ppam

1.1

Generated by Doxygen 1.9.1

Chapter 1

ppamlib: a library to implement the Partitioning Around Medoids (PAM) algorithm in parallel.

1.1 General explanation

This library uses the data in jmatrix format, a specific library of matrix manipulation that allows extremely big matrices (as long as the RAM of the machine allows).
Apart from the PAM itself the library also implements in parallel the calculation of the distance/dissimilarity matrix (metrics L1 and L2 and Pearson dissimilarity) and the silhouette of the resulting clustering.

It includes four example programs (see section Files below):

pardis: Parallel calculation of distance/dissimilarity matrix from a jmatrix with data
parpam: Parallel implementation of the Partitioning Around Medoids (PAM) algorithm from a distance matrix.
parsil: Parallel calculation of the silhouette of each points after the clustering has been applied.
tdvalue: Calculation of the value of the optimization function of the PAM algorithm for a given clusterization result.

These library uses the library jmatlib (see https://github.com/JdMDE/jmatlib) which therefore needs to be installed before compilation and use of ppamlib.

The code of this library with interface modifications is also used inside the parallelpam R package (https://CRAN.R-project.org/package=parallelpam) and inside the scellpam package (https://CRAN.R-project.org/package=scellpam)

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 DifftimeHelper Class Reference

`#include <diftimehelper.h>`

Public Member Functions

- DifftimeHelper ()
- void StartClock (std::string message)
- double EndClock (bool deb)

4.1.1 Detailed Description

@Difftimehelper class to help in measuring and printing time spent by parts of the programs

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DifftimeHelper()

```
DifftimeHelper::DifftimeHelper ( )
```

Default constructor

4.1.3 Member Function Documentation

4.1.3.1 EndClock()

```
double DifftimeHelper::EndClock (
            bool deb )
```

Function to end counting of time elapsed from the last call to StartClock. It prints the message with which StartClock was called if requested.

Parameters

in	*deb*	Boolean value to print or not the message stored by the last call to StartClock

4.1.3.2 StartClock()

```
void DifftimeHelper::StartClock (
            std::string message )
```

Function to start counting of time. It can be called nested inside other call made before; when it finishes each pair of StartClock/EndClock calls will show its message

Parameters

in	*message*	Message that will be printed in the console when the corresponding EndClock that matches this call be called.

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

- src/headers/diftimehelper.h
- src/library/diftimehelper.cpp

4.2 FastPAM< disttype > Class Template Reference

```
#include <fastpam.h>
```

Public Member Functions

- FastPAM (SymmetricMatrix< disttype > *Dm, indextype num_medoids, unsigned char inimet, int limiter, int nthreads)
- void Init (std::vector< indextype > initmedoids, unsigned int nt)
- void Run (unsigned char opt_method, unsigned int nt)
- FullMatrix< indextype > & GetMedoids ()
- FullMatrix< indextype > & GetMedoids (std::vector< std::string > rownames)
- FullMatrix< indextype > & GetAssign ()
- FullMatrix< indextype > & GetAssign (std::vector< std::string > rownames)
- std::vector< disttype > GetTDHistory ()
- std::vector< indextype > GetReassignHistory ()
- double GetInTime ()
- double GetOptTime ()
- unsigned int GetNumIter ()

4.2.1 Detailed Description

template<typename disttype>
class FastPAM< disttype >

A class to implement the Partitioning Around Medoids (PAM) clustering metho described in

Schubert, E. and Rousseeuw, P.J.: "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms."
Information Systems, vol. 101, p. 101804, 2021.
doi: https://doi.org/10.1016/j.is.2021.101804

Notice that the actual values of the vectors (instances) are not needed. To recover them, look at the data matrix used to generate the distance matrix.
The number of instances, N, is never passed since dissimilarity matrix is NxN and therefore its size indicates the N value.
With respect to the calculated value, it consists on two vectors. The first one has as many components as requested medoids and the second has as many components as instances.
Medoids are expressed in the first one by its number in the array of points (row in the dissimilarity matrix) starting at 0 (C++ convention).
The second vector contains the number of the medoid (i.e.: the cluster) to which each instance has been assigned, according to their order in the first vector (also from 0).
These vectors are returned by the functions GetMedoids and GetAssign (see their respective documentation)

