C++ Beamformer Library with RFI Mitigation

0.1.0

Generated by Doxygen 1.6.1

Thu Sep 8 16:09:41 2011

Contents

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

bf **3**

2 Data Structure Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 bf Namespace Reference

Data Structures

- struct ElementXYZ_tt
- class ArrayElements
- struct Beams_tt
- class BeamformerWeights
- class Covariance
- class CovarianceModifier
- class Decomposition
- class DecompositionAnalyzer
- class DecompositionModifier
- class QRDecomposition
- class EVDecomposition
- class SVDecomposition

Typedefs

- typedef struct bf::ElementXYZ_tt ElementXYZ_t
- typedef struct bf::Beams_tt Beams_t
- typedef arma::cx_double complex
- typedef double real

Functions

- std::ostream & operator<< (std::ostream &os, ArrayElements const &a)
- std::ostream & operator<< (std::ostream &os, Beams_t const &b)
- std::ostream & operator<< (std::ostream &os, BeamformerWeights const &w)
- std::ostream & operator<< (std::ostream &os, Covariance const &c)
- std::ostream & operator<< (std::ostream &os, Decomposition const &d)
- std::ostream & operator<< (std::ostream &os, QRDecomposition const &d)
- std::ostream & operator<< (std::ostream &os, EVDecomposition const &d)
- std::ostream & operator<< (std::ostream &os, SVDecomposition const &d)

5.1.1 Typedef Documentation

5.1.1.1 typedef struct bf::Beams_tt bf::Beams_t

Phased array, steering angles of individual beams

5.1.1.2 typedef arma::cx_double bf::complex

5.1.1.3 typedef struct bf::ElementXYZ_tt bf::ElementXYZ_t

Phased array element positions, array dimensions and element properties

5.1.1.4 typedef double bf::real

5.1.2 Function Documentation

5.1.2.1 std::ostream & bf::operator<< (std::ostream & *os*, SVDecomposition const & *d*)

Human-readable data output to stream

5.1.2.2 std::ostream & bf::operator<< (std::ostream & *os*, EVDecomposition const & *d*)

Human-readable data output to stream

5.1.2.3 std::ostream & bf::operator<< (std::ostream & *os*, QRDecomposition const & *d*)

Human-readable data output to stream

5.1.2.4 std::ostream & bf::operator<< (std::ostream & *os*, Decomposition const & *d*)

Human-readable data output to stream

5.1.2.5 std::ostream & bf::operator<< (std::ostream & *os*, Covariance const & *c*)

Human-readable data output to stream

5.1.2.6 std::ostream & bf::operator<< (std::ostream & *os*, BeamformerWeights const & *w*)

Human-readable data output to stream

5.1.2.7 std::ostream & bf::operator<< (std::ostream & *os*, Beams_t const & *b*)

Human-readable data output to stream

5.1.2.8 std::ostream & bf::operator<< (std::ostream & *os*, ArrayElements const & *a*)

Human-readable data output to stream

6 Data Structure Documentation

6.1 bf::ArrayElements Class Reference

Public Types

- enum Pointing { POINT_ASTRO = 0, POINT_RFI_REFERENCE = 128 }
- enum Polarization { POL_LCP = 0, POL_RCP = 1 }

Public Member Functions

- ArrayElements ()
- ∼ArrayElements ()
- const ElementXYZ_t & getPositionSet () const
- arma::Col< int > listReferenceAntennas ()
- int setFlags (const int ielem, const int flags)
- const int getFlags (const int ielem) const
- const int Nant () const
- void generateLinear (int Nant, double spacing)
- void generateGrid (int Nant, double spacing)

Friends

- std::ostream & operator<< (std::ostream &, ArrayElements const &)

6.1.1 Detailed Description

Class for creating and storing the (x,y,z) coordinates of all elements in a phased array.

6.1.2 Member Enumeration Documentation

6.1.2.1 enum bf::ArrayElements::Pointing

Enumerator:

POINT_ASTRO

POINT_RFI_REFERENCE

6.1.2.2 enum bf::ArrayElements::Polarization

Enumerator:

POL_LCP

POL_RCP

6.1.3 Constructor & Destructor Documentation

6.1.3.1 bf::ArrayElements::ArrayElements () `[inline]`

C'stor for initialization.

6.1.3.2 bf::ArrayElements::∼ArrayElements () `[inline]`

D'stor

6.1.4 Member Function Documentation

6.1.4.1 void bf::ArrayElements::generateGrid (int *Nant*, double *spacing*)

Generate a 2D equispaced square grid antenna array centered around origo with antennas placed onto first 2 dimensions (x,y plane)

Parameters:

← ***Nant*** Total number of antennas

← ***spacing*** Antenna spacing in meters

6.1.4.2 void bf::ArrayElements::generateLinear (int *Nant*, double *spacing*)

Generate a 1D equispaced linear antenna array centered around origo with antennas placed along the first dimension (x axis).

Parameters:

← ***Nant*** Total number of antennas

← ***spacing*** Antenna spacing in meters

6.1.4.3 const int bf::ArrayElements::getFlags (const int *ielem*) const

Accessor to array element flags

Parameters:

← ***ielem*** The index of the element 0..N-1

Returns:

Flag value of the element or -1 on error

6.1.4.4 const ElementXYZ_t& bf::ArrayElements::getPositionSet () const `[inline]`

Accessor function to element data struct

Returns:

ref to struct that contains antenna element coordinates

6.1.4.5 arma::Col< int > bf::ArrayElements::listReferenceAntennas ()

Assemble and return a list of reference antenna indices.

Returns:

Column vector with indices of all antennas currently flagged as RFI reference.

6.1.4.6 const int bf::ArrayElements::Nant () const `[inline]`

Accessor to get number of elements in array.

6.1.4.7 int bf::ArrayElements::setFlags (const int *ielem*, const int *flags*)

Set flags on element, overwrites earlier flags.

Parameters:

← ***ielem*** The index of the element 0..N-1

← ***flags*** The value to set, consisting of OR'ed flags

Returns:

Old flag value before it was overwritten or -1 on error Flags are POINT_ASTRO, POINT_RFI_-REFERENCE, POL_LCP and POL_LCP.

