# MIDAPACK - MIcrowave Data Analysis PACKage

1.0beta

Generated by Doxygen 1.7.4

Mon May 7 2012 11:11:44

# Contents

# Chapter 1

# MIDAPACK library

## 1.1 General description

The goal of the **MIDAS** project is to provide high performance, middle-layer software tools, which would aid CMB data analysis efforts, for current and planned CMB experiments, to capitalize on the computational power of parallel (super)computers. The functionality provided by the library is supposed to fill in the gap in between available, low-level, high performance software packages such as Fast Fourier Transforms, dense and sparse linear algebra operations, etc, and the high-level data analysis pipelines, and thus to help the users to benefit from the former, while developing the latter in a more straightforward and transparent way. At the end of the project the library is supposed to provide functionality relevant to all main stages of the data analysis.

For more information about ANR MIDAS'09 project, and to find out how to contact us, see:

http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html

The first installement of the library is a **Toeplitz algebra** package described below.

This documentation covers the following topics:

- Toeplitz algebra documentation
- MIDAPACK development team

### 1.1.1 ACKNOWLEDGMENT:

This work has been supported in part by French National Research Agency (ANR) through its COSINUS program (project MIDAS no. ANR-09-COSI-009). High performance computing resources have been provided:

- in France by CCRT, TGCC, and IDRIS supercomputing centers under the GENCI program through projects: 2011-066647 and 2012-066647;

- in the US by the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

# Chapter 2

# MIDAPACK development team

The MIDAPACK development team:

- Pierre Cargemel (developer);
- Frédéric Dauvergne (developer);
- Giulio Fabbian (validator);
- Laura Grigori (coordinator);
- Maude Le Jeune (senior developer);
- Antoine Rogier (developer - till Aug 31, 2011);
- Mikolaj Szydlarski (developer);
- Radek Stompor (coordinator).

# Chapter 3

# Toeplitz algebra documentation



## 3.1 Introduction

Toeplitz matrices are ubiquitous in the CMB data analysis as they describe correlation properties of stationary time-domain processes (usually instrumental noise). The matrices relevant are therefore **symmetric** and **non-negative** definite. They are also **band-diagonal** as the noise correlation length, i.e., the band-width in the parlance of Toeplitz algebra, is typically much shorter than length of the data. A useful and important generalization of those include :

- **symmetric**, **block-diagonal** Toeplitz matrices - describing piece-wise stationary processes, each of the blocks is in turn a symmetric, band-diagonal matrix, which can be different for the different blocks.

## 3.2 Functionality

The Toeplitz algebra package described here provides functionality for calculating products of a Toeplitz matrix (understood as one of those described above) and a general matrix. The latter is referred to hereafter typically as a data matrix.

The list of specific functions provided is as follows:

- symmetric band Toeplitz matrix-matrix product;
- symmetric block-diagonal Toeplitz matrix-matrix product;
- symmetric block-diagonal Toeplitz matrix-matrix product with missing samples (gaps).

## 3.3 Programming models

The Toeplitz algebra library routines allow the user to take advantage of both **multi-threaded** and **memory-distributed** programming paradigms and are therefore adapted to run efficiently on heteregeneous computer architectures. The multithreading is implemented using **openMP** directives, while distributed programming uses **MPI**. Both shared and/or distributed parallelism can be switched of, at the compilation time, if so desired. Moreover, the user has always access to two versions of each of the routines: openMP/MPI and openMP-only.

We note that the MPI version of the routines are essentially just wrappers on openMP/sequential versions of the corresponding routines, which facilitate necessary data exchanges between distributed MPI processes.

## 3.4 Data distribution and load balancing

In the memory-distributed (MPI) running modes, the data input matrix is assumed to be distributed in between the MPI processes (nodes, processors, etc). The library routines allow for essentially any distribution of the data with a single constraint that a number of data points assigned to any process taking part in the calculation is not smaller than the half band-width of the Toeplitz matrix.

In all the cases, the layout of the output coincides with that of the input.

## 3.5 Availability and bug tracking

You can download the last release from the official website of the ANR-MIDAS'09 project at http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/software/mi

Please report any bugs via bug tracker at: http://code.google.com/p/cmb-da-library/

## 3.6 Installation

This software is reported to work on several Linux distributions and should work on any modern Unix-like system after minimal porting efforts.

The source code is delivered in a set of directories :

- The /src directory contains the sources files for the core library. It's composed by the differents modules of the MIDAS CMB DA library (please refer to the website for more details). You can directly compile theses files and link the generated binaries with your own program.