4.2.2 Constructor & Destructor Documentation

4.2.2.1 FastPAM()

```
template<typename disttype >
template FastPAM< disttype >::FastPAM (
            SymmetricMatrix< disttype > * Dm,
            indextype num_medoids,
            unsigned char inimet,
            int limiter,
            int nthreads )
```

Default (and only available) constructor

Parameters

in	*Dm*	A pointer to a SymmetricMatrix which is the distance/dissimilarity matrix
in	*num_medois*	The number of medoids to be found
in	*initmet*	Initialization method (one of the constants INIT_METHOD_PREVIOUS, INIT_METHOD_BUILD or INIT_METHOD_LAB)
in	*limiter*	Maximum number of iterations allowed in the optimization phase. Use 0 to perform only initialization.
in	*nthreads*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter.

4.2.3 Member Function Documentation

4.2.3.1 GetAssign() [1/2]

```
template<typename disttype >
template FullMatrix< indextype > & FastPAM< disttype >::GetAssign ( )
```

This function gets the medoid to which each point is closest to as a FullMatrix of dimension (num_points x 1), i.e. a column vector

Returns

The column vector (as a FullMatrix) with the indices of the medoids in the vector of medoids (as returned by GetMedoids())
Obviously, and since this index is in [0..(num_medoids-1)], is also a class label.

4.2.3.2 GetAssign() [2/2]

```
template<typename disttype >
FullMatrix<indextype>& FastPAM< disttype >::GetAssign (
            std::vector< std::string > rownames )
```

This function gets the medoid to which each point is closest to as a FullMatrix of dimension (num_medoids x 1), i.e. a column vector with point names

Parameters

in	*rownames*	The names of all points as they are stored in the dissimilarity matrix, if it has names. These names are simply attached in the same order to the returned vector. These parameter can be obtained from the dissimilarity matrix with D.GetRowNames()

Returns

The column vector (as a FullMatrix) with the indices of the medoids in the vector of medoids (as returned by GetMedoids())
Obviously, and since this index is in [0..(num_medoids-1)], is also a class label.

4.2.3.3 GetInTime()

```
template<typename disttype >
double FastPAM< disttype >::GetInTime ( )  [inline]
```

This function returns the total time (in seconds) used for the initialization phase.

Returns

Time (in seconds) used for initalization

4.2.3.4 GetMedoids() [1/2]

```
template<typename disttype >
template FullMatrix< indextype > & FastPAM< disttype >::GetMedoids ( )
```

This function gets the medoids as a FullMatrix of dimension (num_medoids x 1), i.e. a column vector

Returns

The column vector (as a FullMatrix) with the indices of the medoids in the order they appear in the dissimilarity matrix

4.2.3.5 GetMedoids() [2/2]

```
template<typename disttype >
FullMatrix<indextype>& FastPAM< disttype >::GetMedoids (
            std::vector< std::string > rownames )
```

This function gets the medoids as a FullMatrix of dimension (num_medoids x 1), i.e. a column vector with point names

Parameters

in	*rownames*	The names of all points as they are stored in the dissimilarity matrix, if it has names. The function selects specifically those which are medoids and uses them as names for the returned vector. These parameter can be obtained from the dissimilarity matrix with D.GetRowNames()

Returns

The column vector (as a FullMatrix) with the indices of the medoids in the order they appear in the dissimilarity matrix

4.2.3.6 GetNumIter()

```
template<typename disttype >
unsigned int FastPAM< disttype >::GetNumIter ( )  [inline]
```

This function returns the number of iterations done in the optimization phase until convergence (or limiter of no convergence is reached)

Returns

Number of iterations used in optimization

4.2.3.7 GetOptTime()

```
template<typename disttype >
double FastPAM< disttype >::GetOptTime ( )  [inline]
```

This function returns the total time (in seconds) used for the optimization phase.

Returns

Time (in seconds) used for optimization

4.2.3.8 GetReassignHistory()

```
template<typename disttype >
std::vector<indextype> FastPAM< disttype >::GetReassignHistory ( )  [inline]
```

This function returns the number of points that have been swapped between tow clusters along the succesive optimization iterations.

