutine Documentation . . . 33

5.8.1.1 apply_spectral_clustering_sparse . . . 33
5.8.1.2 apply_spectral_embedding_sparse . . . 34
5.8.1.3 compute_matvec_prod . . . 35
5.8.1.4 solve_arpack . . . 36
5.9 module_structure Module Reference . . . 36
5.9.1 Detailed Description . . . 37
5.10 module_teste_clusters Module Reference . . . 37
5.10.1 Function/Subroutine Documentation . . . 37
5.10.1.1 create_data . . . 37
5.10.1.2 create_executable . . . 37
5.10.1.3 create_test . . . 37
5.10.1.4 execute_test . . . 37
5.11 module_visuclusters Module Reference . . . 38
5.11.1 Detailed Description . . . 38
5.11.2 Function/Subroutine Documentation . . . 38
5.11.2.1 list_commands . . . 38
5.11.2.2 read_metadata . . . 39
5.11.2.3 write_assignment . . . 39
5.11.2.4 write_final_clusters . . . 40
5.11.2.5 write_partial_clusters . . . 41
5.11.2.6 write_partitioning . . . 42
5.12 module_visuclusters_gmsh Module Reference . . . 42
5.12.1 Detailed Description . . . 43
5.12.2 Function/Subroutine Documentation . . . 43
5.12.2.1 list_commands_gmsh . . . 43
5.12.2.2 write_assignment_gmsh . . . 43
5.12.2.3 write_final_clusters_gmsh . . . 44
5.12.2.4 write_partial_clusters_gmsh . . . 45
5.12.2.5 write_partitioning_gmsh . . . 45
5.12.2.6 write_point_coord_format . . . 46
5.12.2.7 write_point_picture_format . . . 46
5.13 module_visuclusters_paraview Module Reference . . . 47
5.13.1 Detailed Description . . . 47
5.13.2 Function/Subroutine Documentation . . . 47
5.13.2.1 list_commands_paraview . . . 47
5.13.2.2 write_assignment_paraview . . . 48
5.13.2.3 write_final_clusters_paraview . . . 49
5.13.2.4 write_partial_clusters_paraview . . . 50
5.13.2.5 write_partitioning_paraview . . . 51
5.13.2.6 write_points_coord_format . . . 51

5.13.2.7 write_points_picture_format . . . 52
5.14 module_visuclusters_structure Module Reference . . . 52
5.14.1 Detailed Description . . . 53

6 Data Type Documentation **55**

6.1 module_structure::type_clustering_param Type Reference . . . 55
6.1.1 Member Data Documentation . . . 55
6.1.1.1 bandwidth . . . 55
6.1.1.2 clustering_method_id . . . 55
6.1.1.3 delta . . . 55
6.1.1.4 gam . . . 55
6.1.1.5 kernelfunindex . . . 55
6.1.1.6 sigma . . . 55
6.2 module_structure::type_clusters Type Reference . . . 55
6.2.1 Member Data Documentation . . . 55
6.2.1.1 nb . . . 55
6.2.1.2 nbelt . . . 55
6.3 module_structure::type_data Type Reference . . . 56
6.3.1 Member Data Documentation . . . 56
6.3.1.1 coord . . . 56
6.3.1.2 dim . . . 56
6.3.1.3 geom . . . 56
6.3.1.4 image . . . 56
6.3.1.5 imgdim . . . 56
6.3.1.6 imgmap . . . 56
6.3.1.7 imgt . . . 56
6.3.1.8 interface . . . 57
6.3.1.9 nb . . . 57
6.3.1.10 nbclusters . . . 57
6.3.1.11 pas . . . 57
6.3.1.12 point . . . 57
6.3.1.13 recouvrement . . . 57
6.3.1.14 refimg . . . 57
6.3.1.15 seuil . . . 57
6.4 module_visuclusters_structure::type_params Type Reference . . . 57
6.4.1 Member Data Documentation . . . 57
6.4.1.1 coord . . . 57
6.4.1.2 dim . . . 57
6.4.1.3 geom . . . 57
6.4.1.4 image . . . 57

6.4.1.5 imgdim . . . 57
6.4.1.6 imgmap . . . 57
6.4.1.7 imgt . . . 57
6.4.1.8 interface . . . 57
6.4.1.9 mesh . . . 58
6.4.1.10 nbclusters . . . 58
6.4.1.11 nbp . . . 58
6.4.1.12 nbproc . . . 58
6.4.1.13 pas . . . 58
6.4.1.14 recouvrement . . . 58
6.4.1.15 refimg . . . 58
6.4.1.16 seuil . . . 58
6.5 module_structure::type_points Type Reference . . . 58
6.5.1 Member Data Documentation . . . 58
6.5.1.1 cluster . . . 58
6.5.1.2 coord . . . 58
6.6 module_teste_clusters::type_test Type Reference . . . 58
6.6.1 Member Data Documentation . . . 58
6.6.1.1 datatype . . . 58
6.6.1.2 decoupe . . . 58
6.6.1.3 decoupetype . . . 58
6.6.1.4 dir . . . 59
6.6.1.5 epaisseur . . . 59
6.6.1.6 fichier . . . 59
6.6.1.7 nbproc . . . 59
6.6.1.8 output . . . 59
6.6.1.9 visug . . . 59
6.6.1.10 visup . . . 59

7 File Documentation **61**

7.1 module_calcul.f90 File Reference . . . 61
7.2 module_decoupe.f90 File Reference . . . 61
7.3 module_embed.f90 File Reference . . . 62
7.4 module_entree.f90 File Reference . . . 62
7.5 module_MPI.f90 File Reference . . . 63
7.6 module_solve.f90 File Reference . . . 63
7.7 module_sortie.f90 File Reference . . . 64
7.8 module_sparse.f90 File Reference . . . 64
7.9 module_structure.f90 File Reference . . . 64
7.10 module_teste_clusters.f90 File Reference . . . 65

7.11 module_visuclusters.f90 File Reference 65
7.12 module_visuclusters_gmsh.f90 File Reference 66
7.13 module_visuclusters_paraview.f90 File Reference 66
7.14 module_visuclusters_structure.f90 File Reference 67

Index **69**

Chapter 1

Deprecated List

Subprogram module_calcul::get_sigma_interface (proc_id, partitioned_data, sigma, bounds, partitioning, epsilon)

Use get_sigma() instead

Parameters

in	*partitioned_data*	the partitioned data for computing
in	*bounds*	the intervals along each dimension representing the bounds of each partition
in	*epsilon*	the slice thickness
in	*proc_id*	the processus identifier
in	*partitioning*	the partitionning (number of processors along each dimension)
out	*sigma*	the affinity parameter

Chapter 2

Modules Index

2.1 Modules List

Here is a list of all modules with brief descriptions:

module_calcul
: Contains the spectral clustering method and methods that computes affinity parameters for kernels and overlapping . . . 9

module_decoupe
: Contains methods enabling the partitionning and the groupping of the data . . . 12

module_embed
: Contains K-means and spectral embedding algorithms . . . 17

module_entree
: Contains methods enabling data and parameter file reading . . . 20

module_mpi
: Contains methods related to parallelism . . . 25

module_solve
: Contains methods from Lapack library dealing with eigen values computing . . . 28

module_sortie
: Contains methods enabling writing results in specific formatted files . . . 30

module_sparse . . . 33

module_structure
: Contains structure types required by the different modules . . . 36

module_teste_clusters . . . 37

module_visuclusters
: Contains methods enabling writing results in a selected data file format (for now : Paraview or GMSH) . . . 38

module_visuclusters_gmsh
: Contains methods enabling writing results in file specifically formatted for GMSH . . . 42

module_visuclusters_paraview
: Contains methods enabling writing results in file specifically formatted for Paraview . . . 47

module_visuclusters_structure
: Contains useful data structures . . . 52

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

module_structure::type_clustering_param 55
module_structure::type_clusters 55
module_structure::type_data 56
module_visuclusters_structure::type_params 57
module_structure::type_points 58
module_teste_clusters::type_test 58

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

module_calcul.f90 61
module_decoupe.f90 61
module_embed.f90 62
module_entree.f90 62
module_MPI.f90 63
module_solve.f90 63
module_sortie.f90 64
module_sparse.f90 64
module_structure.f90 64
module_teste_clusters.f90 65
module_visuclusters.f90 65
module_visuclusters_gmsh.f90 66
module_visuclusters_paraview.f90 66
module_visuclusters_structure.f90 67

Chapter 5

Module Documentation

5.1 module_calcul Module Reference

Contains the spectral clustering method and methods that computes affinity parameters for kernels and overlapping.

Functions/Subroutines

- subroutine get_sigma (partitioned_data, sigma)

  Computes the affinity parameter $sigma$.

- subroutine get_sigma_interface (proc_id, partitioned_data, sigma, bounds, partitioning, epsilon)

  Computes the affinity parameter $sigma$ for the interface.

- double precision function, dimension(partitioned_data%nb, partitioned_data%nb) poly_kernel (partitioned↩ _data, gam, delta)
- double precision function, dimension(partitioned_data%nb, partitioned_data%nb) gaussian_kernel (partitioned_data, sigma)
- subroutine apply_kernel_k_means (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, clust_↩ param)
- subroutine apply_spectral_clustering (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, sigma, clust_param)
- subroutine mean_shift (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, bandwidth)

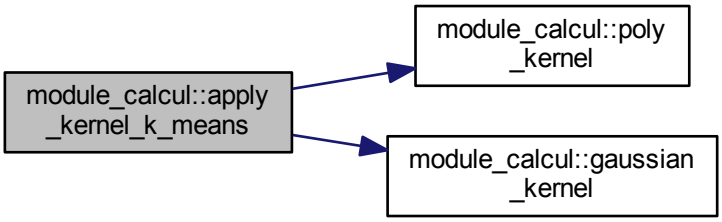
5.1.1 Detailed Description

Contains the spectral clustering method and methods that computes affinity parameters for kernels and overlapping.

5.1.2 Function/Subroutine Documentation

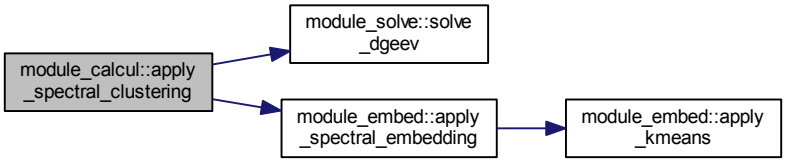
5.1.2.1 subroutine module_calcul::apply_kernel_k_means (integer *proc_id,* integer *nb_clusters_max,* integer *nb_clusters_opt,* type(**type_data**) *partitioned_data,* type(**type_clustering_param**) *clust_param*)

Here is the call graph for this function:



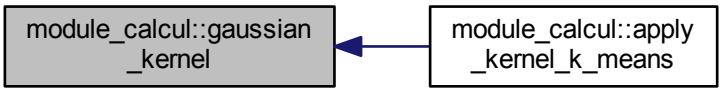
5.1.2.2 subroutine module_calcul::apply_spectral_clustering (integer *proc_id,* integer *nb_clusters_max,* integer *nb_clusters_opt,* type(**type_data**) *partitioned_data,* double precision *sigma,* type(**type_clustering_param**) *clust_param*)

Here is the call graph for this function:



5.1.2.3 double precision function, dimension(partitioned_data%nb,partitioned_data%nb) module_calcul::gaussian_kernel (type(**type_data**) *partitioned_data,* double precision *sigma*)

Here is the caller graph for this function:



5.1.2.4 subroutine module_calcul::get_sigma (type(type_data) *partitioned_data,* double precision *sigma*)

Computes the affinity parameter $sigma$.

The Gaussian affinity matrix is widely used and depends on a free parameter σ. It is known that this parameter affects the results in spectral clustering and spectral embedding. With an assumption that the p dimensionnal data set S composed of n points is isotropic enough, this data set is included in a p dimensional box bounded by D_{max} the largest distance between pairs of points in S :

$$D_{max} = \max_{1 \leq i,j \leq n} \|x_i - x_j\| \tag{5.1}$$

So a reference distance noted σ could be defined. This distance represents the case of an uniform distribution in the sense that all pair of points are seprated by the same distance σ in the box of edge size D_{max} :

$$\sigma = \frac{D_{max}}{n^{1/p}} \tag{5.2}$$

This function is used to compute this parameter on one domain. The algorithm is simple :

1. **Find the maximum distance using two nested loops over the points in the domain**
2. **Divide it by the p^{th} root of n**

Parameters

in	*partitioned_data*	the partitioned data for computing
out	*sigma*	the affinity parameter

Here is the caller graph for this function:

5.1.2.5 subroutine module_calcul::get_sigma_interface (integer *proc_id,* type(type_data) *partitioned_data,* double precision *sigma,* double precision, dimension(:,:,:), pointer *bounds,* integer, dimension(:), pointer *partitioning,* double precision *epsilon*)

Computes the affinity parameter $sigma$ for the interface.

This method is useful when the partitioning is made by interface. Because, the domain defining the interface has a volume whose topology changes drastically, a specific computation of σ has to be made.