- The /test directory contains some Utility/demonstration programs to show some examples of how to use the library fonctionnalities.

## 3.7 User example

Here is a short example showing how to use it:

```
// sequential use
fftw_complex *V_fft, *T_fft;
double *V_rfft;
fftw_plan plan_f, plan_b;
tpltz_init(v1_size, lambda , &nfft, &blocksize, &T_fft, T, &V_fft, &V_rfft, &pla
     n_f, &plan_b);
stmm(V, n, m, id0, local_V_size, T_fft, lambda, V_fft, V_rfft, plan_f, plan_b, b
     locksize, nfft);
tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);


// MPI use
MPI_Scatterv(V, nranks, displs, MPI_DOUBLE, Vrank, maxsize, MPI_DOUBLE, 0, MPI_C
     OMM_WORLD);
mpi_stbmm(&Vrank, n, m, nrow, T, nb_blocks, nb_blocks, lambda, idv, id0, local_V
     _size, MPI_COMM_WORLD);
MPI_Gatherv(Vrank, nranks[rank], MPI_DOUBLE, TV, nranks, displs, MPI_DOUBLE, 0,
     MPI_COMM_WORLD);
```

# Chapter 4

# Module Index

## 4.1 Modules

Here is a list of all modules:

# Chapter 5

# Module Documentation

## 5.1 TOEPLITZ module

### Modules

- user interface (API)
- internal routines

#### 5.1.1 Detailed Description

Toeplitz matrix algebra module

## 5.2 user interface (API)

### Modules

- multithreaded/sequential routines
- distributed memory (MPI) routines

#### 5.2.1 Detailed Description

These routines provide main functionality of the Toeplitz algebra library. They are divided in two groups:

- shared-memory: multithreaded (openMP/sequential) routines
- distributed-memory (MPI) routines

## 5.3 multithreaded/sequential routines

### Functions

- int tpltz_init (int n, int lambda, int ∗nfft, int ∗blocksize, fftw_complex ∗∗T_fft, double ∗T, fftw_complex ∗∗V_fft, double ∗∗V_rfft, fftw_plan ∗plan_f, fftw_plan ∗plan_b)

  *Initialize block size and all the fftw arrays and plans needed for the computation.*
- int tpltz_cleanup (fftw_complex ∗∗T_fft, fftw_complex ∗∗V_fft, double ∗∗V_rfft, fftw_plan ∗plan_f, fftw_plan ∗plan_b)

  *Clean fftw workspace used in the Toeplitz matrix matrix product's computation.*
- int stmm_core (double ∗∗V, int n, int m, fftw_complex ∗T_fft, int blocksize, int lambda, fftw_complex ∗V_fft, double ∗V_rfft, int nfft, fftw_plan plan_f, fftw_plan plan_b, int flag_offset)

  *Perform the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm.*
- int stmm (double ∗∗V, int n, int m, int id0, int l, fftw_complex ∗T_fft, int lambda, fftw_complex ∗V_fft, double ∗V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft)

  *Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping.*
- int reset_gaps (double ∗∗V, int id0, int local_V_size, int m, int nrow, int ∗id0gap, int ∗lgap, int ngap)

  *Set the data to zeros at the gaps location.*
- int stbmm (double ∗∗V, int ∗n, int m, int nrow, double ∗T, int nb_blocks_local, int nb_blocks_all, int ∗lambda, int ∗idv, int idp, int local_V_size)

  *Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.*
- int gstbmm (double ∗∗V, int ∗n, int m, int nrow, double ∗T, int nb_blocks_local, int nb_blocks_all, int ∗lambda, int ∗idv, int id0p, int local_V_size, int ∗id0gap, int ∗lgap, int ngap)

  *Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.*

### 5.3.1 Detailed Description

These are shared-memory routines.

### 5.3.2 Function Documentation

#### 5.3.2.1 int gstbmm ( double ** *V,* int * *n,* int *m,* int *nrow,* double * *T,* int *nb_blocks_local,* int *nb_blocks_all,* int * *lambda,* int * *idv,* int *id0p,* int *local_V_size,* int * *id0gap,* int * *lgap,* int *ngap* )

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.