Returns

The vector with the number of swapped points for each optimization step

4.2.3.9 GetTDHistory()

```
template<typename disttype >
std::vector<disttype> FastPAM< disttype >::GetTDHistory ( )  [inline]
```

This function returns the values of the optimization metrics TD (i.e.: the sum of distances of each point to its closest medoid, divided by the number of points) along the succesive optimization iterations.

Returns

The vector with the values of TD for each optimization step

4.2.3.10 Init()

```
template<typename disttype >
template void FastPAM< disttype >::Init (
            std::vector< indextype > initmedoids,
            unsigned int nt )
```

This function performs the initialization according to the method set at the class constructor

Parameters

in	*initmedoids*	A vector with the indices of the points that are considered as medoids after the initialization phase. This parameter makes sense (and it is used) ONLY for the initialization method PREV and is probably the result of a previous application of the algorithm, possibly with limiter=0. For other methods it is ignored; just pass an empty vector
in	*nt*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter.

4.2.3.11 Run()

```
template<typename disttype >
template void FastPAM< disttype >::Run (
            unsigned char opt_method,
            unsigned int nt )
```

This function runs the optimization phase according to the chosen optimization method

Parameters

in	*opt_method*	Optimization method (one of the constants OPT_METHOD_FASTPAM1 or OPT_METHOD_FASTPAMBSIL)
in	*nt*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter.

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

- src/headers/fastpam.h
- src/library/fastpam.cpp

Chapter 5

File Documentation

5.1 src/examples/pardis.cpp File Reference

```
#include <iostream>
#include <cstdlib>
#include "../headers/fastpam.h"
#include "../headers/debugpar_ppam.h"
#include "../headers/threadhelper.h"
#include "../headers/dissimmat.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

pardis

See program use in the documention to main() below

NOTE: The includes in this source file are for compilation of this program as an example together with the library,
before the library itself is installed. Once you have installed the library (assuming headers are in
/usr/local/include, lib is in /usr/local/lib or in other place included in your compiler search path)
you should substitute this by

#include <parallelpam/debugpar_ppam.h> etc...

and compile with something like

g++ -Wall pardis.cpp -o pardis -ljmatrix -lppam

5.1.2 Function Documentation

5.1.2.1 main()

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

pardis

A program to calculate the distance/dissimilarity matrix between the rows of an input matrix considering each row as a vector/individual and each column as a dimension/feature.

The program must be called as

pardis input_file [-dis distype] [-vtype valuetype] [-nt numthreads] [-com comment] -o out_file_name

where

input_file: File with the input matrix in jmatrix format.
It must be a matrix of float or double with dimension (n x p) where the individuals (points/vectors, which are n) must be the rows and components/dimensions (which are p) must be the columns.
Remember that you can use the program 'jmatrix csvread ...' to create this file from a .csv table
This argument is compulsory and must be immediately after the program name.

dis: Type of metrics/dissimilarity, which must be one of the strings 'L1' (Manhattan), 'L2' (Euclidean) or 'Pe' (Pearson dissimilarity). Default: L2.

vtype: Data type for the output dissimilarity/distance matrix.
It must be one of the strings 'float' or 'double'. Default: float.

numthreads: Requested number of threads.
Setting it to 0 will make the program to choose according to the number of processors/cores of your machine (default value).
Setting to -1 forces serial implementation (no threads)

comment: Comment to be attached to the dissimilarity matrix. Default: no comment will be added.

out_file_name: Name of the file contaning the dissimilarity matrix as a binary jmatrix.
If the input matrix has row names, these names will be copied to the dissimilarity matrix as row names, too.
This argument is compulsory and must be the last one.

Calling this program as **pardisd** turns on debugging; calling it as **pardisdd** turns on the jmatrix library debugging, too.
The distance/dissimilarity matrix in the output file will be a SymmetricMatrix of the requested data type and size (n x n).
The used memory is quadratic with n (concretely, $n*(n+1)/2$) so it can be very big.
The program refuses to create it if not enough RAM is available, and shows a warning
if the required amount of memory is above 75% of the available RAM.

5.2 src/examples/parpam.cpp File Reference