6.1.5 Friends And Related Function Documentation

6.1.5.1 std::ostream& operator<< (std::ostream &, ArrayElements const &) `[friend]`

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

- ArrayElements.h
- ArrayElements.cpp

6.2 bf::BeamformerWeights Class Reference

Public Member Functions

- BeamformerWeights ()
- ~BeamformerWeights ()
- void generateSteerings (Beams_t &beams, ArrayElements const &ae) const
- void generateMVDR (Beams_t const &beams, Covariance const &cov, const double b)
- void generateMVDR (Beams_t const &beams, Decomposition const &deco, const double b)
- const arma::Cube< bf::complex > & getWeights () const
- const arma::Mat< bf::complex > & getWeights (int beam) const

Friends

- std::ostream & operator<< (std::ostream &, BeamformerWeights const &)

6.2.1 Constructor & Destructor Documentation

6.2.1.1 bf::BeamformerWeights::BeamformerWeights () `[inline]`

6.2.1.2 bf::BeamformerWeights::∼BeamformerWeights () `[inline]`

6.2.2 Member Function Documentation

6.2.2.1 void bf::BeamformerWeights::generateMVDR (Beams_t const & *beams*, Decomposition const & *deco*, const double *b*)

Compute MVDR weights using specified steering and decomposed covariance matrix. Pre-decomposed covariance speeds up computation of the (pseudo)inverse needed by MVDR.

The factor 'b' determines the type of beamforming. With b==0 weights are classical, non-adaptive. With 1=>b>0 weights shift towards MVDR (fully MVDR at b==1). With b>1 weights are MVDR Cox Projection WNGC.

Parameters:

← ***beams*** The Beams_t struct with beam steerings to use for MVDR weights
← ***deco*** Some decomposition of the covariance data.
← ***b*** Factor for Cox Projection WNGC

Compute MVDR weights using specified steering and decomposed covariance matrix. Pre-decomposed covariance speeds up computation of the (pseudo)inverse needed by MVDR.

The factor 'b' determines the type of beamforming. With b==0 weights are classical, non-adaptive. With 1=>b>0 weights shift towards MVDR (fully MVDR at b==1). With b>1 weights are MVDR Cox Projection WNGC.

Parameters:

← ***beams*** The Beams_t struct with beam steerings to use for MVDR weights
← ***deco*** Some decomposition of the covariance data.
← ***b*** Factor for Cox Projection WNGC.

6.2.2.2 void bf::BeamformerWeights::generateMVDR (Beams_t const & *beams*, Covariance const & *cov*, const double *b*)

Compute MVDR weights using specified steering and covariance matrix. The factor 'b' determines the type of beamforming. With b==0 weights are classical, non-adaptive. With 1=>b>0 weights shift towards MVDR (fully MVDR at b==1). With b>1 weights are MVDR Cox Projection WNGC.

Parameters:

← ***beams*** The Beams_t struct with beam steerings to use for MVDR weights
← ***cov*** The covariance data to use
← ***b*** Factor for Cox Projection WNGC

6.2.2.3 void bf::BeamformerWeights::generateSteerings (Beams_t & *beams*, ArrayElements const & *ae*) const

Calculate steering vectors for the angles of each beam in the beams list, using the specified array element positions. Steering vectors are the phase delay of a plane wave experienced at each individual element of the array.

Parameters:

$\leftrightarrow$ ***beams*** The Beams_t struct whose angles to convert into complex steerings

$\leftarrow$ ***ae*** Array element positions

Returns:

Steerings written back into the beams_t struct

Calculate steering vectors for the angles of each beam in the beams list, using the specified array element positions and channel frequencies. Steering vectors are the phase delay of a plane wave experienced at each individual element of the array.

Parameters:

$\leftrightarrow$ ***beams*** The Beams_t struct whose angles to convert into complex steerings

$\leftarrow$ ***ae*** Array element positions

Returns:

Steerings written back into the beams_t struct

6.2.2.4 const arma::Mat<bf::complex>& bf::BeamformerWeights::getWeights (int *beam*) const `[inline]`

Accessor to computed weights of one beam.

Parameters:

$\leftarrow$ ***beam*** Index of the beam in 0..Nbeams-1

Returns:

reference to 2D matrix of size Nbeams x Nant

6.2.2.5 const arma::Cube<bf::complex>& bf::BeamformerWeights::getWeights () const `[inline]`

Accessor to computed weights.

Returns:

reference to 3D cube of size Nbeams x Nant x Nchan

6.2.3 Friends And Related Function Documentation

6.2.3.1 std::ostream& operator<< (std::ostream &, BeamformerWeights const &) `[friend]`

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

- BeamformerWeights.h
- BeamformerWeights.cpp

6.3 bf::Beams_tt Struct Reference

Public Member Functions

- void init ()
- void init (const int Nb, const int Na, const int Nc)

Data Fields

- int Nbeams

  Number of beams. Equal to the expected number of items in the phi[], theta[] angles list.

- int Nchan

  Number of channels in the data, must be at least 1 and each needs an entry in the freqs[] list.

- int Nant

  Number of antennas in the array, including RFI reference antennas.

- arma::Col< double > phi

  Beam steering angle phi in radians, azimuth angle of the signal.

- arma::Col< double > theta

  Beam steering angle theta in radians, tilt angle of plane wave normal from zenith.

- arma::Col< double > freqs

  List of channel frequencies in Hertz.

- arma::Cube< bf::complex > steerings

  Matrix with pre-computed steerings (Nbeams x Nant x Nchan).

6.3.1 Detailed Description

Phased array, steering angles of individual beams

6.3.2 Member Function Documentation

6.3.2.1 void bf::Beams_tt::init (const int *Nb*, const int *Na*, const int *Nc*) `[inline]`

Change dimensions of the data storage and set all data to zero.

Parameters:

← ***Nb***	Number of beams
← ***Na***	Number of antennas
← ***Nc***	Number of channels

6.3.2.2 void bf::Beams_tt::init () `[inline]`

Set all current data to zero.

6.3.3 Field Documentation

6.3.3.1 arma::Col<double> bf::Beams_tt::freqs

List of channel frequencies in Hertz.

6.3.3.2 int bf::Beams_tt::Nant

Number of antennas in the array, including RFI reference antennas.

6.3.3.3 int bf::Beams_tt::Nbeams

Number of beams. Equal to the expected number of items in the phi[], theta[] angles list.

6.3.3.4 int bf::Beams_tt::Nchan

Number of channels in the data, must be at least 1 and each needs an entry in the freqs[] list.

6.3.3.5 arma::Col<double> bf::Beams_tt::phi

Beam steering angle phi in radians, azimuth angle of the signal.

6.3.3.6 arma::Cube<bf::complex> bf::Beams_tt::steerings

Matrix with pre-computed steerings (Nbeams x Nant x Nchan).