We first rebuild the Toeplitz block matrix structure to reduce the computation cost and skip the computations of the values on the defined gaps. then, each process performs the multiplication sequentially for each of the gappy block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] distributed data matrix (with the convention V(i,j)=V[i+j*n]) ; [out] result of the product TV |
| *n* | number of rows for each Toeplitz block as stored in T |
| *m* | number of columns of the global data matrix V |
| *nrow* | number of rows of the global data matrix V |
| *T* | Toeplitz matrix composed of the non-zero entries of the first row of each Toeplitz block and concatenated together have to be arranged in the increasing order of n without repetitions and overlaps. |
| *nb_blocks_all* | number of all Toeplitz block on the diagonal of the full Toeplitz matrix |
| *nb_blocks_local* | number of Toeplitz blocks as stored in T |
| *lambda* | half bandwith size for each Toeplitz block stroed in T |
| *idv* | global row index defining for each Toeplitz block as stored in the vector T first element of the interval to which given Toeplitz matrix is to be applied. |
| *id0p* | global index of the first element of the local part of V |
| *local_V_size* | number of all elements in local V |
| *id0gap* | index of the first element of each defined gap |
| *lgap* | length of each defined gaps |
| *ngap* | number of defined gaps |

Definition at line 253 of file toeplitz_seq.c.

#### 5.3.2.2 int reset_gaps ( double ** *V,* int *id0,* int *local_V_size,* int *m,* int *nrow,* int * *id0gap,* int * *lgap,* int *ngap* )

Set the data to zeros at the gaps location.

The data located within the gaps are set to zero. The gaps are defined in the time domain, meaning their indexes are defined in the row dimension.

Definition at line 1680 of file toeplitz.c.

#### 5.3.2.3 int stbmm ( double ** *V,* int * *n,* int *m,* int *nrow,* double * *T,* int *nb_blocks_local,* int *nb_blocks_all,* int * *lambda,* int * *idv,* int *idp,* int *local_V_size* )

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

Each process performs the multiplication sequentially for each diagonal block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. Each of the diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] distributed data matrix (with the convention V(i,j)=V[i+j*n]) ; [out] result of the product TV |
| *n* | number of rows for each Toeplitz block as stored in T |
| *m* | number of columns of the global data matrix V |
| *nrow* | number of rows of the global data matrix V |
| *T* | Toeplitz matrix composed of the non-zero entries of the first row of each Toeplitz block and concatenated together have to be arranged in the increasing order of n without repetitions and overlaps. |
| *nb_blocks_all* | number of all Toeplitz block on the diagonal of the full Toeplitz matrix |
| *nb_blocks_local* | number of Toeplitz blocks as stored in T |
| *lambda* | half bandwith size for each Toeplitz block stroed in T |
| *idv* | global row index defining for each Toeplitz block as stored in the vector T first element of the interval to which given Toeplitz matrix is to be applied. |
| *idp* | global index of the first element of the local part of V |
| *local_V_size* | a number of all elements in local V |

Definition at line 71 of file toeplitz_seq.c.

#### 5.3.2.4 int stmm ( double ** *V,* int *n,* int *m,* int *id0,* int *l,* fftw_complex * *T_fft,* int *lambda,* fftw_complex * *V_fft,* double * *V_rfft,* fftw_plan *plan_f,* fftw_plan *plan_b,* int *blocksize,* int *nfft* )

Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping.

The input matrix is formatted into an optimize matrix depending on the block size and the number of simultaneous ffts (defined with the variable nfft). The obtained number of columns represent the number of vectors FFTs of which are computed simulatenously. The multiplication is then performed block-by-block with the chosen block size using the core routine. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] data matrix (with the convention V(i,j)=V[i+j*n]) ; [out] result of the product TV |
| *n* | number of rows of V |

| | |
|---|---|
| *m* | number of columns of V |
| *id0* | first index of V |
| *l* | length of V |
| *T_fft* | complex array used for FFTs |
| *lambda* | Toeplitz band width |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |
| *blocksize* | block size |
| *nfft* | number of simultaneous FTTs |

Definition at line 833 of file toeplitz.c.

#### 5.3.2.5 int stmm_core ( double ∗∗ *V*, int *n*, int *m*, fftw_complex ∗ *T_fft*, int *blocksize*, int *lambda*, fftw_complex ∗ *V_fft*, double ∗ *V_rfft*, int *nfft*, fftw_plan *plan_f*, fftw_plan *plan_b*, int *flag_offset* )

Perform the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm.