```
#include <iostream>
#include <cstdlib>
#include "../headers/debugpar_ppam.h"
#include "../headers/threadhelper.h"
#include "../headers/fastpam.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.2.1 Detailed Description

parpam

See program use in the documention to main() below

NOTE: The includes in this source file are for compilation of this program as an example together with the library,
before the library itself is installed. Once you have installed the library (assuming headers are in
/usr/local/include, lib is in /usr/local/lib or in other place included in your compiler search path)
you should substitute this by

#include <parallelpam/debugpar_ppam.h> etc...

and compile with something like

g++ -Wall parpam.cpp -o parpam -ljmatrix -lppam

5.2.2 Function Documentation

5.2.2.1 main()

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

parpam

A program to apply the Partitionin Around Medoids (PAM) clustering method to a set of individuals whose dissimilarity matrix is given, in parallel. It implements the FASTPAM1 algorithm described in

Schubert, E. and Rousseeuw, P.J.: "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms."
Information Systems, vol. 101, p. 101804, 2021.
doi: https://doi.org/10.1016/j.is.2021.101804

See documentation of class FastPAM for more information
The program must be called as

parpam ds_file k [-imet method (medoids_file)] [-omet method] [-mit max_iter] [-nt numthreads] -o root_file_name

where

ds_file: File with the dissimilarity matrix in jmatrix format.
It must be a symmetric matrix of float or double with dimension (n x n). You can use program pardis.cpp to generate

it.
This argument is compulsory and must be the first one after the program name.

k: Requested number of medoids (possitive integer number, k<n).
This argument is compulsory and must be the second one after the program name.

imet: Initialization method, which must be one of the strings 'BUILD', 'LAB' or 'PREV'
If you use PREV the file with the initial medoids must be given, too, which must be
a jmatrix FullMatrix of unsiged int with dimension (n x 1) (as returned by another call to this program)
If you use BUILD or LAB no initial medoids file should be provided. Default value: BUILD.

omet: Optimization method, which must be one of the strings 'FASTPAM1' or 'TWOBRANCH'. Default value: FASTPAM1

max_iter: Maximum number of iterations. Set it to 0 to do only the initialization phase (with BUILD or LAB method).
Default value: the value of constant MAX_ITER defined in fastpam.h

numthreads: Requested number of threads.
Setting it to 0 will make the program to choose according to the number of processors/cores of your machine (default value).
Setting to -1 forces serial implementation (no threads)

root_fname: A string used to build root_fname_med.bin and root_fname_clas.bin.
This argument is compulsory and must be the last one.

Calling this program as **parpamd** turns on debugging; calling it as **parpamdd** turns on the jmatrix library debugging, too.
The output files will contain jmatrix vectors of final medoids and classification, respectively.
Both are FullMatrix of indextype (unsigned int) with dimensions (k x 1) for med and (n x 1) for clas.
The first one contains the indices of the found medoids as row indices of the dissimilarity matrix, from 0 (i.e.: integers in range [0..n-1]).
The second contains the index in the first one (from 0) of the medoid to which class each point belongs to (i.e.: integers in range [0..k-1]).
If the dissimilarity matrix contained row names (i.e.: point names) the output vectors will keep them, too.
The program will refuse to load the dissimilarity matrix if not enough RAM is available;
also, it will show a warning if the required amount of memory to load it is above 75% of the available RAM.
Remember that using the program 'jmat csvdump ...' you can convert the output files to .csv format.