See also:

BeamformerWeights::generateSteerings()

6.3.3.7 arma::Col<double> bf::Beams_tt::theta

Beam steering angle theta in radians, tilt angle of plane wave normal from zenith.

The documentation for this struct was generated from the following file:

- BeamformerData.h

6.4 bf::Covariance Class Reference

Public Member Functions

- Covariance (int Nant, int Nchannels, int Msmp, double timestamp, double Tint)
- ∼Covariance ()
- Covariance (const Covariance &o)
- void resize (const int Nant, const int Nchannels)
- const arma::Cube< bf::complex > & get () const
- arma::Cube< bf::complex > & getWriteable ()
- void load (double ∗raw_data, const int format)
- void load (const char ∗fn, const int format)
- void store (const char ∗fn, const int format) const
- void addSignal (int ch, double lambda, ArrayElements const &ae, double phi, double theta, double P, double Pna, double Pnc)
- void addSignal (int ch, double lambda, ArrayElements const &ae, double phi, double theta, double P, double Pna, double Pnc, double Gref, arma::Col< int > const &Iref)
- Covariance & operator+= (const Covariance &rhs)
- void add (const int ch, arma::Col< bf::complex > const &x)
- const int N_ant (void) const

  Accessor to get number of antennas in contained data.

- const int N_chan (void) const

  Accessor to get number of channels in contained data.

- const int M_smp (void) const

  Accessor to get count of time-integrated covariances in contained data.

- const double channel_freq (const int ch)

Friends

- std::ostream & operator<< (std::ostream &, Covariance const &)

6.4.1 Detailed Description

Provides storage for time-integrated covariance data in the form of a 3D data cube (Nantennas x Nantennas x Nchannels). In case of Fourier domain data, the stored data would be called spectral density matrices instead of covariances.

The covariance data contained in this class can be analyzed and modified by other classes, for example for performing adaptive beamforming, RFI template subtraction, or decomposition-based RFI mitigation of the covariance data itself.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 bf::Covariance::Covariance (int *Nant*, int *Nchannels*, int *Msmp*, double *timestamp*, double *Tint*) `[inline]`

C'stor, allocate space for covariances and set their starting timestamp as well as integration time.

Parameters:

← ***Nant*** Number of elements or antennas.

← ***Nchannels*** Number of frequency channels.

← ***Msmp*** Number of sample vectors (x(t)'∗x(t) matrices) that were averaged

← ***timestamp*** Starting time of the data cube.

← ***Tint*** Integration time used for the data cube.

6.4.2.2 bf::Covariance::∼Covariance () `[inline]`

D'stor

6.4.2.3 bf::Covariance::Covariance (const Covariance & *o*) `[inline]`

Copy c'stor, make new covariance object with copy of data from another.

6.4.3 Member Function Documentation

6.4.3.1 void bf::Covariance::add (const int *ch*, arma::Col< bf::complex > const & *x*) `[inline]`

Given a signal vector of Nant elements, calculates covariances and integrates this into the current covariance data of the specified channel (Rxx[ch] = Rxx[ch] + x'∗x).

Parameters:

← ***ch*** Target covariance channel 0..Nch-1

← ***x*** Column vector with data from Nant elements

6.4.3.2 void bf::Covariance::addSignal (int *ch*, double *lambda*, ArrayElements const & *ae*, double *phi*, double *theta*, double *P*, double *Pna*, double *Pnc*, double *Gref*, arma::Col< int > const & *Iref*)

Add an artificial signal to the covariance matrix, separating the array into a set of normal elements and a set of RFI-only reference antennas.

Parameters:

← ***ch*** Channel number

← ***lambda*** Wavelength in meters

← ***ae*** ArrayElement object with element positions

← ***phi*** Azimuth angle of signal

← ***theta*** Tilt angle of plane wave normal from zenith

← ***P*** Signal power

← ***Pna*** Internal noise power (added to autocorrelations)

← ***Pnc*** Correlated noise power (added to cross and auto)

← ***Gref*** Reference antenna gain over array element gain

← ***Iref*** Vector with reference antenna indices between 0:(Nant-1)

6.4.3.3 void bf::Covariance::addSignal (int *ch*, double *lambda*, ArrayElements const & *ae*, double *phi*, double *theta*, double *P*, double *Pna*, double *Pnc*)

Add an artificial signal to the covariance matrix

Parameters:

← ***ch*** Channel number

← ***lambda*** Wavelength in meters

← ***ae*** ArrayElement object with element positions

← ***phi*** Azimuth angle of signal

← ***theta*** Tilt angle of plane wave normal from zenith

← ***P*** Signal power

← ***Pna*** Internal noise power (added to autocorrelations)

← ***Pnc*** Correlated noise power (added to cross and auto)

6.4.3.4 const double bf::Covariance::channel_freq (const int *ch*) `[inline]`

Accessor to get frequency of one channel.

Parameters:

← ***ch*** Number of channel 0..Nchan-1

Returns:

Frequency in Hertz

6.4.3.5 const arma::Cube<bf::complex>& bf::Covariance::get () const `[inline]`

Const accessor to data cube.

Returns:

Const reference to covariance data cube.

6.4.3.6 arma::Cube<bf::complex>& bf::Covariance::getWriteable () `[inline]`

Writeable reference to data cube.

Returns:

Reference to covariance data cube.

6.4.3.7 void bf::Covariance::load (const char $*$ *fn*, const int *format*)

Load data cube contents from a file and reorganize the memory layout if necessary.

Parameters:

← ***fn*** Input file name and path
← ***format*** Data format (0..N, to be defined)

6.4.3.8 void bf::Covariance::load (double $*$ *raw_data*, const int *format*)

Load data cube contents from a memory location and reorganize the memory layout if necessary. Currently a stub that generates test data. Modify this for your custom data e.g. for receiving GPU beamforming cluster covariance output.

Parameters:

← ***raw_data*** Pointer to data to load
← ***format*** Data format (0..N, to be defined)

6.4.3.9 const int bf::Covariance::M_smp (void) const `[inline]`

Accessor to get count of time-integrated covariances in contained data.

6.4.3.10 const int bf::Covariance::N_ant (void) const `[inline]`

Accessor to get number of antennas in contained data.

6.4.3.11 const int bf::Covariance::N_chan (void) const `[inline]`

Accessor to get number of channels in contained data.

6.4.3.12 Covariance& bf::Covariance::operator+= (const Covariance & *rhs*) `[inline]`

Sums the data of another covariance object into the data cube contained in this object.