The product is performed block-by-block with a defined block size or a computed optimized block size that reflects a trade off between cost of a single FFT of a length block_size and a number of blocks needed to perform the mutiplicaton. The latter determines how many spurious values are computed extra due to overlaps between the blocks. Use flag_offset=0 for "classic" algorithm and flag_offset=1 to put an offset to avoid the first and last lambdas terms. Usefull when a reshaping was done before with optimal column for a nfft. Better be inside the arguments of the routine. The parameters are:

**Parameters**

| | |
|---|---|
| *V* | [input] data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV |
| *n* | number of rows of V |
| *m* | number of columns of V |
| *T_fft* | complex array used for FFTs |
| *blocksize* | block size used in the sliding window algorithm |
| *lambda* | Toeplitz band width |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *nfft* | number of simultaneous FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |
| *flag_offset* | flag to avoid extra 2∗lambda padding to zeros on the edges |

Definition at line 515 of file toeplitz.c.

#### 5.3.2.6 int tpltz_cleanup ( fftw_complex ** *T_fft*, fftw_complex ** *V_fft*, double ** *V_rfft*, fftw_plan * *plan_f*, fftw_plan * *plan_b* )

Clean fftw workspace used in the Toeplitz matrix matrix product's computation.

Destroy fftw plans, free memory and reset fftw workspace.

**See also**

tpltz_init

**Parameters**

| | |
|---|---|
| *T_fft* | complex array used for FFTs |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |

Definition at line 324 of file toeplitz.c.

#### 5.3.2.7 int tpltz_init ( int *n*, int *lambda*, int * *nfft*, int * *blocksize*, fftw_complex ** *T_fft*, double * *T*, fftw_complex ** *V_fft*, double ** *V_rfft*, fftw_plan * *plan_f*, fftw_plan * *plan_b* )

Initialize block size and all the fftw arrays and plans needed for the computation.

Initialize the fftw arrays and plans is necessary before any computation of the Toeplitz matrix matrix product. Use tpltz_cleanup afterwards.

**See also**

tpltz_cleanup

**Parameters**

| | |
|---|---|
| *n* | row size of the matrix used for later product |
| *lambda* | Toeplitz band width |
| *nfft* | maximum number of FFTs you want to compute at the same time |
| *blocksize* | optimal block size used in the sliding window algorithm to compute an optimize value) |
| *T_fft* | complex array used for FFTs |
| *T* | Toeplitz matrix |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |

Definition at line 186 of file toeplitz.c.

## 5.4 distributed memory (MPI) routines

### Functions

- int mpi_stmm (double **V, int n, int m, int id0, int l, double *T, int lambda, MPI_Comm comm)

  *Perform the multiplication of a Toeplitz matrix by a matrix with MPI. We assume that the matrix has already been scattered.*
- int mpi_stbmm (double **V, int *n, int m, int nrow, double *T, int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int idp, int local_V_size, MPI_Comm comm)

  *Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.*
- int mpi_gstbmm (double **V, int *n, int m, int nrow, double *T, int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int id0p, int local_V_size, int *id0gap, int *lgap, int ngap, MPI_Comm comm)

  *Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.*

### 5.4.1 Detailed Description

These are distributed-memory routines.

### 5.4.2 Function Documentation

#### 5.4.2.1 int mpi_gstbmm ( double ** *V*, int * *n*, int *m*, int *nrow*, double * *T*, int *nb_blocks_local*, int *nb_blocks_all*, int * *lambda*, int * *idv*, int *id0p*, int *local_V_size*, int * *id0gap*, int * *lgap*, int *ngap*, MPI_Comm *comm* )

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.

We first rebuild the Toeplitz block matrix structure to reduce the computation cost and skip the computations of the values on the defined gaps. then, each process performs the multiplication sequentially for each of the gappy block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV |
| *n* | number of rows for each Toeplitz block as stored in T |
| *m* | number of columns of the global data matrix V |
| *nrow* | number of rows of the global data matrix V |
| *T* | Toeplitz matrix composed of the non-zero entries of the first row of each Toeplitz block and concatenated together have to be arranged in the increasing order of n without repetitions and overlaps. |
| *nb_blocks_all* | number of all Toeplitz block on the diagonal of the full Toeplitz matrix |
| *nb_blocks_local* | number of Toeplitz blocks as stored in T |
| *lambda* | half bandwith size for each Toeplitz block stroed in T |
| *idv* | global row index defining for each Toeplitz block as stored in the vector T first element of the interval to which given Toeplitz matrix is to be applied. |
| *id0p* | global index of the first element of the local part of V |
| *local_V_size* | number of all elements in local V |
| *id0gap* | index of the first element of each defined gap |
| *lgap* | length of each defined gaps |
| *ngap* | number of defined gaps |
| *comm* | MPI communicator |

Definition at line 1576 of file toeplitz.c.