Deprecated Use get_sigma() instead

Parameters

in	*partitioned_data*	the partitioned data for computing

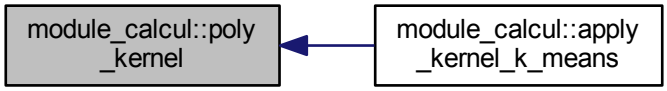
in	*bounds*	the intervals along each dimension representing the bounds of each partition
in	*epsilon*	the slice thickness
in	*proc_id*	the processus identifier
in	*partitioning*	the partitionning (number of processors along each dimension)
out	*sigma*	the affinity parameter

Here is the call graph for this function:

5.1.2.6 subroutine module_calcul::mean_shift (integer *proc_id,* integer *nb_clusters_max,* integer *nb_clusters_opt,* type(type_data) *partitioned_data,* integer *bandwidth*)

5.1.2.7 double precision function, dimension(partitioned_data%nb,partitioned_data%nb) module_calcul::poly_kernel (type(type_data) *partitioned_data,* double precision *gam,* double precision *delta*)

Here is the caller graph for this function:



5.2 module_decoupe Module Reference

Contains methods enabling the partitionning and the groupping of the data.

Functions/Subroutines

- subroutine partition_data (data, epsilon, nb_proc, coord_min, coord_max, partitioning, points_by_domain, assignements, bounds)

  Partitions the data into subdomains for a latter processing by the slaves.
- subroutine define_bounds (data, coord_min, coord_max, bounds, partitioning, epsilon, nb_proc)

  Defines the bounds of each subdomain.
- subroutine define_domains (nb_proc, data, domains, bounds, partitioning)

  Defines the different subdomains.
- subroutine partition_with_interface (nb_proc, data, points_by_domain, assignements, domains, epsilon)

Partitions the data using interface.

- subroutine partition_with_overlapping (nb_proc, data, points_by_domain, assignements, domains)

  Partitions the data using overlapping.

- subroutine group_clusters (nb_clusters, points_by_cluster, cluster_map, data)

  Groups the clusters and removes duplicates from the set of found clusters.

5.2.1 Detailed Description

Contains methods enabling the partitionning and the groupping of the data.

5.2.2 Function/Subroutine Documentation

5.2.2.1 subroutine module_decoupe::define_bounds (type(type_data) *data,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max,* double precision, dimension(:,:,:), pointer *bounds,* integer, dimension(:), pointer *partitioning,* double precision *epsilon,* integer *nb_proc*)

Defines the bounds of each subdomain.

The output *bounds* has to be interpreted as follows:

1. The first index corresponds to the dimensions
2. The second index corresponds to the partitioning along the dimension
3. The third index corresponds to the extrema of the bounds

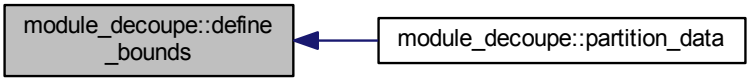
Note

The bounds are composed of two points.
In case of overlapping, the bounds overlapped each others.

Parameters

in	*data*	the entire data for computing
in	*epsilon*	the slice thickness
in	*nb_proc*	the number of processors used
in	*partitioning*	the partitionning (number of processors along each dimension)
in,out	*coord_max*	the maxima along each dimension of the data (coordinates)
in,out	*coord_min*	the minima along each dimension of the data (coordinates)
out	*bounds*	the intervals along each dimension representing the bounds of each partition

Here is the caller graph for this function:



5.2.2.2 subroutine module_decoupe::define_domains (integer *nb_proc,* type(type_data) *data,* double precision, dimension(:,:,:), pointer *domains,* double precision, dimension(:,:,:), pointer *bounds,* integer, dimension(:), pointer *partitioning*)

Defines the different subdomains.

The output *domains* has to be interpreted as follows:

1. The first index corresponds to the domain id
2. The second index corresponds to the dimensions
3. The third index corresponds to the extrema of the bounds

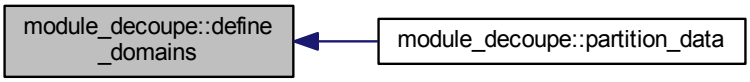
See also

define_bounds()

Parameters

in	*data*	the entire data for computing
in	*bounds*	the intervals along each dimension representing the bounds of each partition
in	*nb_proc*	the number of processors used
in	*partitioning*	the partitionning (number of processors along each dimension)
out	*domains*	the domains constructed from the bounds

Here is the caller graph for this function:



5.2.2.3 subroutine module_decoupe::group_clusters (integer *nb_clusters,* integer, dimension(:), pointer *points_by_cluster,* integer, dimension(:,:), pointer *cluster_map,* type(type_data) *data*)

Groups the clusters and removes duplicates from the set of found clusters.

The method operates on all the computed clusters and when an intersection is found between two of them (at least one point in common), they are melted.

Parameters

in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in,out	*cluster_map*	the cluster indices and the number of points in each cluster
in,out	*points_by_↩ cluster*	the number of points in each cluster

out	*data*	the entire data for computing

5.2.2.4 subroutine module_decoupe::partition_data (type(type_data) *data,* double precision *epsilon,* integer *nb_proc,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max,* integer, dimension(:), pointer *partitioning,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *assignements,* double precision, dimension(:,:,:), pointer *bounds*)

Partitions the data into subdomains for a latter processing by the slaves.

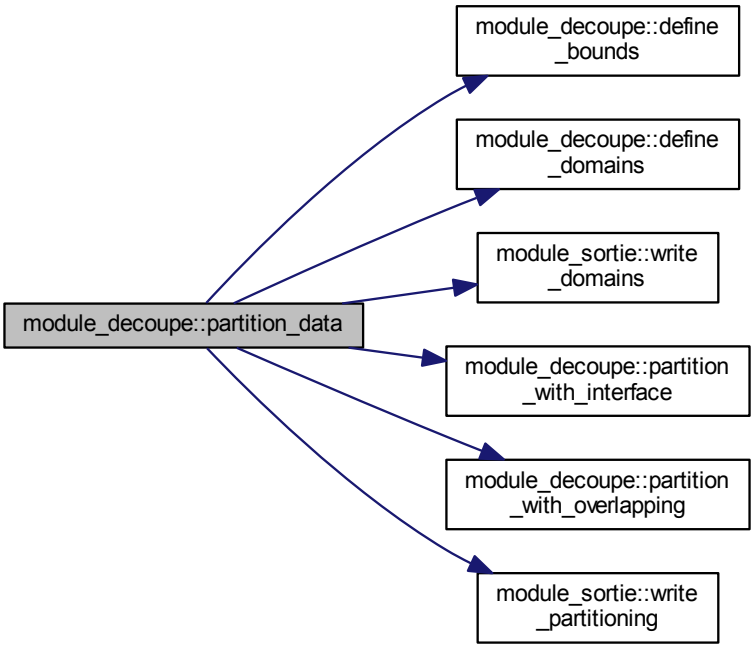
The following is performed :

1. The bounds of each domain are defined (see define_bounds()).
2. The domains are defined using the bounds (see define_domains()).
3. The domains are written in a dedicated file (see write_domains()).
4. The data is partitioned using interface or overlapping (see partition_with_interface() and partition_with_↩ overlapping()).
5. The partitioning is written in dedicated files (see write_partitioning()).

Parameters

in	*data*	the entire data for computing
in	*epsilon*	the slice thickness
in	*nb_proc*	the number of processors used
in	*partitioning*	the partitionning (number of processors along each dimension)
in,out	*coord_max*	the maxima along each dimension of the data (coordinates)
in,out	*coord_min*	the minima along each dimension of the data (coordinates)
out	*bounds*	the intervals along each dimension representing the bounds of each partition
out	*assignements*	the assignement of each point in a partition
out	*points_by_↩ domain*	the number of points in each partition

Here is the call graph for this function:



5.2.2.5 subroutine module_decoupe::partition_with_interface (integer *nb_proc,* type(type_data) *data,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *assignements,* double precision, dimension(:,:,:), pointer *domains,* double precision *epsilon*)

Partitions the data using interface.

The method partitions the entire data by defining which point belongs to which domain. It consists of a loop on all the point in data set. Then it "fills" the domains one after another using a nested loop. When a point does not fit the bounds define in the input *domains*, the algorithm switch to the next domain. Finally, an extra domain is defined (the interface) which corresponds to the area around the bounds with a predefined slice thickness.

See also

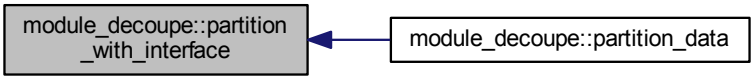
partition_with_overlapping()

Parameters

in	*data*	the entire data for computing
in	*domains*	the domains constructed from the bounds
in	*epsilon*	the slice thickness
in	*nb_proc*	the number of processors used
out	*assignements*	the assignement of each point in a partition

out	*points_by_↩ domain*	the number of points in each partition

Here is the caller graph for this function:



5.2.2.6 subroutine module_decoupe::partition_with_overlapping (integer *nb_proc,* type(type_data) *data,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *assignements,* double precision, dimension(:,:,:), pointer *domains*)

Partitions the data using overlapping.

The method partitions the entire data by defining which point belongs to which domain. It consists of two nested loop. The first one on the points and the second one on the domains (ie the processes). For each point, it checks if it fits the bounds defined in the input *domains* (and that for each domain) and in that case add to it.

Note

Some points will be present in different domains.

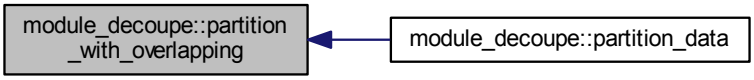
See also

partition_with_interface()

Parameters

in	*data*	the entire data for computing
in	*domains*	the domains constructed from the bounds
in	*nb_proc*	the number of processors used
out	*assignements*	the assignement of each point in a partition
out	*points_by_↩ domain*	the number of points in each partition

Here is the caller graph for this function:



5.3 module_embed Module Reference

Contains K-means and spectral embedding algorithms.

Functions/Subroutines

- subroutine apply_spectral_embedding (nb_clusters, n, Z, A, ratio, clusters, clusters_centers, points_by_↩ clusters, clusters_energies, nb_info, proc_id, ratio_moy, ratio_rij, ratio_rii)

  Computes the clusters using eigen vector matrix.
- subroutine apply_kmeans (dim, nb_points, nb_clusters, nb_iter_max, nb_iter, points, clusters, clusters_↩ centers, points_by_clusters, clusters_energies, proc_id)

  Implements K-Means algorithm (required by spectral clustering and Kernel K-Means methods) The algorithm works as follows:

5.3.1 Detailed Description

Contains K-means and spectral embedding algorithms.

5.3.2 Function/Subroutine Documentation

5.3.2.1 subroutine module_embed::apply_kmeans (integer *dim,* integer *nb_points,* integer *nb_clusters,* integer *nb_iter_max,* integer *nb_iter,* double precision, dimension (dim, nb_points) *points,* integer, dimension (nb_points) *clusters,* double precision, dimension (dim, nb_clusters) *clusters_centers,* integer, dimension (nb_clusters) *points_by_clusters,* double precision, dimension (nb_clusters) *clusters_energies,* integer *proc_id*)

Implements K-Means algorithm (required by spectral clustering and Kernel K-Means methods) The algorithm works as follows:

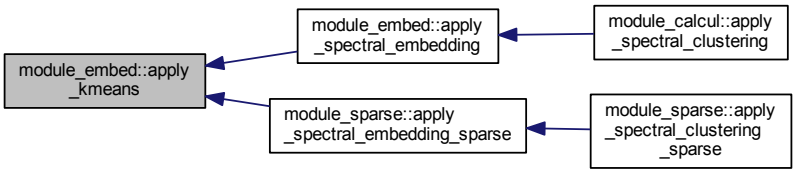
1. Choose *nb_clusters* starting points as cluster centers randomly
2. For each point in the data set, find the minimum distance from a cluster center and attach it to the corresponding cluster
3. Compute the density center of each cluster
4. Stop if the density centers are similar to the cluster centers
5. Select the density centers as new cluster centers and start from the beginning

Parameters

in	*points*	the points
in	*dim*	the number of spatial dimensions
in	*dim*	the number of spatial dimensions
in	*dim*	
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_iter_max*	the maximum number of iterations
in	*nb_points*	the number of points
in	*nb_points*	the number of points
in,out	*clusters_centers*	the cluster centers
	proc_id	the processus identifier
out	*clusters_↩ energies*	the cluster energies

out	*nb_iter*	the number of iterations taken
out	*clusters*	indicates which cluster each point belongs to
out	*points_by_↩ clusters*	the number of points in each cluster

Here is the caller graph for this function:



5.3.2.2 subroutine module_embed::apply_spectral_embedding (integer *nb_clusters,* integer *n,* double precision, dimension(:,:), pointer *Z,* double precision, dimension(:,:), pointer *A,* double precision *ratio,* integer, dimension(:), pointer *clusters,* double precision, dimension(:,:), pointer *clusters_centers,* integer, dimension(:), pointer *points_by_clusters,* double precision, dimension(:), pointer *clusters_energies,* integer *nb_info,* integer *proc_id,* double precision *ratio_moy,* double precision *ratio_rij,* double precision *ratio_rii*)

Computes the clusters using eigen vector matrix.

The first part of the method performs the following:

1. Extract the *nb_clusters* first columns of the eigen vector matrix (corresponding to the highest eigen values)
2. Normalize the matrix and transposes it
3. Apply K-Means on it to find the clusters

Then it operates a quality measurement on the found clusters. It computes the sum of the ratios that indicate if the number of clusters is optimal. The lower this sum is, the best is the number of clusters. This method also compute the number of clusters that have at least one point and that have an internal affinity greater than zero.

Note

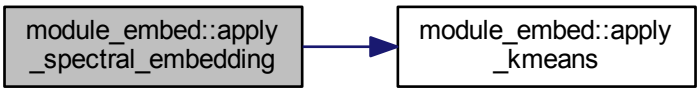
We refer you to the article *"On a strategy for Spectral Clustering with parallel computation"* for a better understanding.