5.3 src/examples/parsil.cpp File Reference

```
#include "../headers/debugpar_ppam.h"
#include "../headers/threadhelper.h"
#include "../headers/fastpam.h"
#include "../headers/silhouette.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.3.1 Detailed Description

parsil

See program use in the documention to main() below

NOTE: The includes in this source file are for compilation of this program as an example together with the library,
before the library itself is installed. Once you have installed the library (assuming headers are in
/usr/local/include, lib is in /usr/local/lib or in other place included in your compiler search path)
you should substitute this by

#include <parallelpam/debugpar_ppam.h> etc...

and compile with something like

g++ -Wall parsil.cpp -o parsil -ljmatrix -lppam

5.3.2 Function Documentation

5.3.2.1 main()

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

parsil

A program to calculate the silhouette of a clustering (usually obtained with parpam) in parallel

The program must be called as

parsil dissim_file clasif_file [-nt numthreads] -o out_file_name

where

dissim_file: File with the dissimilarity matrix in jmatrix format.
It must be a SymmetricMatrix of float or double with dimension (n x n).

clasif_file: File with the clasification result, as obtained from program parpam.
It must be a (n x 1) matrix (a column vector) of unsigned int values with values in 0..(k-1) being k the number of clusters.

numthreads: Requested number of threads.
Setting it to 0 will make the program to choose according to the number of processors/cores of your machine (default value).
Setting to -1 forces serial implementation (no threads)

out_file_name: Name of the file contaning the silhouette. Compulsory.
The output file will be a FullMatrix of double type and dimension (n x 1) (a column vector) with the value of the

silhouette for each point.

Points are assumed to be in the same order in the dissimilarity matrix and the clasification vector, and this is the order in which their
silhouettes will be written in the output vector. If the matrix has row names, they will be set for the output file. It the clasif vector
has row names, they will be checked against the row names of the matrix, if both are present. If only clasification vector has names,
they will be set for the output vector.
The program will refuse to load the dissimilarity matrix if not enough RAM is available; also, it will show a warning if the required amount
of memory to load it is above 75% of the available RAM.
Remember that using the program 'jmat csvdump ...' you can convert the output file to .csv format.

5.4 src/examples/tdvalue.cpp File Reference

```
#include <jmatrixlib/fullmatrix.h>
#include "../headers/debugpar_ppam.h"
#include "../headers/gettd.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.4.1 Detailed Description

tdvalue

See program use in the documention to main() below

NOTE: The includes in this source file are for compilation of this program as an example together with the library,
before the library itself is installed. Once you have installed the library (assuming headers are in
/usr/local/include, lib is in /usr/local/lib or in other place included in your compiler search path)
you should substitute this by

#include <parallelpam/debugpar_ppam.h> etc...

and compile with something like

g++ -Wall tdvalue.cpp -o tdvalue -ljmatrix -lppam

5.4.2 Function Documentation

5.4.2.1 main()

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

tdvalue

A program to obtain the value of the TD optimization value of a clustering result.
TD is defined as the sum of distances of each point to its closest medoid divided by the total number of points.
This program takes as all its inputs binary files in jmatrix format.

The program must be called as

```
tdvalue med_file clas_file ds_file
```

where, if n is the number of points and k the number of medoids,

med_file: File with the indexes of the medoids in jmatrix format. Compulsory
It must be a (k x 1) full matrix (column vector) of indextype (unsigned int)

class_file: File with the number (from 0 to k-1) of the medoid each point is closest to. Compulsory
It must be a (n x 1) full matrix (column vector) of indextype (unsigned int)

ds_file: File with the dissimilarity matrix in jmatrix format. Compulsory
It must be a symmetric matrix of float or double with dimension (n x n).

The only output will be a double number written in the screen (unless you call the program as **tdvalued** or **tdvaluedd** for debugging).
Points are assumed to be in the same order in the dissimilarity matrix and the classification vector.
The program will refuse to load the dissimilarity matrix if not enough RAM is available.
Also, it will show a warning if the required amount of memory to load it is above 75% of the available RAM.

5.5 src/headers/debugpar_ppam.h File Reference

```
#include <iostream>
#include <string>
#include <jmatrixlib/debugpar.h>
```

Functions

- void ParallelpamSetDebug (bool deb, bool debjmat)
- void ParallelpamStop (std::string errortext)
- void ParallelpamWarning (std::string warntext)

Variables

- const unsigned char DEBPP =0x02

5.5.1 Function Documentation

5.5.1.1 ParallelpamSetDebug()

```
void ParallelpamSetDebug (
            bool deb,
            bool debjmat )
```

Sets the debug state to get messages or not in each library

Parameters

in	*deb*	true to get messages, false to suppress them. Default state is false.
in	*debjmar*	true to get debugging messages from the jmatrix library, false to supress them. Default state is false.

5.5.1.2 ParallelpamStop()

```
void ParallelpamStop (
            std::string errortext )
```

Sends an error message to the console and stops the program that is using the library

Parameters

in	*errortext*	The text of the message to be shown. It will appear after a standard message saying that is comes from this library.