#### 5.4.2.2 int mpi_stbmm ( double ∗∗ *V,* int ∗ *n,* int *m,* int *nrow,* double ∗ *T,* int *nb_blocks_local,* int *nb_blocks_all,* int ∗ *lambda,* int ∗ *idv,* int *idp,* int *local_V_size,* MPI_Comm *comm* )

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

Each process performs the multiplication sequentially for each diagonal block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. Each of the diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV |
| *n* | number of rows for each Toeplitz block as stored in T |
| *m* | number of columns of the global data matrix V |
| *nrow* | number of rows of the global data matrix V |
| *T* | Toeplitz matrix composed of the non-zero entries of the first row of each Toeplitz block and concatenated together have to be arranged in the increasing order of n without repetitions and overlaps. |
| *nb_blocks_all* | number of all Toeplitz block on the diagonal of the full Toeplitz matrix |
| *nb_blocks_local* | number of Toeplitz blocks as stored in T |

| | |
|---|---|
| *lambda* | half bandwith size for each Toeplitz block stroed in T |
| *idv* | global row index defining for each Toeplitz block as stored in the vector T first element of the interval to which given Toeplitz matrix is to be applied. |
| *idp* | global index of the first element of the local part of V |
| *local_V_size* | a number of all elements in local V |
| *comm* | MPI communicator |

Definition at line 1090 of file toeplitz.c.

#### 5.4.2.3 int mpi_stmm ( double ** *V*, int *n*, int *m*, int *id0*, int *l*, double * *T*, int *lambda*, MPI_Comm *comm* )

Perform the multiplication of a Toeplitz matrix by a matrix with MPI. We assume that the matrix has already been scattered.

The multiplication is performed using FFT applied to circulant matrix in order to diagonalized it. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] distributed data matrix (with the convention V(i,j)=V[i+j*n]); [out] result of the product TV |
| *n* | number of rows of V |
| *m* | number of columns of V |
| *id0* | first index of scattered V |
| *l* | length of the scattered V |
| *T* | Toeplitz matrix. |
| *lambda* | Toeplitz band width. |
| *comm* | communicator (usually MPI_COMM_WORLD) |

Definition at line 946 of file toeplitz.c.

## 5.5 internal routines

### Modules

- low-level routines
- lower internal routines

### 5.5.1 Detailed Description

These are auxiliary, internal routines, not intended to be used by no-expert user. They are divided in two groups:

- low level routines
- internal routines

## 5.6 low-level routines

### Functions

- int optimal_blocksize (int n, int lambda, int bs_flag)

  *Compute an optimal block size value used in the sliding windows algorithm.*
- int fftw_init_omp_threads ()

  *Initialize omp threads for fftw plans.*
- int rhs_init_fftw (int ∗nfft, int fft_size, fftw_complex ∗∗V_fft, double ∗∗V_rfft, fftw_plan ∗plan_f, fftw_plan ∗plan_b, int fftw_flag)

  *Initialize fftw array and plan for the right hand side matrix V.*
- int circ_init_fftw (double ∗T, int fft_size, int lambda, fftw_complex ∗∗T_fft)

  *Initialize fftw array and plan for the circulant matrix T_circ obtained from T.*
- int scmm_direct (int fft_size, fftw_complex ∗C_fft, int ncol, double ∗V_rfft, double ∗∗CV, fftw_complex ∗V_fft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  *Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.*
- int scmm_basic (double ∗∗V, int blocksize, int m, fftw_complex ∗C_fft, int lambda, double ∗∗CV, fftw_complex ∗V_fft, double ∗V_rfft, int nfft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  *Perform the product of a circulant matrix by a matrix using FFT's.*
- int stmm_reshape (double ∗∗V, int n, int m, int id0, int l, fftw_complex ∗T_fft, int lambda, fftw_complex ∗V_fft, double ∗V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft)

  *Reshape the data structure to optimize the Toeplitz matrix matrix computation by the sliding window algorithm and do the computation of the product using the core routine.*
- int build_gappy_blocks (int ∗n, int m, int nrow, double ∗T, int nb_blocks_local, int nb_blocks_all, int ∗lambda, int ∗idv, int ∗id0gap, int ∗lgap, int ngap, int ∗nb_blocks_gappy_final, double ∗Tgappy, int ∗idvgappy, int ∗ngappy, int ∗lambdagappy, int flag_param_distmin_fixed)

  *Build the gappy Toeplitz block structure to optimise the product computation at gaps location.*

### 5.6.1 Detailed Description

These are low-level routines.

