Help

```
/** @addtogroup CLUSTERING
* @{
*/

/**
 * @author Ricardo RincÃşn GarcÃŋa
 *
 * @file clustering.h
 * @brief Definitions and functions prototypes of the clustering library.
 *
 * @date 17.08.15
 */
#ifndef CLUSTERING_H
#define CLUSTERING_H

//#include "auxiliarFuncs.h"

#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_random.h"


/**
 * @brief Macro for "everything is correct".
 */
#define OK 0
/**
 * @brief Macro for the error type.
 */
#define ERR -1
```

```
/**
 * @brief Maximum number of iterations to run k-means without convergence.
 */
#define MAX_ITERATIONS 100


/**
 * @brief Structure to keep some algorithm information (for k-means).
 */
typedef struct {
    double* centroids; //!< [numCentroids x dim]: Found centers.
    int* labels;       //!< [1 x numCentroids]: Implicitly added labels 1...numC
    int t;             //!< [1x1]: Number of iterations.
    double* MsErr;     //!< [1xt]: Mean-Square error at each iteration; not used
} Model_kmeans;


/**
 * @brief Free memory allocated for a Model structure.
 *
 * @param [in, out] mod Model structure to free.
 *
 * @related Model
 */
void freeModel_kmeans(Model_kmeans* mod);


/**
 * @brief Lloyd's algorithm to bundle the grid points using k-means clustering.
 *
 * @param [in] X A matrix (or vector if dim=1) to cluster by rows.
 * @param [in] n Number of points to cluster (number of rows of X).
 * @param [in] dim Points dimension (number of columns of X).
 * @param [in] k Number of clusters.
 * @param [out] finalAssignedClust Vector indicating which cluster correspond to
 *                                 each row of X. Vector dimension -> n.
 * @param [out] mod Model structure with run algorithm information.
 * @param [in] centroids Initial cluster's centroids (optional, can be NULL).
 * @param [in] maxIters Maximum number of iterations to run without convergence.
 * @param [in] rng Random generator.
 *
```

```
 * @retval OK Clustering done correctly.
 * @retval ERR An error occurred.
 *
 * @see initializeModel
 * @see assignNearestCluster
 * @see getNewClustersCentroids
 */
int kmeans(double* X, int n, int dim, int k, int* finalAssignedClust, Model_kmea
           double* centroids, int maxIters, PnlRng* rng);


/**
_kmeans * @brief Recursive Bifurcation method.
 *
 * The number of partitions/bundles, after p iterations, is equal to (2^d)^p.
 *
 * @param [in] S A matrix to cluster by rows.
 * @param [in] p Number of recursive iterations to do.
 * @param [out] S_Clusterized Vector indicating which cluster correspond to each
 *                            row of S. Vector dimension -> Nrows.
 *
 * @retval OK Clustering done correctly.
 * @retval ERR An error occurred.
 */
int recursiveBifurcation(const PnlMat* S, int p, int* S_Clusterized);


/**
 * @brief Recursive Bifurcation of Reduced State Space method.
 *
 * The number of partitions/bundles, after p iterations, is equal to 2^p.
 *
 * @param [in] S A matrix to cluster by rows.
 * @param [in] p Number of recursive iterations to do.
 * @param [out] S_Clusterized Vector indicating which cluster correspond to each
 *                            row of S. Vector dimension -> Nrows.
 * @param [in] mappingF_ptr Function pointer to the function to reduce the state
 *
 * @retval OK Clustering done correctly.
 * @retval ERR An error occurred.
 */
```

```
int recursiveBifurcationRSS(const PnlMat* S, int p, int* S_Clusterized,
                            void (*mappingF_ptr) (const PnlMat*, PnlVect*));


#endif

/** @}*/
```