5.5.1.3 ParallelpamWarning()

```
void ParallelpamWarning (
            std::string warntext )
```

Sends a warning message to the console and goes on with the program that is using the library

Parameters

in	*errortext*	The text of the message to be shown. It will appear after a standard message saying that is comes from this library.

5.5.2 Variable Documentation

5.5.2.1 DEBPP

```
const unsigned char DEBPP =0x02
```

This constant is to allow selective debug by library. Each library will print messages or not using a test with logical AND between its particular constant and the DEB global variable. This allows the use of the system either in each separate package or in the global one

5.6 src/headers/diftimehelper.h File Reference

```
#include <iostream>
#include <chrono>
#include <vector>
#include <string>
```

Classes

- class DifftimeHelper

5.7 src/headers/dissimmat.h File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include <unistd.h>
#include <cmath>
#include <jmatrixlib/fullmatrix.h>
#include <jmatrixlib/sparsematrix.h>
#include <jmatrixlib/symmetricmatrix.h>
#include <jmatrixlib/memhelper.h>
```

Functions

- template<typename counttype , typename disttype >
  SymmetricMatrix< disttype > & CalcDistFromFull (FullMatrix< counttype > &M, unsigned char dtype, unsigned int nthr)
- template<typename counttype , typename disttype >
  SymmetricMatrix< disttype > & CalcDistFromSparse (SparseMatrix< counttype > &M, unsigned char dtype, unsigned int nthr)

Variables

- const unsigned char **DL1** =0x0
- const unsigned char **DL2** =0x1
- const unsigned char **DPe** =0x2

5.7.1 Function Documentation

5.7.1.1 CalcDistFromFull()

```
template<typename counttype , typename disttype >
SymmetricMatrix<disttype>& CalcDistFromFull (
            FullMatrix< counttype > & M,
            unsigned char dtype,
            unsigned int nthr )
```

Function to calculate the distance matrix from the data matrix if such matrix is a FullMatrix (in the terminology of the JMatrix library)
counttype is the data type of the data matrix
disttype is the data type of the dissimilarity matrix to be returned (use float or double)

Parameters

in	*M*	The FullMatrix with the data where rows represent individuals (points) and columns are characteristics (dimensions)
in	*dtype*	Distance type. Use one of the constants DL1 for Manhattan/City block distance, DL2 for Euclidean distance and Dpe for Pearson dissimilarity coefficient
in	*nthr*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter.

5.7.1.2 CalcDistFromSparse()

```
template<typename counttype , typename disttype >
SymmetricMatrix<disttype>& CalcDistFromSparse (
            SparseMatrix< counttype > & M,
            unsigned char dtype,
            unsigned int nthr )
```

Function to calculate the distance matrix from the data matrix if such matrix is a SparseMatrix (in the terminology of the JMatrix library)
counttype is the data type of the data matrix
disttype is the data type of the dissimilarity matrix to be returned (use float or double)

Parameters

in	*M*	The SparseMatrix with the data where rows represent individuals (points) and columns are characteristics (dimensions)

Parameters

in	*dtype*	Distance type. Use one of the constants DL1 for Manhattan/City block distance, DL2 for Euclidean distance and Dpe for Pearson dissimilarity coefficient
in	*nthr*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter.

5.8 src/headers/fastpam.h File Reference

```
#include <utility>
#include <map>
#include <vector>
#include <typeinfo>
#include <jmatrixlib/fullmatrix.h>
#include <jmatrixlib/symmetricmatrix.h>
#include <jmatrixlib/memhelper.h>
```

Classes

- class FastPAM< disttype >

Macros

- #define MIND std::numeric_limits<disttype>::min()
- #define MAXD std::numeric_limits<disttype>::max()

Variables

- const std::string init_method_names [NUM_INIT_METHODS] ={"PREV","BUILD","LAB"}
- const std::string opt_method_names [NUM_OPT_METHODS] ={"FASTPAM1","TWOBRANCH"}
- const unsigned int MAX_ITER =1001
- const indextype MAX_MEDOIDS =std::numeric_limits<indextype>::max()-1
- const indextype NO_CLUSTER =MAX_MEDOIDS

- const unsigned char INIT_METHOD_PREVIOUS =0
- const unsigned char INIT_METHOD_BUILD =1
- const unsigned char INIT_METHOD_LAB =2
- const unsigned char NUM_INIT_METHODS =3

- const unsigned char OPT_METHOD_FASTPAM1 =0
- const unsigned char OPT_METHOD_FASTPAMBSIL =1
- const unsigned char NUM_OPT_METHODS =2

5.8.1 Macro Definition Documentation

5.8.1.1 MAXD