Parameters

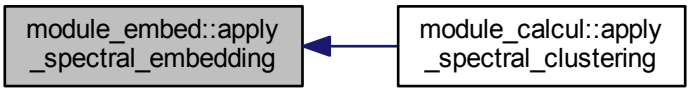
in	*A*	the affinity matrix
in	*Z*	the matrix of eigen vectors
in	*n*	
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*proc_id*	the processus identifier

out	*clusters_centers*	the cluster centers
out	*clusters_energies*	the cluster energies
out	*ratio*	the sum of the ratios between Frobenius norm of the off-diagonal and the diagonal blocks of the normalized affinity matrix
out	*ratio_moy*	
out	*ratio_rii*	the sum of the denominator of each ratio
out	*ratio_rij*	the sum of the numerators of each ratio
out	*nb_info*	the reduced number of clusters (?)
out	*clusters*	indicates which cluster each point belongs to
out	*points_by_clusters*	the number of points in each cluster

Here is the call graph for this function:



Here is the caller graph for this function:



5.4 module_entree Module Reference

Contains methods enabling data and parameter file reading.

Functions/Subroutines

- subroutine help

  Displays help on used formats and keywords for param.in file.
- subroutine read_params (data, epsilon, coord_min, coord_max, nb_proc, partitioning, input_file, sigma, nb_clusters_max, list_nb_clusters, clust_param)

  Reads a file in which there is the whole information on the data.
- subroutine read_coordinates_data (input_file, data, coord_min, coord_max)

  Reads data written in coordinates format.
- subroutine read_picture_data (input_file, data, coord_min, coord_max)

  Reads data written in picture format.

- subroutine read_geometric_data (input_file, data, coord_min, coord_max)

  Reads data written in geometric format.
- subroutine read_threshold_data (input_file, data, coord_min, coord_max)

  Reads data written in threshold format.
- subroutine assign_picture_array (data)

  Puts the index number of the pixels into an array.

5.4.1 Detailed Description

Contains methods enabling data and parameter file reading.

5.4.2 Function/Subroutine Documentation

5.4.2.1 subroutine module_entree::assign_picture_array (type(type_data) *data*)

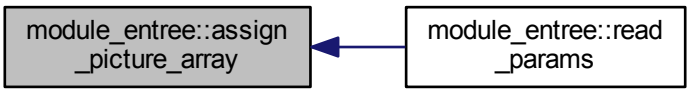
Puts the index number of the pixels into an array.

Each point of the data set is mapped to an array corresponding to an image. Thus, instead of one index for accessing one point, two will be used (row and column).

Parameters

in,out	*data*	the entire data for computing

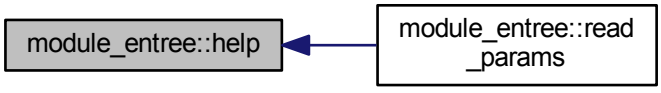
Here is the caller graph for this function:



5.4.2.2 subroutine module_entree::help ( )

Displays help on used formats and keywords for *param.in* file.

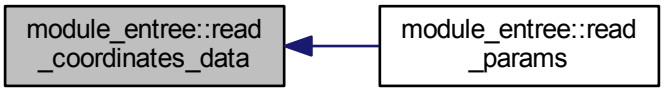
Here is the caller graph for this function:



5.4.2.3 subroutine module_entree::read_coordinates_data (character (len=30) *input_file,* type(type_data) *data,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max*)

Reads data written in coordinates format.

Here is the caller graph for this function:



5.4.2.4 subroutine module_entree::read_geometric_data (character (len=30) *input_file,* type(type_data) *data,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max*)

Reads data written in geometric format.

This type of file starts with two *integers* separated by a blank space: the first one corresponds to the dimension of the image and the second one the number of attributes (typically, the attributes could be the color channel intensities). The following line is composed of two *integers* separated by a blank space: the first one corresponds to the number of pixels in a column and the second one to the number of pixels in a row. Then, next lines are composed of *double* numbers separated by blank spaces. Each of these lines corresponds to the value of the attributes for each pixel. The coordinates of the pixels are implicit as they are ordered by rows.

Note

The attributes and the coordinates are stored in the field type_data::coord and so the position AND the color will be taken into account.
The partitioning is made according to the coordinates

See also

assign_picture_array(), read_coordinates_data(), read_threshold_data(), read_picture_data

Parameters

in	*input_file*	the name of the text file where input data is written
in,out	*data*	the entire data for computing
out	*coord_max*	the maxima along each dimension of the data (coordinates)
out	*coord_min*	the minima along each dimension of the data (coordinates)

Here is the caller graph for this function:

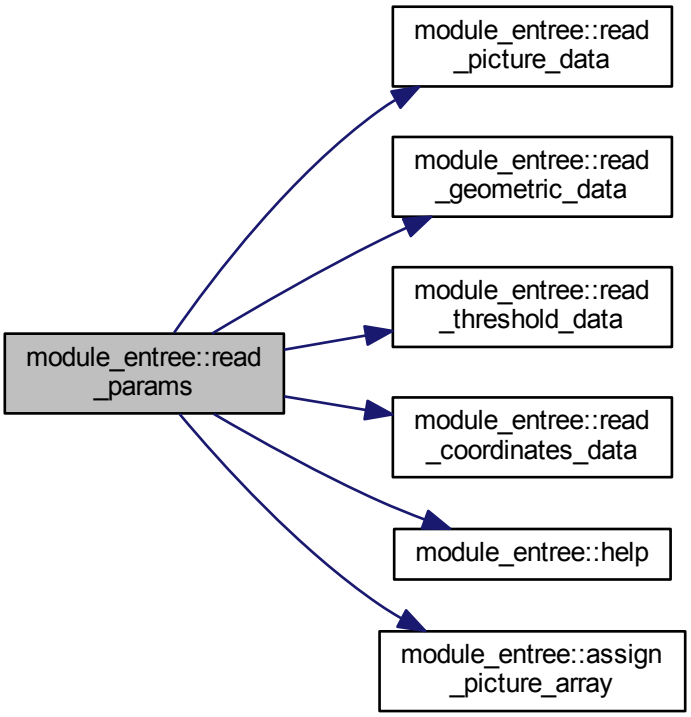
5.4.2.5 subroutine module_entree::read_params (type(type_data) *data,* double precision *epsilon,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max,* integer *nb_proc,* integer, dimension(:), pointer *partitioning,* character (len=30) *input_file,* double precision *sigma,* integer *nb_clusters_max,* integer, dimension(:), pointer *list_nb_clusters,* type(type_clustering_param) *clust_param*)

Reads a file in which there is the whole information on the data.

Parameters

in	*nb_proc*	the number of processors used
in,out	*data*	the entire data for computing
out	*input_file*	the name of the text file where input data is written
out	*coord_max*	the maxima along each dimension of the data (coordinates)
out	*coord_min*	the minima along each dimension of the data (coordinates)
out	*epsilon*	the slice thickness
out	*sigma*	the affinity parameter
out	*nb_clusters_max*	the maximum number of clusters
out	*list_nb_clusters*	the imposed numbers of clusters list for testing purpose (useless)
out	*partitioning*	the partitionning (number of processors along each dimension)

Here is the call graph for this function:



5.4.2.6 subroutine module_entree::read_picture_data (character (len=30) *input_file,* type(type_data) *data,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max*)

Reads data written in picture format.

This type of file starts with two *integers* separated by a blank space: the first one corresponds to the dimension of the image and the second one the number of attributes (typically, the attributes could be the color channel intensities). The following line is composed of two *integers* separated by a blank space: the first one corresponds to the number of pixels in a column and the second one to the number of pixels in a row. Then, next lines are composed of *double* numbers separated by blank spaces. Each of these lines corresponds to the value of the attributes for each pixel. The coordinates of the pixels are implicit as they are ordered by rows.

Note

The attributes are stored in the field type_data::coord and so ONLY the color will be taken into account. The outputs *coordmax* and *coordmin* are computed according to the position of the pixels.

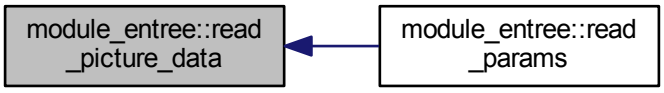
See also

assign_picture_array(), read_coordinates_data(), read_threshold_data(), read_geometric_data

Parameters

in	*input_file*	the name of the text file where input data is written
in,out	*data*	the entire data for computing
out	*coord_max*	the maxima along each dimension of the data (coordinates)
out	*coord_min*	the minima along each dimension of the data (coordinates)

Here is the caller graph for this function:



5.4.2.7 subroutine module_entree::read_threshold_data (character (len=30) *input_file,* type(type_data) *data,* double precision, dimension(:), pointer *coord_min,* double precision, dimension(:), pointer *coord_max*)

Reads data written in threshold format.

This type of file starts with two *integers* separated by a blank space: the first one corresponds to the dimension of the image and the second one the number of attributes (typically, the attributes could be the color channel intensities). The following line is composed of two *integers* separated by a blank space: the first one corresponds to the number of pixels in a column and the second one to the number of pixels in a row. Then, next lines are composed of *double* numbers separated by blank spaces. Each of these lines corresponds to the value of the attributes for each pixel. The coordinates of the pixels are implicit as they are ordered by rows.

Note

The attributes are stored in the field type_data::coord and so ONLY the color will be taken into account The outputs *coordmax* and *coordmin* are computed according to the color of the pixels.

See also

assign_picture_array(), read_picture_data(), read_coordinates_data(), read_geometric_data

Parameters

in	*input_file*	the name of the text file where input data is written
in,out	*data*	the entire data for computing
out	*coord_max*	the maxima along each dimension of the data (coordinates)
out	*coord_min*	the minima along each dimension of the data (coordinates)

Here is the caller graph for this function:

5.5 module_mpi Module Reference

Contains methods related to parallelism.

Functions/Subroutines

- subroutine send_partitioning (nb_proc, data, points_by_domain, ddat, partitioned_data)

  Sends the partitionning.
- subroutine receive_partitioning (proc_id, partitioned_data)

  Receives the partitionning.
- subroutine receive_number_clusters (nb_proc, nb_clusters, points_by_domain, partitioned_data, array_clust)

  Receives the number of clusters.
- subroutine send_number_clusters (proc_id, partitioned_data)

  Sends the number of clusters.
- subroutine send_clusters (proc_id, partitioned_data)

  Sends the clusters.
- subroutine receive_clusters (nb_proc, nb_clusters, points_by_domain, ddat, partitioned_data, cluster_map, array_clust, points_by_cluster)

  Receives the clusters.

5.5.1 Detailed Description

Contains methods related to parallelism.

5.5.2 Function/Subroutine Documentation

5.5.2.1 subroutine module_mpi::receive_clusters (integer *nb_proc,* integer *nb_clusters,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *ddat,* type(type_data) *partitioned_data,* integer, dimension(:,:), pointer *cluster_map,* type(type_clusters), dimension(:), pointer *array_clust,* integer, dimension(:), pointer *points_by_cluster*)

Receives the clusters.

This method receives the computed clusters in each domain from the slave processes.

Note

It has to be called by the master process.

See also

send_clusters()

Parameters

in	*array_clust*	the number of clusters and elements per cluster computed by each processor
in	*partitioned_data*	the partitioned data for computing
in	*assignements*	the assignement of each point in a partition
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_proc*	the number of processors used
in	*points_by_↩ domain*	the number of points in each partition
out	*cluster_map*	the cluster indices and the number of points in each cluster
out	*points_by_↩ cluster*	the number of points in each cluster

5.5.2.2 subroutine module_mpi::receive_number_clusters (integer *nb_proc,* integer *nb_clusters,* integer, dimension(:), pointer *points_by_domain,* type(type_data) *partitioned_data,* type(type_clusters), dimension(:), pointer *array_clust*)

Receives the number of clusters.

This method receives from the slave processes the number of clusters in each domain and the number of elements in each cluster.

Note

It has to be called by the master process.

See also

send_number_clusters()

Parameters

in	*partitioned_data*	the partitioned data for computing
in	*nb_proc*	the number of processors used
in	*points_by_↩ domain*	the number of points in each partition
out	*array_clust*	the number of clusters and elements per cluster computed by each processor
out	*nb_clusters*	the number of clusters
out	*nb_clusters*	the number of clusters
out	*nb_clusters*	the number of clusters

5.5.2.3 subroutine module_mpi::receive_partitioning (integer *proc_id,* type(type_data) *partitioned_data*)

Receives the partitionning.

This method receives from the master process the number of points, the dimension and all the points of the dedicated domain.

Note

It has to be called by a slave process

See also

send_partitioning(), partition_with_interface(), partition_with_overlapping()

Parameters

in	*proc_id*	the processus identifier
out	*partitioned_data*	the partitioned data for computing

5.5.2.4 subroutine module_mpi::send_clusters (integer *proc_id,* type(type_data) *partitioned_data*)

Sends the clusters.

This method sends the computed clusters to the master process.

Note

It has to be called by a slave process.

See also

receive_clusters()

Parameters

in	*partitioned_data*	the partitioned data for computing
in	*proc_id*	the processus identifier

5.5.2.5 subroutine module_mpi::send_number_clusters (integer *proc_id,* type(type_data) *partitioned_data*)

Sends the number of clusters.

This method sends to the master process the number of clusters and the number of elements in each cluster.

Note

It has to be called by a slave process.