6.4.3.13 void bf::Covariance::resize (const int *Nant*, const int *Nchannels*) `[inline]`

Clear all contained data and set new size.

Parameters:

← ***Nant*** Number of elements or antennas.

← ***Nchannels*** Number of frequency channels.

6.4.3.14 void bf::Covariance::store (const char ∗ *fn*, const int *format*) const

Store data cube contents into a file.

Parameters:

← ***fn*** Output file name and path

← ***format*** Data format (0..N, to be defined)

6.4.4 Friends And Related Function Documentation

6.4.4.1 std::ostream& operator<< (std::ostream &, Covariance const &) `[friend]`

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

- Covariance.h
- Covariance.cpp

6.5 bf::CovarianceModifier Class Reference

Public Member Functions

- CovarianceModifier (Covariance &cov)
- ∼CovarianceModifier ()
- int templateSubtraction (arma::Col< int > const &Iref, const int Nrfi)
- int templateSubtraction (arma::Col< int > const &Iref, const int Nrfi, const int startch, const int endch)

6.5.1 Detailed Description

Set of non-toxic RFI mitigation algorithms that are applied to the raw covariance data directly. Currently this includes two RFI reference antenna methods for RFI templating and subtraction. They are applicable for all RFI types including sporadic, dynamic environments with beamformer adaptive nulling, but might work less well in pulsar observations.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 bf::CovarianceModifier::CovarianceModifier (Covariance & *cov*) `[inline]`

C'stor. Ties object to Covariance class to be modified.

Parameters:

← ***cov*** Covariance class to modify

6.5.2.2 bf::CovarianceModifier::∼CovarianceModifier () `[inline]`

D'stor

6.5.3 Member Function Documentation

6.5.3.1 int bf::CovarianceModifier::templateSubtraction (arma::Col< int > const & *Iref*, const int *Nrfi*, const int *startch*, const int *endch*)

Attempts to clean RFI in a subset of channels of a Covariance data set. See templateSubtraction(arma::Col, in) for details on processing.

Parameters:

← ***Iref*** List of reference antenna indices between 0..Nant-1

← ***Nrfi*** Expected number of strong RFI signals per channel

← ***startch*** First channel 0..Nch-1 to process

← ***endch*** Last channel 0..Nch-1 to process (inclusive)

Returns:

Returns 0 on success

6.5.3.2 int bf::CovarianceModifier::templateSubtraction (arma::Col< int > const & *Iref*, const int *Nrfi*)

Attempts to clean RFI from a Covariance data set.

Uses two or more RFI reference antennas and subtracts the weaker RFI seen by the array elements themselves from the covariance matrix.

The reference antennas need to be present in the covariance matrix. The full cross-correlation data between reference antennas and array elements must be present in the covariance matrix.

For efficiency, cross-correlation matrix layout needs to be such that RFI reference antennas come first. Thus antennas with low indices (0, 1, 2, ...) must be the RFI reference antennas. All remaining antennas must be elements of the array.

If interferers are not overlapping in frequency, there is at most one interferer per channel.

At cost of higher noise, more than one RFI per channel can be handled, provided that Nrfi <= N_ref_-antennas;

If Nrfi > N_ref_antennas, no subtraction is possible.

Correction bias: any correlated noise in the reference antenna data adds bias to the correction. This can be avoided by not spatially co-locating the reference antennas.

Toxicity: at low INR or no RFI presence, the correction reduces to zero, the used algorithms are safe.

Applicability: all RFI including sporadic, dynamic environments with beamformer adaptive nulling, but not for Pulsar observations.

Parameters:

← ***Iref*** List of reference antenna indices between 0..Nant-1

← ***Nrfi*** Expected number of strong RFI signals per channel

Returns:

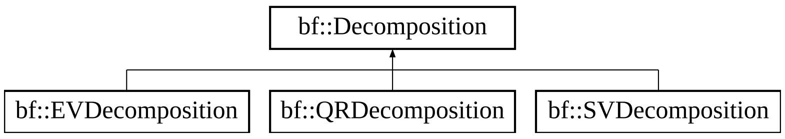
Returns 0 on success

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

- CovarianceModifier.h
- CovarianceModifier.cpp

6.6 bf::Decomposition Class Reference

Inheritance diagram for bf::Decomposition::



Public Types

- enum Type { None = -1, EVD = 0, SVD = 1, QR = 2 }

  Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Public Member Functions

- Decomposition (Covariance &Rxx)
- Decomposition (arma::Mat< bf::complex > &Rxx)
- ∼Decomposition ()
- void set_M_smp (int M_smp_new)
- int decompose (Covariance const &cov)
- int decompose (Covariance const &cov, const int startch, const int endch)
- int decompose (arma::Mat< bf::complex > const &Rxx)
- int recompose (Covariance &cov)
- int recompose (Covariance &cov, const int startch, const int endch)
- int recompose (arma::Mat< bf::complex > &Rxx)

Data Fields

- const int N_ant
- const int N_chan
- int M_smp
- int _deco_type

Friends

- std::ostream & operator<< (std::ostream &, Decomposition const &)

6.6.1 Detailed Description

Base class for computing matrix decompositions of a 3D data cube or single 2D matrix. The base class provides only the generic interface implementation and allocation functions, it does not perform an actual matrix decomposition. Functions allow to process subsets of the 3D data cube, this allows multithreading and threads to work on different subsets in parallel.

6.6.2 Member Enumeration Documentation

6.6.2.1 enum bf::Decomposition::Type

Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Enumerator:

None

EVD

SVD

QR

6.6.3 Constructor & Destructor Documentation

6.6.3.1 bf::Decomposition::Decomposition (Covariance & *Rxx*) `[inline]`

Base class C'stor for batch decomposition. Allocates sufficient internal space to compute and store decomposition results of multiple covariance matrices that are found in the Rxx Covariance object. The dimensions and properties of the specified Rxx determine the required amount of internal memory.

Calls to other member functions should either use the same Rxx passed to this c'stor, or must use some other Covariance data of equal size.

Parameters:

← ***Rxx*** Reference to covariance class

6.6.3.2 bf::Decomposition::Decomposition (arma::Mat< bf::complex > & *Rxx*) `[inline]`

Base class C'stor for decomposition. Allocates sufficient internal space to compute and store decomposition results of a single covariance matrix.

Calls to other member functions should either use the same Rxx passed to this c'stor, or must use some other matrix of equal size.