```
#define MAXD std::numeric_limits<disttype>::max()
```

The possible minimum and maximum values the dissimilarity can take. To initialize before loops.

5.8.1.2 MIND

```
#define MIND std::numeric_limits<disttype>::min()
```

The possible minimum and maximum values the dissimilarity can take. To initialize before loops.

5.8.2 Variable Documentation

5.8.2.1 INIT_METHOD_BUILD

```
const unsigned char INIT_METHOD_BUILD =1
```

The values of this constants is arbitrary, just a mark to distinguish it as a different initalization method
If you add other initialization method, do it at the end and increase the NUM_INIT_METHODS constant.

5.8.2.2 INIT_METHOD_LAB

```
const unsigned char INIT_METHOD_LAB =2
```

The values of this constants is arbitrary, just a mark to distinguish it as a different initalization method
If you add other initialization method, do it at the end and increase the NUM_INIT_METHODS constant.

5.8.2.3 init_method_names

```
const std::string init_method_names[NUM_INIT_METHODS] ={"PREV","BUILD","LAB"}
```

Names of the initialization methods. Their positions in the array must coincide with its constant.

5.8.2.4 INIT_METHOD_PREVIOUS

```
const unsigned char INIT_METHOD_PREVIOUS =0
```

The values of this constants is arbitrary, just a mark to distinguish it as a different initalization method
If you add other initialization method, do it at the end and increase the NUM_INIT_METHODS constant.

5.8.2.5 MAX_ITER

```
const unsigned int MAX_ITER =1001
```

The maximum number of iterations we will allow

5.8.2.6 MAX_MEDOIDS

```
const indextype MAX_MEDOIDS =std::numeric_limits<indextype>::max()-1
```

The maximum number of medoids we allow.

5.8.2.7 NO_CLUSTER

```
const indextype NO_CLUSTER =MAX_MEDOIDS
```

A convenience constant to indicate a point is not currently assigned to any medoid

5.8.2.8 NUM_INIT_METHODS

```
const unsigned char NUM_INIT_METHODS =3
```

The values of this constants is arbitrary, just a mark to distinguish it as a different initalization method
If you add other initialization method, do it at the end and increase the NUM_INIT_METHODS constant.

5.8.2.9 NUM_OPT_METHODS

```
const unsigned char NUM_OPT_METHODS =2
```

Arbitrary constant, just a mark to distinguish the different algorithms for the optimizacion phase

5.8.2.10 OPT_METHOD_FASTPAM1

```
const unsigned char OPT_METHOD_FASTPAM1 =0
```

Arbitrary constant, just a mark to distinguish the different algorithms for the optimizacion phase

5.8.2.11 OPT_METHOD_FASTPAMBSIL

```
const unsigned char OPT_METHOD_FASTPAMBSIL =1
```

Arbitrary constant, just a mark to distinguish the different algorithms for the optimizacion phase

5.8.2.12 opt_method_names

```
const std::string opt_method_names[NUM_OPT_METHODS] ={"FASTPAM1","TWOBRANCH"}
```

Names of the optimization methods. Their positions in the array must coincide with its constant.

5.9 src/headers/gettd.h File Reference

```
#include <jmatrixlib/symmetricmatrix.h>
```

Functions

- template<typename disttype >
  double GetTD (std::vector< indextype > Lmed, std::vector< indextype > Lclasif, SymmetricMatrix< disttype > &D)

5.9.1 Function Documentation

5.9.1.1 GetTD()