See also

receive_number_clusters()

Parameters

in	*partitioned_data*	the partitioned data for computing
in	*proc_id*	the processus identifier

5.5.2.6 subroutine module_mpi::send_partitioning (integer *nb_proc,* type(type_data) *data,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *ddat,* type(type_data) *partitioned_data*)

Sends the partitionning.

This method sends to each slave process the number of points, the dimension and all the points of the dedicated domain. Then, it creates

Note

It has to be called by the master process

See also

receive_partitioning(), partition_with_interface(), partition_with_overlapping()

Parameters

in	*assignements*	the assignement of each point in a partition
in	*nb_proc*	the number of processors used
in	*points_by_↩ domain*	the number of points in each partition
in,out	*data*	the entire data for computing
out	*partitioned_data*	the partitioned data for computing

5.6 module_solve Module Reference

Contains methods from Lapack library dealing with eigen values computing.

Functions/Subroutines

- subroutine solve_dgeevx (N, A, VR, WR)
- subroutine solve_dgeev (N, A, VR, WR)
- subroutine solve_dsyev (N, A, W, LWORK)
- subroutine solve_dsyevr (k, N, A, Z, LWORK, LIWORK, W, M)
- subroutine solve_dsyevx (k, N, A, Z, LWORK, LIWORK, W, M)

5.6.1 Detailed Description

Contains methods from Lapack library dealing with eigen values computing.

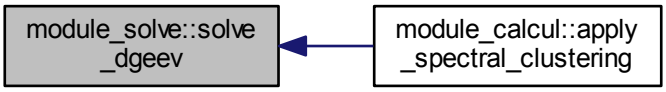
5.6.2 Function/Subroutine Documentation

5.6.2.1 subroutine module_solve::solve_dgeev (integer *N*, double precision, dimension(:,:), pointer *A*, double precision, dimension(:,:), pointer *VR*, double precision, dimension(:), pointer *WR*)

Parameters

in	*A*	the affinity matrix
in	*N*	
out	*VR*	
out	*WR*	

Here is the caller graph for this function:



5.6.2.2 subroutine module_solve::solve_dgeevx (integer *N,* double precision, dimension(:,:), pointer *A,* double precision, dimension(:,:), pointer *VR,* double precision, dimension(:), pointer *WR*)

Parameters

A	the affinity matrix
VR	
WR	
N	

5.6.2.3 subroutine module_solve::solve_dsyev (integer *N,* double precision, dimension(n,n) *A,* double precision, dimension(n) *W,* integer *LWORK*)

Parameters

A	the affinity matrix
W	
LWORK	
N	

5.6.2.4 subroutine module_solve::solve_dsyevr (integer *k,* integer *N,* double precision, dimension(n,n) *A,* double precision, dimension(n,n) *Z,* integer *LWORK,* integer *LIWORK,* double precision, dimension(n) *W,* integer *M*)

Parameters

A	the affinity matrix
Z	the matrix of eigen vectors
W	
k	
LIWORK	
LWORK	
M	
N	

5.6.2.5 subroutine module_solve::solve_dsyevx (integer *k,* integer *N,* double precision, dimension(n,n) *A,* double precision, dimension(n,n) *Z,* integer *LWORK,* integer *LIWORK,* double precision, dimension(n) *W,* integer *M*)

Parameters

A	the affinity matrix
Z	the matrix of eigen vectors
W	
k	
LIWORK	
LWORK	
M	
N	

5.7 module_sortie Module Reference

Contains methods enabling writing results in specific formatted files.

Functions/Subroutines

- subroutine write_domains (data, nb_proc, domains)

  Writes the file containing the domain definitions.
- subroutine write_partitioning (nb_proc, data, points_by_domain, assignements)

  Writes the files containing the partitionning.
- subroutine write_partial_clusters (proc_id, partitioned_data)

  Writes the file containing the clusters computed by the process.
- subroutine write_final_clusters (nb_clusters, points_by_cluster, cluster_map)

  Writes the files containing the final clusters after grouping.
- subroutine write_metadata (mesh, data, nb_proc, nb_clusters)

  Writes a file containing various information.

5.7.1 Detailed Description

Contains methods enabling writing results in specific formatted files.

5.7.2 Function/Subroutine Documentation

5.7.2.1 subroutine module_sortie::write_domains (type(type_data) *data,* integer *nb_proc,* double precision, dimension(:,:,:), pointer *domains*)

Writes the file containing the domain definitions.

This methods writes the domain definitions using the following formatting: each line contains the two point coordinates of a bound separated by the character "|".

Note

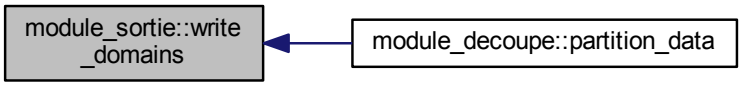
The written file is *fort.2*.S

Parameters

in	*data*	the entire data for computing

in	*domains*	the domains constructed from the bounds
in	*nb_proc*	the number of processors used

Here is the caller graph for this function:



5.7.2.2 subroutine module_sortie::write_final_clusters (integer *nb_clusters,* integer, dimension(:), pointer *points_by_cluster,* integer, dimension(:,:), pointer *cluster_map*)

Writes the files containing the final clusters after grouping.

Each file correspond to a cluster. The first number is the number of points in the cluster. Then all the following numbers are the indices of the points that belongs to the cluster.

Note

The written files are *cluster.final.x* with x the ids of the clusters.

Parameters

in	*cluster_map*	the cluster indices and the number of points in each cluster
in	*points_by_↩ cluster*	the number of points in each cluster
in,out	*nb_clusters*	the number of clusters
in,out	*nb_clusters*	the number of clusters
in,out	*nb_clusters*	the number of clusters

5.7.2.3 subroutine module_sortie::write_metadata (character (len=30) *mesh,* type(type_data) *data,* integer *nb_proc,* integer *nb_clusters*)

Writes a file containing various information.

The following information is written in *fort.3* file :

1. The data file name
2. The number of the points in the entire data set
3. The number of processes used
4. The partitioning mode (by interface or overlapping)
5. The number of clusters found
6. The data file format

In the case of a picture format: this extra information is written :

1. The image dimension

2. The image partitioning
3. The number of attributes
4. The number of steps (only in geometric format)

See also

read_metadata()

Parameters

in	*data*	the entire data for computing
in	*input_file*	the name of the text file where input data is written
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_proc*	the number of processors used

5.7.2.4 subroutine module_sortie::write_partial_clusters (integer *proc_id,* type(type_data) *partitioned_data*)

Writes the file containing the clusters computed by the process.

The file starts with the number of points in the domain and the number of dimensions separated by a blank space. The following lines are :

- **For coordinates format** : the coordinates of a point and the id of the cluster which it belongs to separated by a comma
- **For picture formats** : the id of the point and the id of the cluster which it belongs to separated by a comma

Note

The written file is *cluster.partiel.x* with x the id of the process

Parameters

in	*partitioned_data*	the partitioned data for computing
in	*proc_id*	the processus identifier

5.7.2.5 subroutine module_sortie::write_partitioning (integer *nb_proc,* type(type_data) *data,* integer, dimension(:), pointer *points_by_domain,* integer, dimension(:,:), pointer *assignements*)

Writes the files containing the partitionning.

Each file correspond to a domain. The first number is the number of points in the domain. Then the following lines are simply the coordinates of the points (in case of coordinates format) or the color of the pixels (in case of picture format) separated by blank spaces.

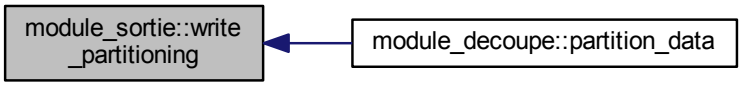
Note

The written files are *decoupe.x* with x the ids of the processes.

Parameters

in	*data*	the entire data for computing
in	*assignements*	the assignement of each point in a partition
in	*nb_proc*	the number of processors used
in	*points_by_↩ domain*	the number of points in each partition

Here is the caller graph for this function:



5.8 module_sparse Module Reference

Functions/Subroutines

- subroutine apply_spectral_clustering_sparse (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, sigma)

  Computes the clusters using spectral clustering algorithm using sparsity.
- subroutine apply_spectral_embedding_sparse (nb_clusters, n, Z, nnz, AS, IAS, JAS, ratio, clusters, clusters↩ _centers, points_by_clusters, clusters_energies, nb_info, proc_id, ratio_moy, ratio_rij, ratio_rii)

  Computes the ideal number of clusters using sparsity.
- subroutine compute_matvec_prod (A, IA, JA, X, Y, n, nnz)

  Computes the matrix vector product using sparsity.
- subroutine solve_arpack (A, IA, JA, dim, nnz, nb_clusters_max, W, Z)

5.8.1 Function/Subroutine Documentation

5.8.1.1 subroutine module_sparse::apply_spectral_clustering_sparse (integer *proc_id,* integer *nb_clusters_max,* integer *nb_clusters_opt,* type(type_data) *partitioned_data,* double precision *sigma*)

Computes the clusters using spectral clustering algorithm using sparsity.

See also

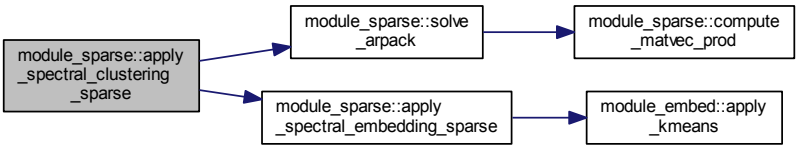
apply_spectral_clustering()

Parameters

in	*sigma*	the affinity parameter
in	*nb_clusters_max*	the maximum number of clusters
in	*nb_clusters_opt*	the optimal number of clusters

in	*proc_id*	the processus identifier
in,out	*partitioned_data*	the partitioned data for computing

Here is the call graph for this function:



5.8.1.2 subroutine module_sparse::apply_spectral_embedding_sparse (integer *nb_clusters,* integer *n,* double precision, dimension(:,:), pointer *Z,* integer *nnz,* double precision, dimension(:), pointer *AS,* integer, dimension(:), pointer *IAS,* integer, dimension(:), pointer *JAS,* double precision *ratio,* integer, dimension(:), pointer *clusters,* double precision, dimension(:,:), pointer *clusters_centers,* integer, dimension(:), pointer *points_by_clusters,* double precision, dimension(:), pointer *clusters_energies,* integer *nb_info,* integer *proc_id,* double precision *ratio_moy,* double precision *ratio_rij,* double precision *ratio_rii*)

Computes the ideal number of clusters using sparsity.

See also

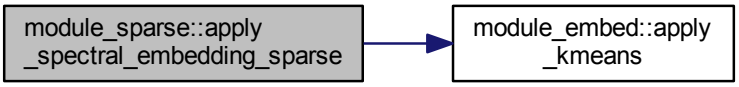
apply_spectral_embedding()

Parameters

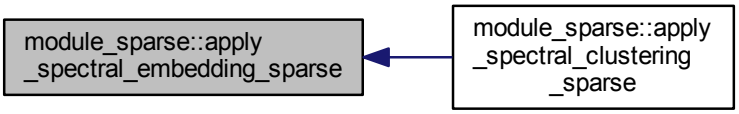
in	*Z*	the matrix of eigen vectors
in	*AS*	the affinity sparse matrix
in	*n*	
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nb_clusters*	the number of clusters
in	*nnz*	the number of non-zero coefficients
in	*proc_id*	the processus identifier
in	*IAS*	the row indices of the affinity matrix coefficients
in	*JAS*	the column indices of the affinity matrix coefficients
out	*clusters_centers*	the cluster centers
out	*clusters_↩ energies*	the cluster energies
out	*ratio*	the sum of the ratios between Frobenius norm of the off-diagonal and the diagonal blocks of the normalized affinity matrix
out	*ratio_moy*	
out	*ratio_rii*	the sum of the denominator of each ratio
out	*ratio_rij*	the sum of the numerators of each ratio
out	*nb_info*	the reduced number of clusters (?)

out	*clusters*	indicates which cluster each point belongs to
out	*points_by_↩ clusters*	the number of points in each cluster

Here is the call graph for this function:



Here is the caller graph for this function:



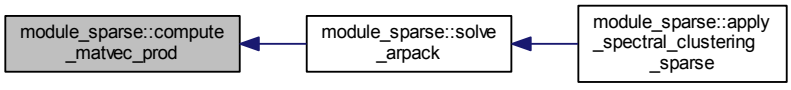
5.8.1.3 subroutine module_sparse::compute_matvec_prod (double precision, dimension(nnz), intent(in) *A,* integer, dimension(nnz), intent(in) *IA,* integer, dimension(nnz), intent(in) *JA,* double precision, dimension(n), intent(in) *X,* double precision, dimension(n), intent(out) *Y,* integer, intent(in) *n,* integer, intent(in) *nnz*)

Computes the matrix vector product using sparsity.