### 5.6.2 Function Documentation

#### 5.6.2.1 int build_gappy_blocks ( int ∗ *n*, int *m*, int *nrow*, double ∗ *T*, int *nb_blocks_local*, int *nb_blocks_all*, int ∗ *lambda*, int ∗ *idv*, int ∗ *id0gap*, int ∗ *lgap*, int *ngap*, int ∗ *nb_blocks_gappy_final*, double ∗ *Tgappy*, int ∗ *idvgappy*, int ∗ *ngappy*, int ∗ *lambdagappy*, int *flag_param_distmin_fixed* )

Build the gappy Toeplitz block structure to optimise the product computation at gaps location.

Considering the significant gaps, the blocks to which they belong are cut and split between the gap's edges to reduce the total row size of the flotting blocks. It take into consideration the minimum correlation length and a parameter allows us to control the minimum gap size allowed for the blocks splitting. In some cases, the gap can be partially reduce to fit the minimum block size needed for computation or just for performance criteria. This is based on the fact that the gaps are set to zeros in the main routine.

Definition at line 1707 of file toeplitz.c.

#### 5.6.2.2 int circ_init_fftw ( double * *T,* int *fft_size,* int *lambda,* fftw_complex ** *T_fft* )

Initialize fftw array and plan for the circulant matrix T_circ obtained from T.

Build the circulant matrix T_circ from T and initilize his fftw arrays and plans. Use tpltz_cleanup afterwards.

**See also**

tpltz_cleanup

**Parameters**

| | |
|---|---|
| *T* | Toeplitz matrix. |
| *fft_size* | effective FFT size for the circulant matrix (usually equal to blocksize) |
| *lambda* | Toeplitz band width. |
| *T_fft* | complex array used for FFTs. |

Definition at line 281 of file toeplitz.c.

#### 5.6.2.3 int fftw_init_omp_threads ( )

Initialize omp threads for fftw plans.

Initialize omp threads for fftw plans. The number of threads used for ffts (define by the variable n_thread) is read from OMP_NUM_THREAD environment variable. fftw multithreaded option is controlled by fftw_MULTITHREADING macro.

Definition at line 217 of file toeplitz.c.

#### 5.6.2.4 int optimal_blocksize ( int *n,* int *lambda,* int *bs_flag* )

Compute an optimal block size value used in the sliding windows algorithm.

The optimal block size is computed as the minimum power of two above 3*lambda, i.e. the smallest value equal to 2^x, where x is an integer, and above 3*lambda. If bs_flag is set to one, a different formula is used to compute the optimal block size (see MADmap: A MASSIVELY PARALLEL MAXIMUM LIKELIHOOD COSMIC MICROWAVE BACKGROUND MAP-MAKER, C. M. Cantalupo, J. D. Borrill, A. H. Jaffe, T. S. Kisner, and R. Stompor, The Astrophysical Journal Supplement Series, 187:212–227, 2010 March). To avoid using block size much bigger than the matrix, the block size is set to

3∗lambda when his previous computed size is bigger than the matrix size n. This case append mostly for small matrix compared to his bandwith.

**Parameters**

| | |
|---|---|
| *n* | matrix row dimension |
| *lambda* | half bandwidth of the Toeplitz matrix |
| *bs_flag* | flag to use a different formula for optimal block size computation |

Definition at line 144 of file toeplitz.c.

#### 5.6.2.5 int rhs_init_fftw ( int ∗ *nfft*, int *fft_size*, fftw_complex ∗∗ *V_fft*, double ∗∗ *V_rfft*, fftw_plan ∗ *plan_f*, fftw_plan ∗ *plan_b*, int *fftw_flag* )

Initialize fftw array and plan for the right hand side matrix V.

Initialize fftw array and plan for the right hand side matrix V.

**Parameters**

| | |
|---|---|
| *nfft* | maximum number of FFTs you want to compute at the same time |
| *fft_size* | effective FFT size for the general matrix V (usually equal to blocksize) |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |
| *fftw_flag* | fftw plan allocation flag |

Definition at line 254 of file toeplitz.c.

#### 5.6.2.6 int scmm_basic ( double ∗∗ *V*, int *blocksize*, int *m*, fftw_complex ∗ *C_fft*, int *lambda*, double ∗∗ *CV*, fftw_complex ∗ *V_fft*, double ∗ *V_rfft*, int *nfft*, fftw_plan *plan_f_V*, fftw_plan *plan_b_CV* )

Perform the product of a circulant matrix by a matrix using FFT's.