Parameters:

$\leftarrow$ ***Rxx*** Reference to raw covariance data

6.6.3.3 bf::Decomposition::∼Decomposition () `[inline]`

Base class D'stor to free internal allocations.

6.6.4 Member Function Documentation

6.6.4.1 int bf::Decomposition::decompose (arma::Mat< bf::complex > const & *Rxx*) `[inline]`

Perform single decomposition of given covariance matrix class.

Parameters:

$\leftarrow$ ***Rxx*** The covariance matrix to decompose.

Returns:

0 on success.

6.6.4.2 int bf::Decomposition::decompose (Covariance const & *cov*, const int *startch*, const int *endch*)

Perform batch decomposition of a range of covariances in the argument class.

Parameters:

$\leftarrow$ ***cov*** The covariance class with one or more matrices.

$\leftarrow$ ***startch*** Channel at which to start decomposition, 0 is first

$\leftarrow$ ***endch*** Last channel (inclusive) to decompose

Returns:

0 on success.

6.6.4.3 int bf::Decomposition::decompose (Covariance const & *cov*)

Perform batch decomposition of all covariances in the argument class.

Parameters:

$\leftarrow$ ***cov*** The covariance class with one or more matrices.

Returns:

0 on success.

Perform batch decomposition of all covariances in the argument class.

Parameters:

$\leftarrow$ ***cov*** The covariance class with one or more matrices.

Returns:

0 on success

6.6.4.4 int bf::Decomposition::recompose (arma::Mat< bf::complex > & *Rxx*) `[inline]`

Recompute one covariance matrix based on the first (0-channel) decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***Rxx*** Output covariance matrix.

Returns:

0 on success

6.6.4.5 int bf::Decomposition::recompose (Covariance & *cov*, const int *startch*, const int *endch*)

Perform batch recomposition of a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

$\leftarrow$ ***startch*** Channel at which to start recomposition, 0 is first

$\leftarrow$ ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

Batch recompute a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

$\leftarrow$ ***startch*** Channel at which to start recomposition, 0 is first

$\leftarrow$ ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

6.6.4.6 int bf::Decomposition::recompose (Covariance & *cov*)

Perform batch recomposition of all covariance matrices based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

Returns:

0 on success

6.6.4.7 void bf::Decomposition::set_M_smp (int *M_smp_new*) `[inline]`

Accessors

6.6.5 Friends And Related Function Documentation

6.6.5.1 std::ostream& operator<< (std::ostream &, Decomposition const &) `[friend]`

Reimplemented in bf::QRDecomposition, bf::EVDecomposition, and bf::SVDecomposition.

6.6.6 Field Documentation

6.6.6.1 int bf::Decomposition::_deco_type

6.6.6.2 int bf::Decomposition::M_smp

6.6.6.3 const int bf::Decomposition::N_ant

6.6.6.4 const int bf::Decomposition::N_chan

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

- Decomposition.h
- Decomposition.cpp

6.7 bf::DecompositionAnalyzer Class Reference

Public Member Functions

- DecompositionAnalyzer (Decomposition const &deco)
- DecompositionAnalyzer (SVDecomposition const &deco)
- DecompositionAnalyzer (EVDecomposition const &deco)
- ~DecompositionAnalyzer ()
- double getMDL (int channel, const int M_smp, int &rank) const
- double getAIC (int channel, const int M_smp, int &rank) const
- double getMDL (int channel, const int M_smp, const int Ndiscard, int &rank) const
- double getAIC (int channel, const int M_smp, const int Ndiscard, int &rank) const
- double getMDL (int c, int &ref) const
- double getAIC (int c, int &ref) const
- double get3Sigma (int channel, int &rank) const
- double get3Sigma (int channel, const int Ndiscard, int &rank) const
- bool utest ()

6.7.1 Detailed Description

Helper class to pull information out of a covariance matrix decomposition.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 bf::DecompositionAnalyzer::DecompositionAnalyzer (Decomposition const & *deco*)

C'stor

Parameters:

← ***deco*** Reference to decomposition results to analyze

6.7.2.2 bf::DecompositionAnalyzer::DecompositionAnalyzer (SVDecomposition const & *deco*)

C'stor

Parameters:

← ***deco*** Reference to decomposition results to analyze

6.7.2.3 bf::DecompositionAnalyzer::DecompositionAnalyzer (EVDecomposition const & *deco*)

C'stor

Parameters:

← ***deco*** Reference to decomposition results to analyze

6.7.2.4 bf::DecompositionAnalyzer::~DecompositionAnalyzer () `[inline]`

D'stor

6.7.3 Member Function Documentation

6.7.3.1 double bf::DecompositionAnalyzer::get3Sigma (int *channel*, const int *Ndiscard*, int & *rank*) const

Three sigma thresholding detector to make a guess at the number of eigenvalues that are above an unknown noise power threshold.

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***Ndiscard*** Number of smallest eigenvalues to ignore in 3sigma.

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the estimated number of interferers.

Three sigma thresholding detector to make a guess at the number of eigenvalues that are above an unknown noise power threshold. When it is known that some elements of the array have no signal, the corresponding lowest eigenvalues can be ignored using Ndiscard.

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***Ndiscard*** Number of smallest eigenvalues to ignore in 3sigma.

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the estimated number of interferers.

6.7.3.2 double bf::DecompositionAnalyzer::get3Sigma (int *channel*, int & *rank*) const `[inline]`

Three sigma thresholding detector to make a guess at the number of eigenvalues that are above an unknown noise power threshold.

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the estimated number of interferers.

6.7.3.3 double bf::DecompositionAnalyzer::getAIC (int *c*, int & *ref*) const `[inline]`

Wraps getAIC() call with M_smp stored in decomposition object

6.7.3.4 double bf::DecompositionAnalyzer::getAIC (int *channel*, const int *M_smp*, const int *Ndiscard*, int & *rank*) const

An AIC detector that makes a guess at the number of eigenvalues that are above an unknown noise power threshold. This can work reasonably but requires more than N/2 of eigenvalues are indeed from noise space. When it is known that some elements of the array have no signal, the corresponding lowest eigenvalues can be ignored using Ndiscard. Finds rank = arg min(AIC(k)|k=0..Nant-Ndiscard-1).

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***M_smp*** Number of samples (x(t)'∗x(t) matrices) that were averaged before decomposition

← ***Ndiscard*** Number of smallest eigenvalues to ignore in MDL.

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the minimum detected MDL value.