Parameters

in	*A*	the sparse matrix
in	*X*	the input vector
in	*n*	
in	*nnz*	the number of non-zero coefficients
in	*IA*	the row indices of the matrix coefficients
in	*JA*	the column indices of the matrix coefficients
out	*Y*	the resulting vector

Here is the caller graph for this function:

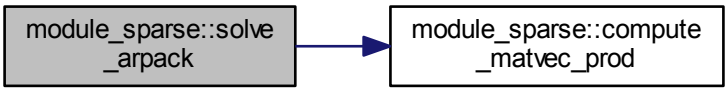


5.8.1.4 subroutine module_sparse::solve_arpack (double precision, dimension(:), intent(in) *A,* integer, dimension(:), intent(in) *IA,* integer, dimension(:), intent(in) *JA,* integer, intent(in) *dim,* integer, intent(in) *nnz,* integer, intent(in) *nb_clusters_max,* double precision, dimension(:), intent(out), pointer *W,* double precision, dimension(:,:), intent(out), pointer *Z*)

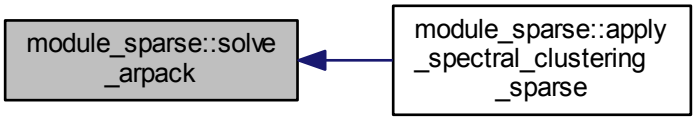
Parameters

in	*A*	the affinity sparse matrix
in	*dim*	the number of spatial dimensions
in	*dim*	the number of spatial dimensions
in	*dim*	
in	*nb_clusters_max*	the maximum number of clusters
in	*nnz*	the number of non-zero coefficients
in	*IA*	the row indices of the affinity matrix coefficients
in	*JA*	the column indices of the affinity matrix coefficients
out	*Z*	the matrix of eigen vectors
out	*W*	

Here is the call graph for this function:



Here is the caller graph for this function:



5.9 module_structure Module Reference

Contains structure types required by the different modules.

Data Types

- type type_clustering_param
- type type_clusters
- type type_data
- type type_points

5.9.1 Detailed Description

Contains structure types required by the different modules.

5.10 module_teste_clusters Module Reference

Data Types

- type type_test

Functions/Subroutines

- subroutine create_executable (test)

  Creates an executable called go for runcluster.
- subroutine create_test (test)

  Creates a test file.
- subroutine execute_test (test)

  Executes the script go and displays information on the console screen.
- subroutine create_data

  Generates an example of a raw geometric data file for testing purpose.

5.10.1 Function/Subroutine Documentation

5.10.1.1 subroutine module_teste_clusters::create_data ( )

Generates an example of a raw geometric data file for testing purpose.

5.10.1.2 subroutine module_teste_clusters::create_executable (type(type_test) *test*)

Creates an executable called *go* for runcluster.

Parameters

in	*test*	

5.10.1.3 subroutine module_teste_clusters::create_test (type(type_test) *test*)

Creates a test file.

Parameters

in	*test*	

5.10.1.4 subroutine module_teste_clusters::execute_test (type(type_test) *test*)

Executes the script *go* and displays information on the console screen.

Parameters

in	*test*	

5.11 module_visuclusters Module Reference

Contains methods enabling writing results in a selected data file format (for now : Paraview or GMSH)

Functions/Subroutines

- subroutine read_metadata (params)

  Reads a file containing metadata on the data and the computed clusters.
- subroutine write_partitioning (format_output, params)

  Writes the geometry of the partitioning (Gmsh or Paraview) and calls the eponym function.
- subroutine write_assignment (format_output, params)

  Initializes the file of the partitionning.
- subroutine write_partial_clusters (format_output, params)

  Writes the clusters before grouping by calling the corresponding method (Gmsh or Paraview)
- subroutine write_final_clusters (format_output, params)

  Writes the clusters after grouping by calling the corresponding method (Gmsh or Paraview)
- subroutine list_commands (format_output)

  Lists the commands related to Gmsh or Paraview depending on the input parameter.

5.11.1 Detailed Description

Contains methods enabling writing results in a selected data file format (for now : Paraview or GMSH)

5.11.2 Function/Subroutine Documentation

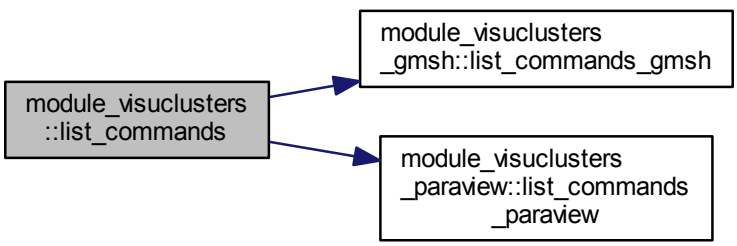
5.11.2.1 subroutine module_visuclusters::list_commands (character (len=30) *format_output*)

Lists the commands related to Gmsh or Paraview depending on the input parameter.

Parameters

in	*format_output*	the file format for visualization

Here is the call graph for this function:



5.11.2.2 subroutine module_visuclusters::read_metadata (type(type_params) *params*)

Reads a file containing metadata on the data and the computed clusters.

This function extracts the following information from the input *fort.3* file :

1. The data file name
2. The number of the points in the entire data set
3. The number of processes used
4. The partitioning mode (by interface or overlapping)
5. The number of clusters found
6. The data file format

In the case of a picture format: this extra information is written :

1. The image dimension
2. The image partitioning
3. The number of attributes
4. The number of steps (only in geometric format)

See also

write_metadata()

Parameters

in,out	*params*	the parameters defined in the *param.in* file

5.11.2.3 subroutine module_visuclusters::write_assignment (character (len=30) *format_output,* type(type_params) *params*)

Initializes the file of the partitionning.

This method extracts details on partitioning from the *decoupe.x* files.

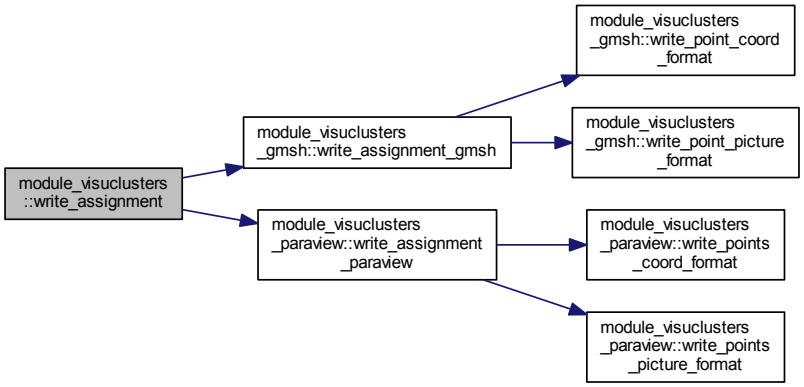
See also

module_calcul::write_partial_clusters()

Parameters

in	*params*	the parameters defined in the *param.in* file
in	*format_output*	the file format for visualization

Here is the call graph for this function:



5.11.2.4 subroutine module_visuclusters::write_final_clusters (character (len=30) *format_output*, type(type_params) *params*)

Writes the clusters after grouping by calling the corresponding method (Gmsh or Paraview)

This methods extracts details on computed clusters from *cluster.final.x* files.

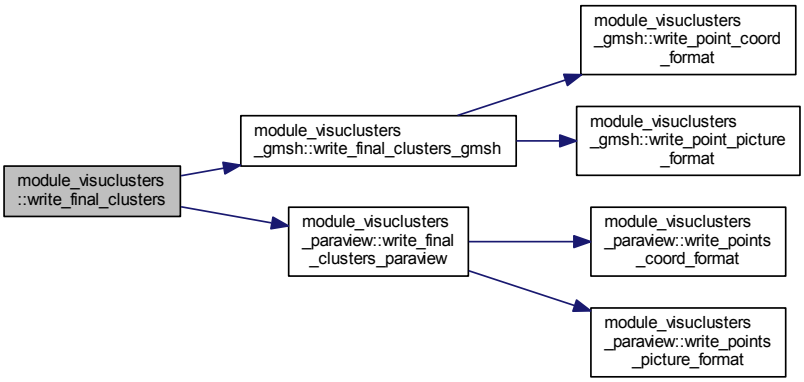
See also

module_calcul::write_final_clusters()

Parameters

in	*params*	the parameters defined in the *param.in* file
in	*format_output*	the file format for visualization

Here is the call graph for this function:



5.11.2.5 subroutine module_visuclusters::write_partial_clusters (character (len=30) *format_output,* type(type_params) *params*)

Writes the clusters before grouping by calling the corresponding method (Gmsh or Paraview)

This methods extracts details on computed clusters on each domain from *cluster.partiel.x* files.

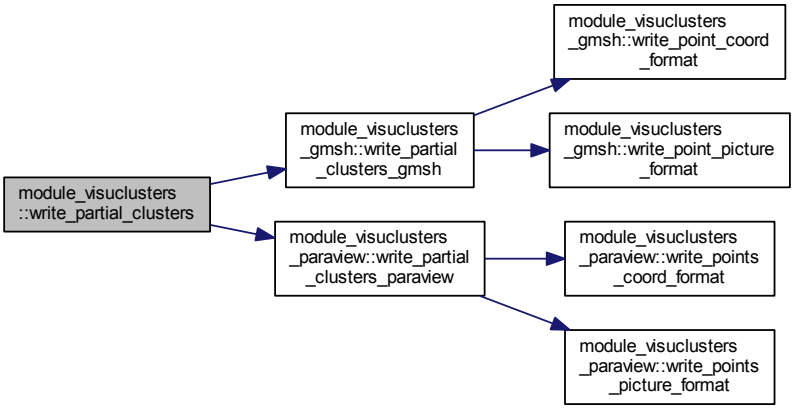
See also

module_calcul::write_partial_clusters()

Parameters

in	*params*	the parameters defined in the *param.in* file
in	*format_output*	the file format for visualization

Here is the call graph for this function:



5.11.2.6 subroutine module_visuclusters::write_partitioning (character (len=30) *format_output,* type(type_params) *params*)

Writes the geometry of the partitioning (Gmsh or Paraview) and calls the eponym function.

This methods extracts the domain definitions from the *fort.2* file.

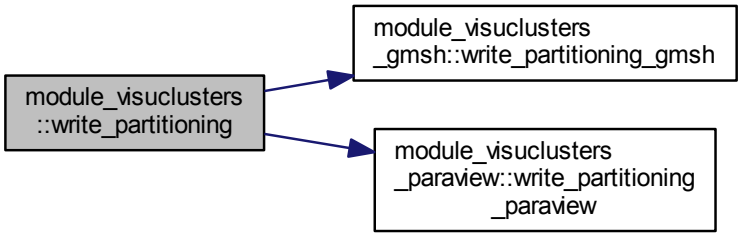
See also

module_calcul::write_partitioning()

Parameters

in	*params*	the parameters defined in the *param.in* file
in	*format_output*	the file format for visualization

Here is the call graph for this function:



5.12 module_visuclusters_gmsh Module Reference

Contains methods enabling writing results in file specifically formatted for GMSH.

Functions/Subroutines

- subroutine write_partitioning_gmsh (params)

  Writes the geometry partitioning for Gmsh visualization.
- subroutine write_assignment_gmsh (params)

  Initializes the file of the partitioning for Gmsh visualization.
- subroutine write_partial_clusters_gmsh (params)

  Writes the clusters before grouping for Gmsh visualization.
- subroutine write_final_clusters_gmsh (params)

  Writes the clusters after grouping for Gmsh visualization.
- subroutine write_point_coord_format (unit, dim, coords, id, k)

  Writes point coordinates from coordinates format for Gmsh visualization.
- subroutine write_point_picture_format (unit, params, id, k)

  Writes point coordinates from picture format for Gmsh visualization.
- subroutine list_commands_gmsh

  Lists the commands related to Gmsh visualization.

5.12.1 Detailed Description

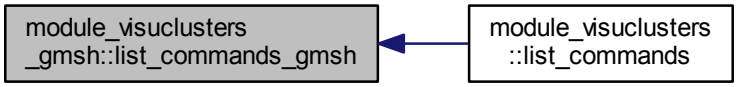
Contains methods enabling writing results in file specifically formatted for GMSH.

5.12.2 Function/Subroutine Documentation

5.12.2.1 subroutine module_visuclusters_gmsh::list_commands_gmsh ( )

Lists the commands related to Gmsh visualization.

Here is the caller graph for this function:



5.12.2.2 subroutine module_visuclusters_gmsh::write_assignment_gmsh (type(type_params) *params*)

Initializes the file of the partitioning for Gmsh visualization.

This method extracts details on partitioning from the *decoupe.x* files and writes them in *decoupe.visu* file.

Note

x is the identifier (number) of a process.

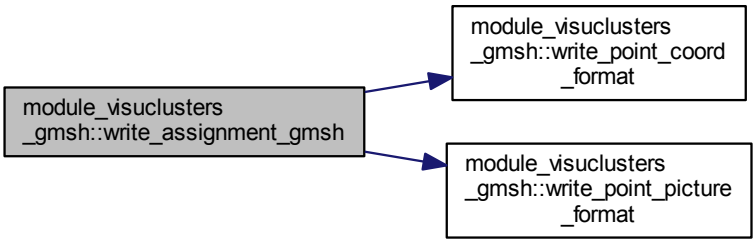
See also

write_partitioning()

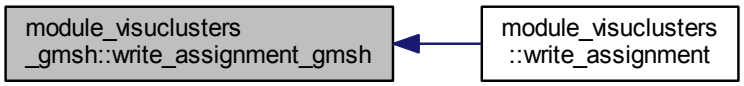
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.3 subroutine module_visuclusters_gmsh::write_final_clusters_gmsh (type(type_params) *params*)

Writes the clusters after grouping for Gmsh visualization.

This methods extracts details on computed clusters from *cluster.final.x* files and writes them in *cluster.final.visu* file.

Note

x is the identifier (number) of a cluster.