This routine multiplies a circulant matrix, represented by C_fft, by a general matrix V, and stores the output as a matrix CV. In addition the routine requires two workspace objects, V_fft and V_rfft, to be allocated prior to a call to it as well as two fftw plans: one forward (plan_f_V), and one backward (plan_b_TV). The sizes of the input general matrix V and the ouput CV are given by blocksize rows and m columns. They are stored as a vector in the column-wise order. The circulant matrix, which is assumed to be band-diagonal with a band-width lambda, is represented by a Fourier transform with its coefficients stored in a vector C_fft (length blocksize). blocksize also defines the size of the FFTs, which will be performed and therefore this is the value which has to be used while creating the fftw plans and allocating the workspaces. The latter are given as: nfft∗(blocksize/2+1) for V_fft and nfft∗blocksize for V_rfft. The fftw plans should correspond to doing the transforms of nfft vectors simultaneously. Typically, the parameters of this routine are fixed by a preceding call to Toeplitz_init(). The parameters are :

**Parameters**

| | | |
|---|---|---|
| | *V* | matrix (with the convention V(i,j)=V[i+j∗n]) |
| | *blocksize* | row dimension of V |
| | *m* | column dimension of V |
| | *C_fft* | complex array used for FFTs (FFT of the Toeplitz matrix) |
| | *lambda* | half band width Toeplitz |
| out | *CV* | product of the circulant matrix C_fft by the matrix V_rfft |
| | *V_fft* | complex array used for FFTs |
| | *V_rfft* | real array used for FFTs |
| | *nfft* | number of simultaneous FFTs |
| | *plan_f_V* | fftw plan forward (r2c) |
| | *plan_b_CV* | fftw plan backward (c2r) |

Definition at line 449 of file toeplitz.c.

#### 5.6.2.7 int scmm_direct ( int *fft_size*, fftw_complex ∗ *C_fft*, int *ncol*, double ∗ *V_rfft*, double ∗∗ *CV*, fftw_complex ∗ *V_fft*, fftw_plan *plan_f_V*, fftw_plan *plan_b_CV* )

Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.

Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans: forward - plan_f_V; and backward - plan_b_CV. C_fft is a Fourier (complex representation of the circulant matrix) of length fft_size/2+1; V_rfft is a matrix with ncol columns and fft_size rows; V_fft is a workspace of fft_size/2+1 complex numbers as required by the backward FFT (plan_b_CV); CV is the output matrix of the same size as the input V_rfft one. The FFTs transform ncol vectors simultanously.

**Parameters**

| | | |
|---|---|---|
| | *fft_size* | row dimension |
| | *C_fft* | complex array used for FFTs |
| | *ncol* | column dimension |
| | *V_rfft* | real array used for FFTs |
| out | *CV* | product of the circulant matrix C_fft by the matrix V_rfft |
| | *V_fft* | complex array used for FFTs |
| | *plan_f_V* | fftw plan forward (r2c) |
| | *plan_b_CV* | fftw plan backward (c2r) |

Definition at line 397 of file toeplitz.c.

#### 5.6.2.8 int stmm_reshape ( double ∗∗ *V*, int *n*, int *m*, int *id0*, int *l*, fftw_complex ∗ *T_fft*, int *lambda*, fftw_complex ∗ *V_fft*, double ∗ *V_rfft*, fftw_plan *plan_f*, fftw_plan *plan_b*, int *blocksize*, int *nfft* )

Reshape the data structure to optimize the Toeplitz matrix matrix computation by the sliding window algorithm and do the computation of the product using the core routine.

The input matrix is formatted into an optimize matrix depending on the defined block size and the number of simultaneous ffts (defined as a variable nfft). The obtained number of

columns represent the number of vectors FFTs of which are computed simulatenously. The product is then performed block-by-block with the chosen block size using the core routine. The parameters are :

**Parameters**

| | |
|---|---|
| *V* | [input] data matrix (with the convention V(i,j)=V[i+j∗n]); [out] result of the product TV |
| *n* | number of rows of V |
| *m* | number of columns of V |
| *id0* | first index of V |
| *l* | length of V |
| *T_fft* | complex array used for FFTs |
| *lambda* | Toeplitz band width |
| *V_fft* | complex array used for FFTs |
| *V_rfft* | real array used for FFTs |
| *plan_f* | fftw plan forward (r2c) |
| *plan_b* | fftw plan backward (c2r) |
| *blocksize* | block size used in the sliding window algorithm |
| *nfft* | number of simultaneous FFTs |

you need to put flag_offset=0 as parameter for the stmm_core routine.