6.7.3.5 double bf::DecompositionAnalyzer::getAIC (int *channel*, const int *M_smp*, int & *rank*) const `[inline]`

An AIC detector that makes a guess at the number of eigenvalues that are above an unknown noise power threshold. This can work reasonably but requires more than N/2 of eigenvalues are indeed from noise space. Finds rank = arg min(AIC(k)|k=0..Nant-1).

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***M_smp*** Number of samples (x(t)'∗x(t) matrices) that were averaged before decomposition

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the minimum detected MDL value.

6.7.3.6 double bf::DecompositionAnalyzer::getMDL (int *c*, int & *ref*) const `[inline]`

Wraps getMDL() call with M_smp stored in decomposition object

6.7.3.7 double bf::DecompositionAnalyzer::getMDL (int *channel*, const int *M_smp*, const int *Ndiscard*, int & *rank*) const

An MDL detector that makes a guess at the number of eigenvalues that are above an unknown noise power threshold. This can work reasonably but requires more than N/2 of eigenvalues are indeed from noise space. When it is known that some elements of the array have no signal, the corresponding lowest eigenvalues can be ignored using Ndiscard. Finds rank = arg min(MDL(k)|k=0..Nant-Ndiscard-1).

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***M_smp*** Number of samples (x(t)'∗x(t) matrices) that were averaged before decomposition

← ***Ndiscard*** Number of smallest eigenvalues to ignore in MDL.

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the minimum detected MDL value.

6.7.3.8 double bf::DecompositionAnalyzer::getMDL (int *channel*, const int *M_smp*, int & *rank*) const `[inline]`

An MDL detector that makes a guess at the number of eigenvalues that are above an unknown noise power threshold. This can work reasonably but requires more than N/2 of eigenvalues are indeed from noise space. Finds rank = arg min(MDL(k)|k=0..Nant-1).

Parameters:

← ***channel*** Which channel of multi-channel data to analyse

← ***M_smp*** Number of samples (x(t)'∗x(t) matrices) that were averaged before decomposition

↔ ***rank*** Final determined interference space rank (0..Nch-1), 0 for no RFI found

Returns:

Returns the minimum detected MDL value.

6.7.3.9 bool bf::DecompositionAnalyzer::utest ()

Unit test

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

- DecompositionAnalyzer.h
- DecompositionAnalyzer.cpp

6.8 bf::DecompositionModifier Class Reference

Public Member Functions

- DecompositionModifier (Decomposition &dc)
- ∼DecompositionModifier ()
- void interfererNulling (const int Nmax, const bool autodetect, const int startch, const int endch)
- void interfererNulling (const int Nmax, const bool autodetect, const int startch, const int endch, const int Ndiscard)

6.8.1 Detailed Description

Set of RFI mitigation algorithms that are applied to the SVD/EVD/QR/... decomposed covariance data.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 bf::DecompositionModifier::DecompositionModifier (Decomposition & *dc*) `[inline]`

C'stor. Ties object to Decomposition class to be modified.

Parameters:

← ***dc*** Decomposition class to modify

6.8.2.2 bf::DecompositionModifier::∼DecompositionModifier () `[inline]`

D'stor

6.8.3 Member Function Documentation

6.8.3.1 void bf::DecompositionModifier::interfererNulling (const int *Nmax*, const bool *autodetect*, const int *startch*, const int *endch*, const int *Ndiscard*)

Classic nulling of dominant eigenvalues of the decomposition. Each channel is checked with DecompositionAnalyzer using MDL and 3sigma. The initial interference count estimate is Nrfi=max(MDL,3sig). If Nmax>0 this estimate is clipped to Nrfi=min(Nrfi,Nmax). Finally the Nrfi largest eigenvalues are nulled. Nulling is done using median replacement.

When original covariance data contains disconnected antennas, you must specify the total number of disconnected antennas with Ndiscard. Otherwise the MDL estimate on Nrfi will be wrong.

In pulsar and fast transient observations you should take care to null only channels that are not expected to contain the observable.

Parameters:

← ***Nmax*** Upper limit on detected interferers to null or <1 to null all

← ***autodetect*** True to estimate Nrfi, false to apply Nrfi=Nmax>0 directly.

← ***startch*** First channel where to start nulling

← ***endch*** Last channel to null (inclusive)

← ***Ndiscard*** Number of smallest eigenvalues to ignore in MDL and 3sigma estimation.

6.8.3.2 void bf::DecompositionModifier::interfererNulling (const int *Nmax*, const bool *autodetect*, const int *startch*, const int *endch*) `[inline]`

Classic nulling of dominant eigenvalues of the decomposition. Each channel is checked with DecompositionAnalyzer using MDL and 3sigma. The initial interference count estimate is Nrfi=max(MDL,3sig). If Nmax>0 this estimate is clipped to Nrfi=min(Nrfi,Nmax). Finally the Nrfi largest eigenvalues are nulled. Nulling is done using median replacement.

In pulsar and fast transient observations you should take care to null only channels that are not expected to contain the observable.

Parameters:

← ***Nmax*** Upper limit on detected interferers to null or <1 to null all

← ***autodetect*** True to estimate Nrfi, false to apply Nrfi=Nmax>0 directly.

← ***startch*** First channel where to start nulling

← ***endch*** Last channel to null (inclusive)

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

- DecompositionModifier.h
- DecompositionModifier.cpp

6.9 bf::ElementXYZ_tt Struct Reference

Data Fields

- int Nant

  Number of antenna positions stored.

- int Ldim [3]

  Max nr of antennas along the dimensions x,y,z. For example 3x3x3 if 27-antenna cube.

- arma::Col< bf::real > x

  X coordinates of the Nant antennas.

- arma::Col< bf::real > y

  Y coordinates of the Nant antennas.

- arma::Col< bf::real > z

  Z coordinates of the Nant antennas.

- arma::Col< int > flag

  Flags are OR'red ArrayElement::Pointing and ArrayElement::Polarization values.

6.9.1 Detailed Description

Phased array element positions, array dimensions and element properties

6.9.2 Field Documentation

6.9.2.1 arma::Col<int> bf::ElementXYZ_tt::flag

Flags are OR'red ArrayElement::Pointing and ArrayElement::Polarization values.

6.9.2.2 int bf::ElementXYZ_tt::Ldim[3]

Max nr of antennas along the dimensions x,y,z. For example 3x3x3 if 27-antenna cube.

6.9.2.3 int bf::ElementXYZ_tt::Nant

Number of antenna positions stored.

6.9.2.4 arma::Col<bf::real> bf::ElementXYZ_tt::x

X coordinates of the Nant antennas.