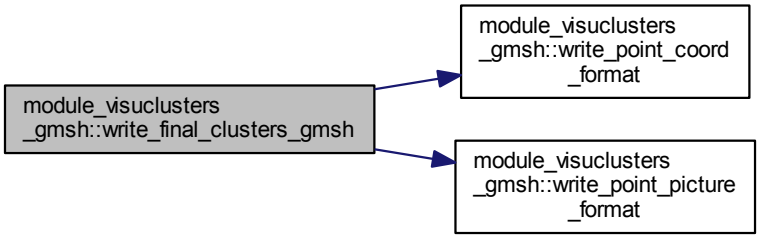
See also

module_calcul::write_final_clusters()

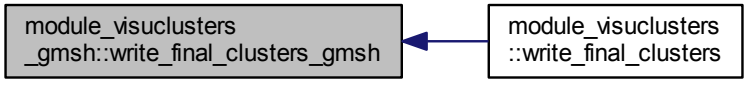
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.4 subroutine module_visuclusters_gmsh::write_partial_clusters_gmsh (type(type_params) *params*)

Writes the clusters before grouping for Gmsh visualization.

This methods extracts details on computed clusters on each domain from *cluster.partiel.x* files and writes them in corresponding *cluster.partiel.x.visu* files.

Note

x is the identifier (number) of a process.

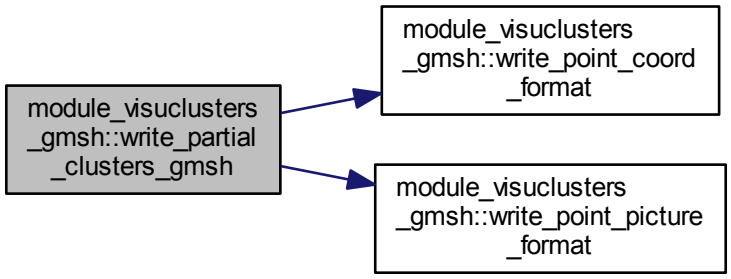
See also

module_calcul::write_partial_clusters()

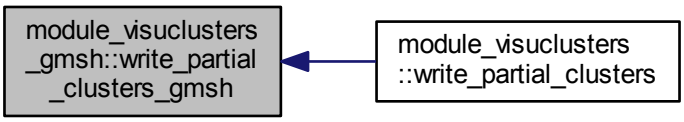
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.5 subroutine module_visuclusters_gmsh::write_partitioning_gmsh (type(type_params) *params*)

Writes the geometry partitioning for Gmsh visualization.

This methods extracts the domain definitions from the *fort.2* file and writes to *decoupe.geo* file, a source code file. We refer you to a Gmsh tutorial.

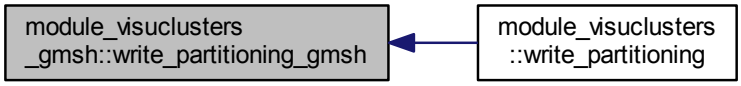
See also

module_calcul::write_partitioning()

Parameters

in	*params*	the parameters defined in the *param.in* file

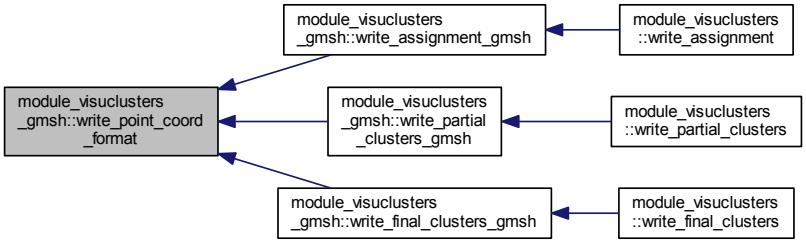
Here is the caller graph for this function:



5.12.2.6 subroutine module_visuclusters_gmsh::write_point_coord_format (integer *unit,* integer *dim,* double precision, dimension(:,:), pointer *coords,* integer *id,* integer *k*)

Writes point coordinates from coordinates format for Gmsh visualization.

Here is the caller graph for this function:



5.12.2.7 subroutine module_visuclusters_gmsh::write_point_picture_format (integer *unit,* type(type_params) *params,* integer *id,* integer *k*)

Writes point coordinates from picture format for Gmsh visualization.

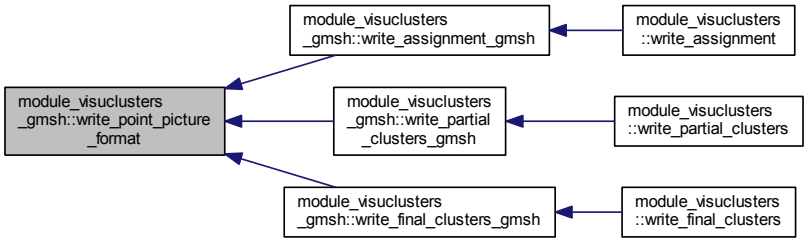
The following is written in the file :

1. **For 2D case**: *SP(x,y){id}*
2. **For 3D case**: *SP(x,y,z){id}*

Note

x,y and z are the coordinates and id is the identifier of the process

Here is the caller graph for this function:



5.13 module_visuclusters_paraview Module Reference

Contains methods enabling writing results in file specifically formatted for Paraview.

Functions/Subroutines

- subroutine write_partitioning_paraview (params)

  Writes the geometry partitioning for Paraview visualization.
- subroutine write_assignment_paraview (params)

  Initializes the file of the partitioning for Paraview visualization.
- subroutine write_partial_clusters_paraview (params)

  Writes the clusters before grouping for Paraview visualization.
- subroutine write_final_clusters_paraview (params)

  Writes the clusters after grouping for Paraview visualization.
- subroutine write_points_coord_format (unit_geo, unit_ind, nb_points, dim, coords, ids, k)

  Writes point coordinates from coordinates format for Paraview visualization.
- subroutine write_points_picture_format (unit_geo, unit_ind, nb_pixels, params, ids, proc_ids)

  Writes point coordinates from picture format for Paraview visualization.
- subroutine list_commands_paraview

  Lists the commands related to Paraview visualization.

5.13.1 Detailed Description

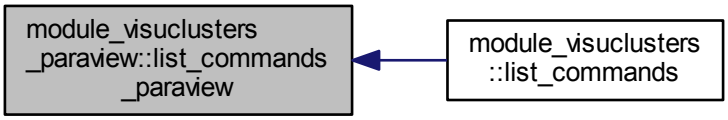
Contains methods enabling writing results in file specifically formatted for Paraview.

5.13.2 Function/Subroutine Documentation

5.13.2.1 subroutine module_visuclusters_paraview::list_commands_paraview ( )

Lists the commands related to Paraview visualization.

Here is the caller graph for this function:



5.13.2.2 subroutine module_visuclusters_paraview::write_assignment_paraview (type(type_params) *params*)

Initializes the file of the partitioning for Paraview visualization.

This method extracts details on partitioning from the *decoupe.x* files and writes them in *visu/affectation.∗* files. Extra files are written in a case of interface partitioning. *visu/affectation-interface.∗* files are the assignment for the interface.

Note

x is the identifier (number) of a process.

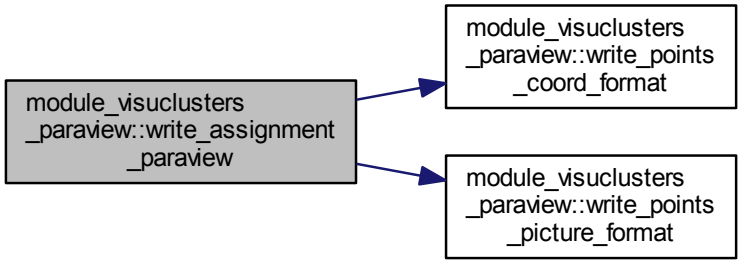
See also

module_calcul::write_partitioning(), partition_with_interface()

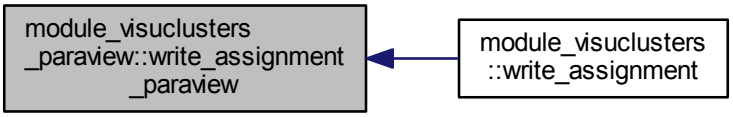
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.3 subroutine module_visuclusters_paraview::write_final_clusters_paraview (type(type_params) *params*)

Writes the clusters after grouping for Paraview visualization.

This methods extracts details on computed clusters from *cluster.final.x* files and writes them in *cluster.final.visu* file.

Note

x is the identifier (number) of a cluster.

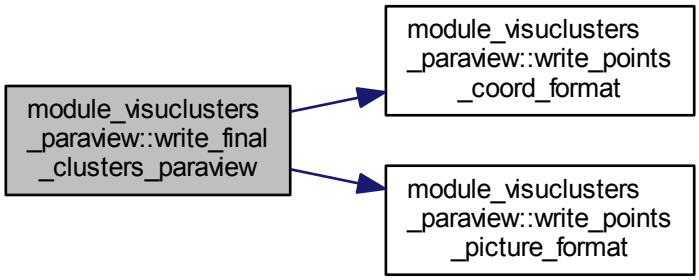
See also

module_calcul::write_final_clusters()

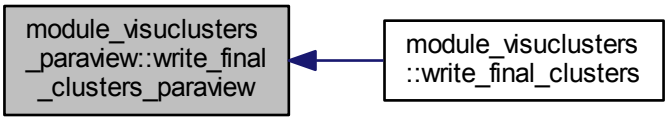
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.4 subroutine module_visuclusters_paraview::write_partial_clusters_paraview (type(type_params) *params*)

Writes the clusters before grouping for Paraview visualization.

This methods extracts details on computed clusters on each domain from *cluster.partiel.x* files and writes them in corresponding *visu/cluster.partiel.x.∗* files.

Note

x is the identifier (number) of a process.

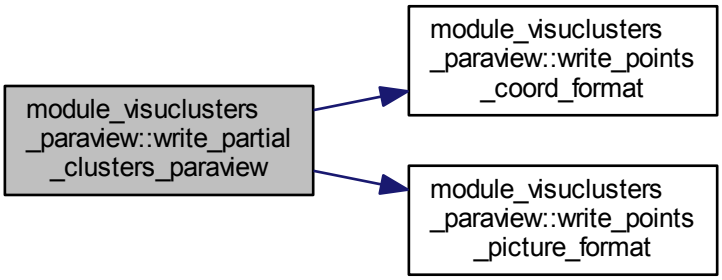
See also

module_calcul::write_partial_clusters()

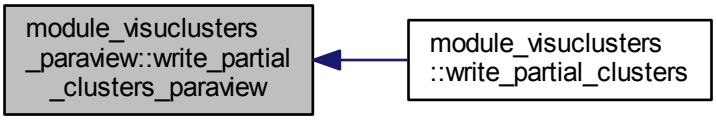
Parameters

in	*params*	the parameters defined in the *param.in* file

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.5 subroutine module_visuclusters_paraview::write_partitioning_paraview (type(type_params) *params*)

Writes the geometry partitioning for Paraview visualization.

This methods extracts the domain definitions from the *fort.2* file and writes to *visu/decoupe.∗* files.

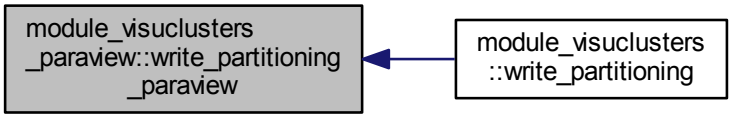
See also

module_calcul::write_partitioning()

Parameters

in	*params*	the parameters defined in the *param.in* file

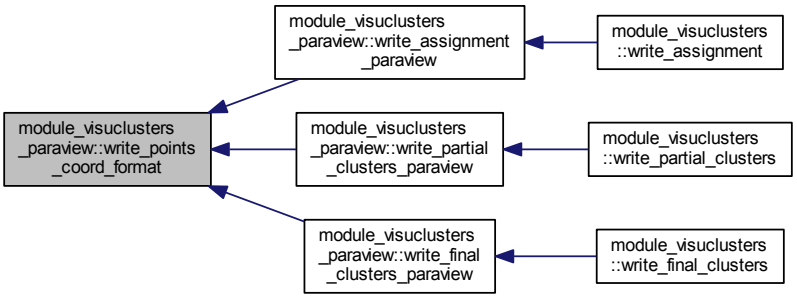
Here is the caller graph for this function:



5.13.2.6 subroutine module_visuclusters_paraview::write_points_coord_format (integer *unit_geo,* integer *unit_ind,* integer *nb_points,* integer *dim,* double precision, dimension(:,:), pointer *coords,* integer, dimension(:), pointer *ids,* integer *k*)

Writes point coordinates from coordinates format for Paraview visualization.

Here is the caller graph for this function:



5.13.2.7 subroutine module_visuclusters_paraview::write_points_picture_format (integer *unit_geo,* integer *unit_ind,* integer *nb_pixels,* type(type_params) *params,* integer, dimension(:), pointer *ids,* integer, dimension(:), pointer *proc_ids*)

Writes point coordinates from picture format for Paraview visualization.

The following is written in the geometric file for each point :

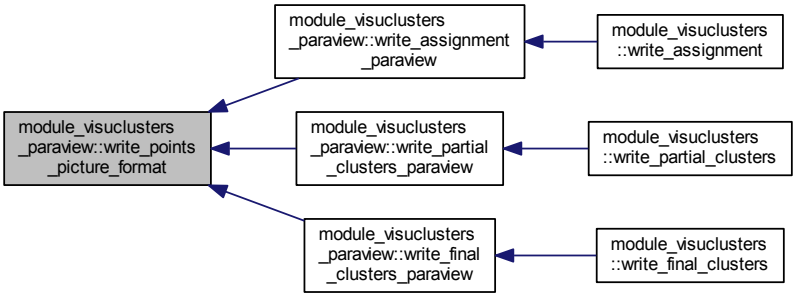
- **For 2D case**: *x y*
- **For 3D case**: *x y z*

And the following is written in the process partitioning file for each point : *ind*

Note

x,y and z are the coordinates and ind is the identifier of the process.
This method writes headers at first in the files.