Definition at line 635 of file toeplitz.c.

## 5.7 lower internal routines

### Functions

- int print_error_message (int error_number, char const ∗file, int line)

  *Print error message corresponding to an error number.*

- int copy_block (int ninrow, int nincol, double ∗Vin, int noutrow, int noutcol, double ∗Vout, int inrow, int incol, int nblockrow, int nblockcol, int outrow, int outcol, double norm, int set_zero_flag)

  *Copy a matrix block from an input matrix inside an output matrix.*

- int vect2nfftblock (double ∗V1, int v1_size, double ∗V2, int fft_size, int nfft, int lambda)

  *convert the data vector structure into a matrix structure optimized for nfft*

- int nfftblock2vect (double ∗V2, int fft_size, int nfft, int lambda, double ∗V1, int v1_size)

  *convert the matrix structure optimized for nfft into the data vector structure*

- int get_overlapping_blocks_params (int nbloc, int ∗idv, int ∗n, int local_V_size, int nrow, int idp, int ∗idpnew, int ∗local_V_size_new, int ∗nnew, int ∗ifirstBlock, int ∗ilastBlock)

  *..Copy a matrix block from an input matrix inside an output matrix.*

### 5.7.1 Detailed Description

These are lower internal routines.

### 5.7.2 Function Documentation

#### 5.7.2.1 int copy_block ( int *ninrow,* int *nincol,* double * *Vin,* int *noutrow,* int *noutcol,* double * *Vout,* int *inrow,* int *incol,* int *nblockrow,* int *nblockcol,* int *outrow,* int *outcol,* double *norm,* int *set_zero_flag* )

Copy a matrix block from an input matrix inside an output matrix.

Copy a matrix block of a size nblockrow x nblockcol from the input matrix Vin (size ninrow x nincol) starting with the element (inrow, incol) to the output matrix Vout (size notrow x noutcol) starting with the element (outrow, outcol) after multiplying by norm. If the output matrix is larger than the block the extra elements are either left as they were on the input or zeroed if zero_flag is set to 1. If the block to be copied is larger than either the input or the output matrix an error occurs.

Definition at line 348 of file toeplitz.c.

#### 5.7.2.2 int get_overlapping_blocks_params ( int *nbloc,* int * *idv,* int * *n,* int *local_V_size,* int *nrow,* int *idp,* int * *idpnew,* int * *local_V_size_new,* int * *nnew,* int * *ifirstBlock,* int * *ilastBlock* )

..Copy a matrix block from an input matrix inside an output matrix.

Copies a matrix block of a size nblockrow x nblockcol from the input matrix Vin (size ninrow x nincol) starting with the element (inrow, incol) to the output matrix Vout (size notrow x noutcol) starting with the element (outrow, outcol) after multiplying by norm. If the output matrix is larger than the block the extra elements are either left as they were on the input or zeroed if zero_flag is set to 1. If the block to be copied is larger than either the input or the output matrix an error occurs.

Definition at line 1424 of file toeplitz.c.

#### 5.7.2.3 int nfftblock2vect ( double * *V2,* int *fft_size,* int *nfft,* int *lambda,* double * *V1,* int *v1_size* )

convert the matrix structure optimized for nfft into the data vector structure

Copy only the middle part of the matrix structure into the previous vector structure. Indeed, we don't need the extra terms located on the edges of each column used only to keep the correlation of the datas in the product computation.

Definition at line 786 of file toeplitz.c.

#### 5.7.2.4 int print_error_message ( int *error_number,* char const * *file,* int *line* )

Print error message corresponding to an error number.

**Parameters**

| | |
|---|---|
| *error_number* | error number |
| *file* | file name |
| *line* | line number |

Definition at line 108 of file toeplitz.c.

#### 5.7.2.5 int vect2nfftblock ( double * *V1,* int *v1_size,* double * *V2,* int *fft_size,* int *nfft,* int *lambda* )

convert the data vector structure into a matrix structure optimized for nfft

Copy the data vector structure into an equivalent matrix with nfft column. Thus, the obtained matrix is optimize for the nfft multithreading algorithm use. The middle part is a direct copy of the data vector and we copy on the edges of each column the lambda terms needed to fullfill the correlation of theses data.

Definition at line 740 of file toeplitz.c.