```
template<typename disttype >
double GetTD (
            std::vector< indextype > Lmed,
            std::vector< indextype > Lclasif,
            SymmetricMatrix< disttype > & D )
```

Function to get the value of the metrics usually employed in PAM minimization: sum of distances of each point to its closest medoid divided by number of points.

Parameters

in	*Lmed*	A vector with the indices of the points which are medoids. These indices refer to the order of points in the distance/dissimilarity matrix
in	*Lclasif*	A vector with the index (as position in Lmed) of the medoid closest to each point
in	*D*	A reference to the dissimilariry matrix, as a SymmetricMatrix

Returns

The value of the total sum of distances divided by the number of points

5.10 src/headers/intropage.h File Reference

5.11 src/headers/silhouette.h File Reference

```
#include <thread>
#include <jmatrixlib/fullmatrix.h>
#include <jmatrixlib/sparsematrix.h>
#include <jmatrixlib/symmetricmatrix.h>
#include <jmatrixlib/memhelper.h>
```

Typedefs

- typedef double siltype

Functions

- template<typename disttype >
  std::vector< siltype > CalculateSilhouette (std::vector< indextype > cl, SymmetricMatrix< disttype > &D, unsigned int nt)
- template<typename disttype >
  siltype CalculateMeanSilhouette (std::vector< indextype > cl, indextype nmed, SymmetricMatrix< disttype > *D, unsigned int nt)

5.11.1 Typedef Documentation

5.11.1.1 siltype

```
typedef double siltype
```

The values of the silhouette will be stored as double. Since its number is always linear with the number of points, this should not increase too much memory usage and is simpler.

5.11.2 Function Documentation

5.11.2.1 CalculateMeanSilhouette()

```
template<typename disttype >
siltype CalculateMeanSilhouette (
            std::vector< indextype > cl,
            indextype nmed,
            SymmetricMatrix< disttype > * D,
            unsigned int nt )
```

Function to calculate in parallel the mean values of the silhouette of all points after a clustering has been done
disttype is the value type used to represent distances in the dissimilarity matrix, either float or double
siltype is the value type used to store the silhouette, here defined as double

Parameters

in	*cl*	A vector with the class each point belong to, as a number in [0..(num_classes-1)]. Its length must be the number of points, which is the number of rows (and of columns) of the dissimilarity matrix
in	*D*	A reference to the dissimilariry matrix, as a SymmetricMatrix
in	*nt*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter

Returns

The mean value of the silhouette of all points.

5.11.2.2 CalculateSilhouette()

```
template<typename disttype >
std::vector<siltype> CalculateSilhouette (
            std::vector< indextype > cl,
            SymmetricMatrix< disttype > & D,
            unsigned int nt )
```

Function to calculate in parallel the silhouette of each point after a clustering has been done
disttype is the value type used to represent distances in the dissimilarity matrix, either float or double
siltype is the value type used to store the silhouette, here defined as double

Parameters

in	*cl*	A vector with the class each point belong to, as a number in [0..(num_classes-1)]. Its length must be the number of points, which is the number of rows (and of columns) of the dissimilarity matrix
in	*D*	A reference to the dissimilariry matrix, as a SymmetricMatrix
in	*nt*	Number of threads to be opened. Normally, use the result of function ChooseNumThreads(AS_MANY_AS_POSSIBLE) to get this parameter

Returns

A vector with as many components as points containing the silhouette value of each one. Order of points is as in the dissimilarity matrix.

5.12 src/headers/threadhelper.h File Reference

```
#include <sstream>
#include <cstdlib>
#include <thread>
```

Functions

- unsigned int ChooseNumThreads (int nthreads)

Variables

- const int **NO_THREADS** =-1
- const int **AS_MANY_AS_POSSIBLE** =0

5.12.1 Function Documentation

5.12.1.1 ChooseNumThreads()

```
unsigned int ChooseNumThreads (
            int nthreads )
```

Auxiliary function to decide the number of threads

Parameters

nthreads	Number of thread the user wish to launch. It can be: -1 to indicate not to use threads (strictly sequential computation) 0 to let the program choose according to the available number of cores/threads in the machine Other possitive number to ask for such number of threads

Returns

The number of threads that will be really used. This is:
1 for input value -1
the number of cores/threads in the machine for input value 0 (if hypertheading exists, it is taken into account)
the chosen number for any other input value (but a warning message is raised if the chosen number is bigger than the number of cores/threads)

Index