Here is the caller graph for this function:



5.14 module_visuclusters_structure Module Reference

Contains useful data structures.

Data Types

- type type_params

5.14.1 Detailed Description

Contains useful data structures.

Chapter 6

Data Type Documentation

6.1 module_structure::type_clustering_param Type Reference

Public Attributes

- integer clustering_method_id
- integer kernelfunindex
- double precision sigma
- double precision gam
- double precision delta
- integer bandwidth

6.1.1 Member Data Documentation

6.1.1.1 integer module_structure::type_clustering_param::bandwidth

6.1.1.2 integer module_structure::type_clustering_param::clustering_method_id

6.1.1.3 double precision module_structure::type_clustering_param::delta

6.1.1.4 double precision module_structure::type_clustering_param::gam

6.1.1.5 integer module_structure::type_clustering_param::kernelfunindex

6.1.1.6 double precision module_structure::type_clustering_param::sigma

6.2 module_structure::type_clusters Type Reference

Public Attributes

- integer, dimension(:), pointer nbelt
- integer nb

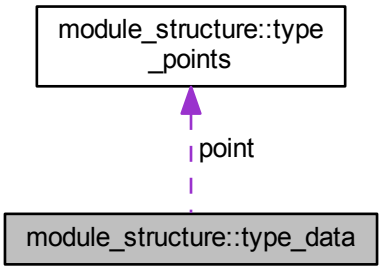
6.2.1 Member Data Documentation

6.2.1.1 integer module_structure::type_clusters::nb

6.2.1.2 integer, dimension(:), pointer module_structure::type_clusters::nbelt

6.3 module_structure::type_data Type Reference

Collaboration diagram for module_structure::type_data:



Public Attributes

- type(type_points), dimension(:), pointer point
- integer nb
- integer dim
- integer nbclusters
- integer coord
- integer image
- integer geom
- integer seuil
- integer interface
- integer recouvrement
- double precision, dimension(:), pointer pas
- integer, dimension(:,:), pointer refimg
- integer, dimension(:), pointer imgmap
- integer imgdim
- integer imgt

6.3.1 Member Data Documentation

6.3.1.1 integer module_structure::type_data::coord

6.3.1.2 integer module_structure::type_data::dim

6.3.1.3 integer module_structure::type_data::geom

6.3.1.4 integer module_structure::type_data::image

6.3.1.5 integer module_structure::type_data::imgdim

6.3.1.6 integer, dimension(:), pointer module_structure::type_data::imgmap

6.3.1.7 integer module_structure::type_data::imgt

6.3.1.8 integer module_structure::type_data::interface

6.3.1.9 integer module_structure::type_data::nb

6.3.1.10 integer module_structure::type_data::nbclusters

6.3.1.11 double precision, dimension(:), pointer module_structure::type_data::pas

6.3.1.12 type(type_points), dimension(:), pointer module_structure::type_data::point

6.3.1.13 integer module_structure::type_data::recouvrement

6.3.1.14 integer, dimension(:,:), pointer module_structure::type_data::refimg

6.3.1.15 integer module_structure::type_data::seuil

6.4 module_visuclusters_structure::type_params Type Reference

Public Attributes

- character(len=30) mesh
- double precision, dimension(:), pointer pas
- integer, dimension(:,:), pointer refimg
- integer, dimension(:), pointer imgmap
- integer coord
- integer dim
- integer geom
- integer image
- integer imgdim
- integer imgt
- integer interface
- integer nbclusters
- integer nbp
- integer nbproc
- integer recouvrement
- integer seuil

6.4.1 Member Data Documentation

6.4.1.1 integer module_visuclusters_structure::type_params::coord

6.4.1.2 integer module_visuclusters_structure::type_params::dim

6.4.1.3 integer module_visuclusters_structure::type_params::geom

6.4.1.4 integer module_visuclusters_structure::type_params::image

6.4.1.5 integer module_visuclusters_structure::type_params::imgdim

6.4.1.6 integer, dimension(:), pointer module_visuclusters_structure::type_params::imgmap

6.4.1.7 integer module_visuclusters_structure::type_params::imgt

6.4.1.8 integer module_visuclusters_structure::type_params::interface

6.4.1.9 character (len=30) module_visuclusters_structure::type_params::mesh

6.4.1.10 integer module_visuclusters_structure::type_params::nbclusters

6.4.1.11 integer module_visuclusters_structure::type_params::nbp

6.4.1.12 integer module_visuclusters_structure::type_params::nbproc

6.4.1.13 double precision, dimension(:), pointer module_visuclusters_structure::type_params::pas

6.4.1.14 integer module_visuclusters_structure::type_params::recouvrement

6.4.1.15 integer, dimension(:,:), pointer module_visuclusters_structure::type_params::refimg

6.4.1.16 integer module_visuclusters_structure::type_params::seuil

6.5 module_structure::type_points Type Reference

Public Attributes

- double precision, dimension(:), pointer coord
- integer cluster

6.5.1 Member Data Documentation

6.5.1.1 integer module_structure::type_points::cluster

6.5.1.2 double precision, dimension(:), pointer module_structure::type_points::coord

6.6 module_teste_clusters::type_test Type Reference

Public Attributes

- character(len=80) datatype
- character(len=80) decoupetype
- character(len=80) dir
- character(len=80) fichier
- character(len=80) output
- character(len=80) visup
- character(len=80) visug
- double precision epaisseur
- integer, dimension(:), pointer decoupe
- integer nbproc

6.6.1 Member Data Documentation

6.6.1.1 character (len=80) module_teste_clusters::type_test::datatype

6.6.1.2 integer, dimension(:), pointer module_teste_clusters::type_test::decoupe

6.6.1.3 character (len=80) module_teste_clusters::type_test::decoupetype

6.6.1.4 character (len=80) module_teste_clusters::type_test::dir

6.6.1.5 double precision module_teste_clusters::type_test::epaisseur

6.6.1.6 character (len=80) module_teste_clusters::type_test::fichier

6.6.1.7 integer module_teste_clusters::type_test::nbproc

6.6.1.8 character (len=80) module_teste_clusters::type_test::output

6.6.1.9 character (len=80) module_teste_clusters::type_test::visug

6.6.1.10 character (len=80) module_teste_clusters::type_test::visup

Chapter 7

File Documentation

7.1 module_calcul.f90 File Reference

Modules

- module module_calcul

  Contains the spectral clustering method and methods that computes affinity parameters for kernels and overlapping.

Functions/Subroutines

- subroutine module_calcul::get_sigma (partitioned_data, sigma)

  Computes the affinity parameter $sigma$.
- subroutine module_calcul::get_sigma_interface (proc_id, partitioned_data, sigma, bounds, partitioning, epsilon)

  Computes the affinity parameter $sigma$ for the interface.
- double precision function, dimension(partitioned_data%nb, partitioned_data%nb) module_calcul::poly_kernel (partitioned_data, gam, delta)
- double precision function, dimension(partitioned_data%nb, partitioned_data%nb) module_calcul::gaussian↩_kernel (partitioned_data, sigma)
- subroutine module_calcul::apply_kernel_k_means (proc_id, nb_clusters_max, nb_clusters_opt, partitioned↩_data, clust_param)
- subroutine module_calcul::apply_spectral_clustering (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, sigma, clust_param)
- subroutine module_calcul::mean_shift (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, bandwidth)

7.2 module_decoupe.f90 File Reference

Modules

- module module_decoupe

  Contains methods enabling the partitionning and the groupping of the data.

Functions/Subroutines

- subroutine module_decoupe::partition_data (data, epsilon, nb_proc, coord_min, coord_max, partitioning, points_by_domain, assignements, bounds)

Partitions the data into subdomains for a latter processing by the slaves.

- subroutine module_decoupe::define_bounds (data, coord_min, coord_max, bounds, partitioning, epsilon, nb_proc)

  Defines the bounds of each subdomain.
- subroutine module_decoupe::define_domains (nb_proc, data, domains, bounds, partitioning)

  Defines the different subdomains.
- subroutine module_decoupe::partition_with_interface (nb_proc, data, points_by_domain, assignements, domains, epsilon)

  Partitions the data using interface.
- subroutine module_decoupe::partition_with_overlapping (nb_proc, data, points_by_domain, assignements, domains)

  Partitions the data using overlapping.
- subroutine module_decoupe::group_clusters (nb_clusters, points_by_cluster, cluster_map, data)

  Groups the clusters and removes duplicates from the set of found clusters.

7.3 module_embed.f90 File Reference

Modules

- module module_embed

  Contains K-means and spectral embedding algorithms.

Functions/Subroutines

- subroutine module_embed::apply_spectral_embedding (nb_clusters, n, Z, A, ratio, clusters, clusters_centers, points_by_clusters, clusters_energies, nb_info, proc_id, ratio_moy, ratio_rij, ratio_rii)

  Computes the clusters using eigen vector matrix.
- subroutine module_embed::apply_kmeans (dim, nb_points, nb_clusters, nb_iter_max, nb_iter, points, clusters, clusters_centers, points_by_clusters, clusters_energies, proc_id)

  Implements K-Means algorithm (required by spectral clustering and Kernel K-Means methods) The algorithm works as follows:

7.4 module_entree.f90 File Reference

Modules

- module module_entree

  Contains methods enabling data and parameter file reading.

Functions/Subroutines

- subroutine module_entree::help

  Displays help on used formats and keywords for param.in file.
- subroutine module_entree::read_params (data, epsilon, coord_min, coord_max, nb_proc, partitioning, input_file, sigma, nb_clusters_max, list_nb_clusters, clust_param)

  Reads a file in which there is the whole information on the data.
- subroutine module_entree::read_coordinates_data (input_file, data, coord_min, coord_max)

  Reads data written in coordinates format.
- subroutine module_entree::read_picture_data (input_file, data, coord_min, coord_max)

Reads data written in picture format.

- subroutine module_entree::read_geometric_data (input_file, data, coord_min, coord_max)

  Reads data written in geometric format.

- subroutine module_entree::read_threshold_data (input_file, data, coord_min, coord_max)

  Reads data written in threshold format.

- subroutine module_entree::assign_picture_array (data)

  Puts the index number of the pixels into an array.

7.5 module_MPI.f90 File Reference

Modules

- module module_mpi

  Contains methods related to parallelism.

Functions/Subroutines

- subroutine module_mpi::send_partitioning (nb_proc, data, points_by_domain, ddat, partitioned_data)

  Sends the partitionning.

- subroutine module_mpi::receive_partitioning (proc_id, partitioned_data)

  Receives the partitionning.

- subroutine module_mpi::receive_number_clusters (nb_proc, nb_clusters, points_by_domain, partitioned_↩ data, array_clust)

  Receives the number of clusters.

- subroutine module_mpi::send_number_clusters (proc_id, partitioned_data)

  Sends the number of clusters.

- subroutine module_mpi::send_clusters (proc_id, partitioned_data)

  Sends the clusters.

- subroutine module_mpi::receive_clusters (nb_proc, nb_clusters, points_by_domain, ddat, partitioned_data, cluster_map, array_clust, points_by_cluster)

  Receives the clusters.

7.6 module_solve.f90 File Reference

Modules

- module module_solve

  Contains methods from Lapack library dealing with eigen values computing.

Functions/Subroutines

- subroutine module_solve::solve_dgeevx (N, A, VR, WR)
- subroutine module_solve::solve_dgeev (N, A, VR, WR)
- subroutine module_solve::solve_dsyev (N, A, W, LWORK)
- subroutine module_solve::solve_dsyevr (k, N, A, Z, LWORK, LIWORK, W, M)
- subroutine module_solve::solve_dsyevx (k, N, A, Z, LWORK, LIWORK, W, M)

7.7 module_sortie.f90 File Reference

Modules

- module module_sortie

  Contains methods enabling writing results in specific formatted files.

Functions/Subroutines

- subroutine module_sortie::write_domains (data, nb_proc, domains)

  Writes the file containing the domain definitions.
- subroutine module_sortie::write_partitioning (nb_proc, data, points_by_domain, assignements)

  Writes the files containing the partitionning.
- subroutine module_sortie::write_partial_clusters (proc_id, partitioned_data)

  Writes the file containing the clusters computed by the process.
- subroutine module_sortie::write_final_clusters (nb_clusters, points_by_cluster, cluster_map)

  Writes the files containing the final clusters after grouping.
- subroutine module_sortie::write_metadata (mesh, data, nb_proc, nb_clusters)

  Writes a file containing various information.

7.8 module_sparse.f90 File Reference

Modules

- module module_sparse

Functions/Subroutines

- subroutine module_sparse::apply_spectral_clustering_sparse (proc_id, nb_clusters_max, nb_clusters_opt, partitioned_data, sigma)

  Computes the clusters using spectral clustering algorithm using sparsity.
- subroutine module_sparse::apply_spectral_embedding_sparse (nb_clusters, n, Z, nnz, AS, IAS, JAS, ratio, clusters, clusters_centers, points_by_clusters, clusters_energies, nb_info, proc_id, ratio_moy, ratio_rij, ratio↩ _rii)

  Computes the ideal number of clusters using sparsity.
- subroutine module_sparse::compute_matvec_prod (A, IA, JA, X, Y, n, nnz)

  Computes the matrix vector product using sparsity.
- subroutine module_sparse::solve_arpack (A, IA, JA, dim, nnz, nb_clusters_max, W, Z)

7.9 module_structure.f90 File Reference

Data Types

- type module_structure::type_data
- type module_structure::type_points
- type module_structure::type_clusters
- type module_structure::type_clustering_param

Modules

- module module_structure

  Contains structure types required by the different modules.