6.9.2.5 arma::Col<bf::real> bf::ElementXYZ_tt::y

Y coordinates of the Nant antennas.

6.9.2.6 arma::Col<bf::real> bf::ElementXYZ_tt::z

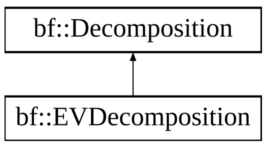
Z coordinates of the Nant antennas.

The documentation for this struct was generated from the following file:

- ArrayElements.h

6.10 bf::EVDecomposition Class Reference

Inheritance diagram for bf::EVDecomposition::



Public Types

- enum Type { None = -1, EVD = 0, SVD = 1, QR = 2 }

  Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Public Member Functions

- EVDecomposition (Covariance &Rxx)
- EVDecomposition (arma::Mat< bf::complex > &Rxx)
- ~EVDecomposition ()
- void set_M_smp (int M_smp_new)
- int decompose (Covariance const &cov)

- int decompose (Covariance const &cov, const int startch, const int endch)
- int decompose (arma::Mat< bf::complex > const &Rxx)
- int recompose (Covariance &cov)
- int recompose (Covariance &cov, const int startch, const int endch)
- int recompose (arma::Mat< bf::complex > &Rxx)

Data Fields

- const int N_ant
- const int N_chan
- int M_smp
- int _deco_type

Friends

- std::ostream & operator<< (std::ostream &, EVDecomposition const &)

6.10.1 Detailed Description

Derived class for computing eigen decompositions of a 3D data cube or single 2D matrix.

6.10.2 Member Enumeration Documentation

6.10.2.1 enum bf::Decomposition::Type `[inherited]`

Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Enumerator:

None

EVD

SVD

QR

6.10.3 Constructor & Destructor Documentation

6.10.3.1 bf::EVDecomposition::EVDecomposition (Covariance & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.10.3.2 bf::EVDecomposition::EVDecomposition (arma::Mat< bf::complex > & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.10.3.3 bf::EVDecomposition::∼EVDecomposition () `[inline]`

6.10.4 Member Function Documentation

6.10.4.1 int bf::Decomposition::decompose (arma::Mat< bf::complex > const & *Rxx*) `[inline, inherited]`

Perform single decomposition of given covariance matrix class.

Parameters:

← ***Rxx*** The covariance matrix to decompose.

Returns:

0 on success.

6.10.4.2 int bf::Decomposition::decompose (Covariance const & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch decomposition of a range of covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.
← ***startch*** Channel at which to start decomposition, 0 is first
← ***endch*** Last channel (inclusive) to decompose

Returns:

0 on success.

6.10.4.3 int bf::Decomposition::decompose (Covariance const & *cov*) `[inherited]`

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success.

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success

6.10.4.4 int bf::Decomposition::recompose (arma::Mat< bf::complex > & *Rxx*) `[inline, inherited]`

Recompute one covariance matrix based on the first (0-channel) decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***Rxx*** Output covariance matrix.

Returns:

0 on success

6.10.4.5 int bf::Decomposition::recompose (Covariance & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch recomposition of a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.
← ***startch*** Channel at which to start recomposition, 0 is first
← ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

Batch recompute a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.
← ***startch*** Channel at which to start recomposition, 0 is first
← ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

6.10.4.6 int bf::Decomposition::recompose (Covariance & *cov*) `[inherited]`

Perform batch recomposition of all covariance matrices based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.

Returns:

0 on success

6.10.4.7 void bf::Decomposition::set_M_smp (int *M_smp_new*) `[inline, inherited]`

Accessors

6.10.5 Friends And Related Function Documentation

6.10.5.1 std::ostream& operator<< (std::ostream &, EVDecomposition const &) `[friend]`

Reimplemented from bf::Decomposition.

6.10.6 Field Documentation

6.10.6.1 int bf::Decomposition::_deco_type `[inherited]`

6.10.6.2 int bf::Decomposition::M_smp `[inherited]`

6.10.6.3 const int bf::Decomposition::N_ant `[inherited]`

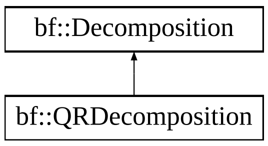
6.10.6.4 const int bf::Decomposition::N_chan `[inherited]`

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

- Decompositions.h
- Decompositions.cpp

6.11 bf::QRDecomposition Class Reference

Inheritance diagram for bf::QRDecomposition::



Public Types

- enum Type { None = -1, EVD = 0, SVD = 1, QR = 2 }

Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Public Member Functions

- QRDecomposition (Covariance &Rxx)
- QRDecomposition (arma::Mat< bf::complex > &Rxx)
- ∼QRDecomposition ()
- void set_M_smp (int M_smp_new)
- int decompose (Covariance const &cov)
- int decompose (Covariance const &cov, const int startch, const int endch)
- int decompose (arma::Mat< bf::complex > const &Rxx)
- int recompose (Covariance &cov)
- int recompose (Covariance &cov, const int startch, const int endch)
- int recompose (arma::Mat< bf::complex > &Rxx)

Data Fields

- const int N_ant
- const int N_chan
- int M_smp
- int _deco_type

Friends

- std::ostream & operator<< (std::ostream &, QRDecomposition const &)

6.11.1 Detailed Description

Derived class for computing QR matrix decompositions of a 3D data cube or single 2D matrix.

6.11.2 Member Enumeration Documentation

6.11.2.1 enum bf::Decomposition::Type `[inherited]`

Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Enumerator:

None

EVD

SVD

QR

6.11.3 Constructor & Destructor Documentation

6.11.3.1 bf::QRDecomposition::QRDecomposition (Covariance & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.11.3.2 bf::QRDecomposition::QRDecomposition (arma::Mat< bf::complex > & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.11.3.3 bf::QRDecomposition::∼QRDecomposition () `[inline]`

6.11.4 Member Function Documentation

6.11.4.1 int bf::Decomposition::decompose (arma::Mat< bf::complex > const & *Rxx*) `[inline, inherited]`

Perform single decomposition of given covariance matrix class.

Parameters:

← ***Rxx*** The covariance matrix to decompose.

Returns:

0 on success.

6.11.4.2 int bf::Decomposition::decompose (Covariance const & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch decomposition of a range of covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.
← ***startch*** Channel at which to start decomposition, 0 is first
← ***endch*** Last channel (inclusive) to decompose

Returns:

0 on success.

6.11.4.3 int bf::Decomposition::decompose (Covariance const & *cov*) `[inherited]`

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success.