7.10 module_teste_clusters.f90 File Reference

Data Types

- type module_teste_clusters::type_test

Modules

- module module_teste_clusters

Functions/Subroutines

- subroutine module_teste_clusters::create_executable (test)

  Creates an executable called go for runcluster.
- subroutine module_teste_clusters::create_test (test)

  Creates a test file.
- subroutine module_teste_clusters::execute_test (test)

  Executes the script go and displays information on the console screen.
- subroutine module_teste_clusters::create_data

  Generates an example of a raw geometric data file for testing purpose.

7.11 module_visuclusters.f90 File Reference

Modules

- module module_visuclusters

  Contains methods enabling writing results in a selected data file format (for now : Paraview or GMSH)

Functions/Subroutines

- subroutine module_visuclusters::read_metadata (params)

  Reads a file containing metadata on the data and the computed clusters.
- subroutine module_visuclusters::write_partitioning (format_output, params)

  Writes the geometry of the partitioning (Gmsh or Paraview) and calls the eponym function.
- subroutine module_visuclusters::write_assignment (format_output, params)

  Initializes the file of the partitionning.
- subroutine module_visuclusters::write_partial_clusters (format_output, params)

  Writes the clusters before grouping by calling the corresponding method (Gmsh or Paraview)
- subroutine module_visuclusters::write_final_clusters (format_output, params)

  Writes the clusters after grouping by calling the corresponding method (Gmsh or Paraview)
- subroutine module_visuclusters::list_commands (format_output)

  Lists the commands related to Gmsh or Paraview depending on the input parameter.

7.12 module_visuclusters_gmsh.f90 File Reference

Modules

- module module_visuclusters_gmsh

  Contains methods enabling writing results in file specifically formatted for GMSH.

Functions/Subroutines

- subroutine module_visuclusters_gmsh::write_partitioning_gmsh (params)

  Writes the geometry partitioning for Gmsh visualization.
- subroutine module_visuclusters_gmsh::write_assignment_gmsh (params)

  Initializes the file of the partitioning for Gmsh visualization.
- subroutine module_visuclusters_gmsh::write_partial_clusters_gmsh (params)

  Writes the clusters before grouping for Gmsh visualization.
- subroutine module_visuclusters_gmsh::write_final_clusters_gmsh (params)

  Writes the clusters after grouping for Gmsh visualization.
- subroutine module_visuclusters_gmsh::write_point_coord_format (unit, dim, coords, id, k)

  Writes point coordinates from coordinates format for Gmsh visualization.
- subroutine module_visuclusters_gmsh::write_point_picture_format (unit, params, id, k)

  Writes point coordinates from picture format for Gmsh visualization.
- subroutine module_visuclusters_gmsh::list_commands_gmsh

  Lists the commands related to Gmsh visualization.

7.13 module_visuclusters_paraview.f90 File Reference

Modules

- module module_visuclusters_paraview

  Contains methods enabling writing results in file specifically formatted for Paraview.

Functions/Subroutines

- subroutine module_visuclusters_paraview::write_partitioning_paraview (params)

  Writes the geometry partitioning for Paraview visualization.
- subroutine module_visuclusters_paraview::write_assignment_paraview (params)

  Initializes the file of the partitioning for Paraview visualization.
- subroutine module_visuclusters_paraview::write_partial_clusters_paraview (params)

  Writes the clusters before grouping for Paraview visualization.
- subroutine module_visuclusters_paraview::write_final_clusters_paraview (params)

  Writes the clusters after grouping for Paraview visualization.
- subroutine module_visuclusters_paraview::write_points_coord_format (unit_geo, unit_ind, nb_points, dim, coords, ids, k)

  Writes point coordinates from coordinates format for Paraview visualization.
- subroutine module_visuclusters_paraview::write_points_picture_format (unit_geo, unit_ind, nb_pixels, params, ids, proc_ids)

  Writes point coordinates from picture format for Paraview visualization.
- subroutine module_visuclusters_paraview::list_commands_paraview

  Lists the commands related to Paraview visualization.

7.14 module_visuclusters_structure.f90 File Reference

Data Types

- type module_visuclusters_structure::type_params

Modules

- module module_visuclusters_structure

  Contains useful data structures.

Index

apply_kernel_k_means
    module_calcul, 9
apply_kmeans
    module_embed, 18
apply_spectral_clustering
    module_calcul, 10
apply_spectral_clustering_sparse
    module_sparse, 33
apply_spectral_embedding
    module_embed, 19
apply_spectral_embedding_sparse
    module_sparse, 34
assign_picture_array
    module_entree, 21

bandwidth
    module_structure::type_clustering_param, 55

cluster
    module_structure::type_points, 58
clustering_method_id
    module_structure::type_clustering_param, 55
compute_matvec_prod
    module_sparse, 35
coord
    module_structure::type_data, 56
    module_structure::type_points, 58
    module_visuclusters_structure::type_params, 57
create_data
    module_teste_clusters, 37
create_executable
    module_teste_clusters, 37
create_test
    module_teste_clusters, 37

datatype
    module_teste_clusters::type_test, 58
decoupe
    module_teste_clusters::type_test, 58
decoupetype
    module_teste_clusters::type_test, 58
define_bounds
    module_decoupe, 13
define_domains
    module_decoupe, 13
delta
    module_structure::type_clustering_param, 55
dim
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
dir
    module_teste_clusters::type_test, 58

epaisseur
    module_teste_clusters::type_test, 59
execute_test
    module_teste_clusters, 37

fichier
    module_teste_clusters::type_test, 59

gam
    module_structure::type_clustering_param, 55
gaussian_kernel
    module_calcul, 10
geom
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
get_sigma
    module_calcul, 10
get_sigma_interface
    module_calcul, 11
group_clusters
    module_decoupe, 14

help
    module_entree, 21

image
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
imgdim
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
imgmap
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
imgt
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57
interface
    module_structure::type_data, 56
    module_visuclusters_structure::type_params, 57

kernelfunindex
    module_structure::type_clustering_param, 55

list_commands
    module_visuclusters, 38
list_commands_gmsh
    module_visuclusters_gmsh, 43

list_commands_paraview
    module_visuclusters_paraview, 47

mean_shift
    module_calcul, 12
mesh
    module_visuclusters_structure::type_params, 57
module_MPI.f90, 63
module_calcul, 9
    apply_kernel_k_means, 9
    apply_spectral_clustering, 10
    gaussian_kernel, 10
    get_sigma, 10
    get_sigma_interface, 11
    mean_shift, 12
    poly_kernel, 12
module_calcul.f90, 61
module_decoupe, 12
    define_bounds, 13
    define_domains, 13
    group_clusters, 14
    partition_data, 15
    partition_with_interface, 16
    partition_with_overlapping, 17
module_decoupe.f90, 61
module_embed, 17
    apply_kmeans, 18
    apply_spectral_embedding, 19
module_embed.f90, 62
module_entree, 20
    assign_picture_array, 21
    help, 21
    read_coordinates_data, 21
    read_geometric_data, 22
    read_params, 23
    read_picture_data, 23
    read_threshold_data, 24
module_entree.f90, 62
module_mpi, 25
    receive_clusters, 25
    receive_number_clusters, 26
    receive_partitioning, 26
    send_clusters, 27
    send_number_clusters, 27
    send_partitioning, 27
module_solve, 28
    solve_dgeev, 28
    solve_dgeevx, 29
    solve_dsyev, 29
    solve_dsyevr, 29
    solve_dsyevx, 29
module_solve.f90, 63
module_sortie, 30
    write_domains, 30
    write_final_clusters, 31
    write_metadata, 31
    write_partial_clusters, 32
    write_partitioning, 32
module_sortie.f90, 64
module_sparse, 33
    apply_spectral_clustering_sparse, 33
    apply_spectral_embedding_sparse, 34
    compute_matvec_prod, 35
    solve_arpack, 35
module_sparse.f90, 64
module_structure, 36
module_structure.f90, 64
module_structure::type_clustering_param, 55
    bandwidth, 55
    clustering_method_id, 55
    delta, 55
    gam, 55
    kernelfunindex, 55
    sigma, 55
module_structure::type_clusters, 55
    nb, 55
    nbelt, 55
module_structure::type_data, 56
    coord, 56
    dim, 56
    geom, 56
    image, 56
    imgdim, 56
    imgmap, 56
    imgt, 56
    interface, 56
    nb, 57
    nbclusters, 57
    pas, 57
    point, 57
    recouvrement, 57
    refimg, 57
    seuil, 57
module_structure::type_points, 58
    cluster, 58
    coord, 58
module_teste_clusters, 37
    create_data, 37
    create_executable, 37
    create_test, 37
    execute_test, 37
module_teste_clusters.f90, 65
module_teste_clusters::type_test, 58
    datatype, 58
    decoupe, 58
    decoupetype, 58
    dir, 58
    epaisseur, 59
    fichier, 59
    nbproc, 59
    output, 59
    visug, 59
    visup, 59
module_visuclusters, 38
    list_commands, 38
    read_metadata, 38
    write_assignment, 39

write_final_clusters, 40
write_partial_clusters, 41
write_partitioning, 41
module_visuclusters.f90, 65
module_visuclusters_gmsh, 42
list_commands_gmsh, 43
write_assignment_gmsh, 43
write_final_clusters_gmsh, 44
write_partial_clusters_gmsh, 45
write_partitioning_gmsh, 45
write_point_coord_format, 46
write_point_picture_format, 46
module_visuclusters_gmsh.f90, 66
module_visuclusters_paraview, 47
list_commands_paraview, 47
write_assignment_paraview, 48
write_final_clusters_paraview, 49
write_partial_clusters_paraview, 50
write_partitioning_paraview, 51
write_points_coord_format, 51
write_points_picture_format, 52
module_visuclusters_paraview.f90, 66
module_visuclusters_structure, 52
module_visuclusters_structure.f90, 67
module_visuclusters_structure::type_params, 57
coord, 57
dim, 57
geom, 57
image, 57
imgdim, 57
imgmap, 57
imgt, 57
interface, 57
mesh, 57
nbclusters, 58
nbp, 58
nbproc, 58
pas, 58
recouvrement, 58
refimg, 58
seuil, 58

nb
module_structure::type_clusters, 55
module_structure::type_data, 57
nbclusters
module_structure::type_data, 57
module_visuclusters_structure::type_params, 58
nbelt
module_structure::type_clusters, 55
nbp
module_visuclusters_structure::type_params, 58
nbproc
module_teste_clusters::type_test, 59
module_visuclusters_structure::type_params, 58

output
module_teste_clusters::type_test, 59

partition_data
module_decoupe, 15
partition_with_interface
module_decoupe, 16
partition_with_overlapping
module_decoupe, 17
pas
module_structure::type_data, 57
module_visuclusters_structure::type_params, 58
point
module_structure::type_data, 57
poly_kernel
module_calcul, 12

read_coordinates_data
module_entree, 21
read_geometric_data
module_entree, 22
read_metadata
module_visuclusters, 38
read_params
module_entree, 23
read_picture_data
module_entree, 23
read_threshold_data
module_entree, 24
receive_clusters
module_mpi, 25
receive_number_clusters
module_mpi, 26
receive_partitioning
module_mpi, 26
recouvrement
module_structure::type_data, 57
module_visuclusters_structure::type_params, 58
refimg
module_structure::type_data, 57
module_visuclusters_structure::type_params, 58

send_clusters
module_mpi, 27
send_number_clusters
module_mpi, 27
send_partitioning
module_mpi, 27
seuil
module_structure::type_data, 57
module_visuclusters_structure::type_params, 58
sigma
module_structure::type_clustering_param, 55
solve_arpack
module_sparse, 35
solve_dgeev
module_solve, 28
solve_dgeevx
module_solve, 29
solve_dsyev
module_solve, 29
solve_dsyevr

module_solve, 29
solve_dsyevx
module_solve, 29

visug
module_teste_clusters::type_test, 59
visup
module_teste_clusters::type_test, 59

write_assignment
module_visuclusters, 39
write_assignment_gmsh
module_visuclusters_gmsh, 43
write_assignment_paraview
module_visuclusters_paraview, 48
write_domains
module_sortie, 30
write_final_clusters
module_sortie, 31
module_visuclusters, 40
write_final_clusters_gmsh
module_visuclusters_gmsh, 44
write_final_clusters_paraview
module_visuclusters_paraview, 49
write_metadata
module_sortie, 31
write_partial_clusters
module_sortie, 32
module_visuclusters, 41
write_partial_clusters_gmsh
module_visuclusters_gmsh, 45
write_partial_clusters_paraview
module_visuclusters_paraview, 50
write_partitioning
module_sortie, 32
module_visuclusters, 41
write_partitioning_gmsh
module_visuclusters_gmsh, 45
write_partitioning_paraview
module_visuclusters_paraview, 51
write_point_coord_format
module_visuclusters_gmsh, 46
write_point_picture_format
module_visuclusters_gmsh, 46
write_points_coord_format
module_visuclusters_paraview, 51
write_points_picture_format
module_visuclusters_paraview, 52