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success

6.11.4.4 int bf::Decomposition::recompose (arma::Mat< bf::complex > & *Rxx*) `[inline, inherited]`

Recompute one covariance matrix based on the first (0-channel) decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***Rxx*** Output covariance matrix.

Returns:

0 on success

6.11.4.5 int bf::Decomposition::recompose (Covariance & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch recomposition of a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.

← ***startch*** Channel at which to start recomposition, 0 is first

← ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

Batch recompute a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.

← ***startch*** Channel at which to start recomposition, 0 is first

← ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

6.11.4.6 int bf::Decomposition::recompose (Covariance & *cov*) `[inherited]`

Perform batch recomposition of all covariance matrices based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***cov*** Output covariance class for the resulting matrices.

Returns:

0 on success

6.11.4.7 void bf::Decomposition::set_M_smp (int *M_smp_new*) `[inline, inherited]`

Accessors

6.11.5 Friends And Related Function Documentation

6.11.5.1 std::ostream& operator<< (std::ostream &, QRDecomposition const &) `[friend]`

Reimplemented from bf::Decomposition.

6.11.6 Field Documentation

6.11.6.1 int bf::Decomposition::_deco_type `[inherited]`

6.11.6.2 int bf::Decomposition::M_smp `[inherited]`

6.11.6.3 const int bf::Decomposition::N_ant `[inherited]`

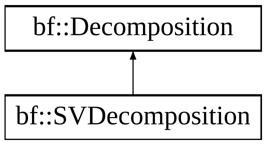
6.11.6.4 const int bf::Decomposition::N_chan `[inherited]`

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

- Decompositions.h
- Decompositions.cpp

6.12 bf::SVDecomposition Class Reference

Inheritance diagram for bf::SVDecomposition::



Public Types

- enum Type { None = -1, EVD = 0, SVD = 1, QR = 2 }

  Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Public Member Functions

- SVDecomposition (Covariance &Rxx)
- SVDecomposition (arma::Mat< bf::complex > &Rxx)
- ∼SVDecomposition ()
- void set_M_smp (int M_smp_new)
- int decompose (Covariance const &cov)
- int decompose (Covariance const &cov, const int startch, const int endch)
- int decompose (arma::Mat< bf::complex > const &Rxx)
- int recompose (Covariance &cov)
- int recompose (Covariance &cov, const int startch, const int endch)
- int recompose (arma::Mat< bf::complex > &Rxx)

Data Fields

- const int N_ant
- const int N_chan
- int M_smp
- int _deco_type

Friends

- std::ostream & operator<< (std::ostream &, SVDecomposition const &)

6.12.1 Detailed Description

Derived class for computing SVD matrix decompositions of a 3D data cube or single 2D matrix.

6.12.2 Member Enumeration Documentation

6.12.2.1 enum bf::Decomposition::Type `[inherited]`

Identifier for type of decomposition (eigenvalue, singular value or QR decomposition).

Enumerator:

None

EVD

SVD

QR

6.12.3 Constructor & Destructor Documentation

6.12.3.1 bf::SVDecomposition::SVDecomposition (Covariance & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.12.3.2 bf::SVDecomposition::SVDecomposition (arma::Mat< bf::complex > & *Rxx*) `[inline]`

C'stor. See parent class for documentation.

6.12.3.3 bf::SVDecomposition::∼SVDecomposition () `[inline]`

6.12.4 Member Function Documentation

6.12.4.1 int bf::Decomposition::decompose (arma::Mat< bf::complex > const & *Rxx*) `[inline, inherited]`

Perform single decomposition of given covariance matrix class.

Parameters:

← ***Rxx*** The covariance matrix to decompose.

Returns:

0 on success.

6.12.4.2 int bf::Decomposition::decompose (Covariance const & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch decomposition of a range of covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

← ***startch*** Channel at which to start decomposition, 0 is first

← ***endch*** Last channel (inclusive) to decompose

Returns:

0 on success.

6.12.4.3 int bf::Decomposition::decompose (Covariance const & *cov*) `[inherited]`

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success.

Perform batch decomposition of all covariances in the argument class.

Parameters:

← ***cov*** The covariance class with one or more matrices.

Returns:

0 on success

6.12.4.4 int bf::Decomposition::recompose (arma::Mat< bf::complex > & *Rxx*) `[inline, inherited]`

Recompute one covariance matrix based on the first (0-channel) decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

↔ ***Rxx*** Output covariance matrix.

Returns:

0 on success

6.12.4.5 int bf::Decomposition::recompose (Covariance & *cov*, const int *startch*, const int *endch*) `[inherited]`

Perform batch recomposition of a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

$\leftarrow$ ***startch*** Channel at which to start recomposition, 0 is first

$\leftarrow$ ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

Batch recompute a range of main covariance matrice(s) based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

$\leftarrow$ ***startch*** Channel at which to start recomposition, 0 is first

$\leftarrow$ ***endch*** Last channel (inclusive) to recompose

Returns:

0 on success

6.12.4.6 int bf::Decomposition::recompose (Covariance & *cov*) `[inherited]`

Perform batch recomposition of all covariance matrices based on the decomposition data stored internally in this object. Internal and output object data cube sizes must be identical.

Parameters:

$\leftrightarrow$ ***cov*** Output covariance class for the resulting matrices.

Returns:

0 on success

6.12.4.7 void bf::Decomposition::set_M_smp (int *M_smp_new*) `[inline, inherited]`

Accessors

6.12.5 Friends And Related Function Documentation

6.12.5.1 std::ostream& operator<< (std::ostream &, SVDecomposition const &) `[friend]`

Reimplemented from bf::Decomposition.

6.12.6 Field Documentation

6.12.6.1 int bf::Decomposition::_deco_type `[inherited]`

6.12.6.2 int bf::Decomposition::M_smp `[inherited]`

6.12.6.3 const int bf::Decomposition::N_ant `[inherited]`

6.12.6.4 const int bf::Decomposition::N_chan `[inherited]`

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

- Decompositions.h
- Decompositions.cpp

7 File Documentation

7.1 ArrayElements.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, ArrayElements const &a)

7.2 ArrayElements.h File Reference

Data Structures

- struct bf::ElementXYZ_tt
- class bf::ArrayElements

Namespaces

- namespace bf

Typedefs

- typedef struct bf::ElementXYZ_tt bf::ElementXYZ_t

Functions

- std::ostream & bf::operator<< (std::ostream &os, ArrayElements const &a)

7.3 Beamformer.h File Reference

7.4 BeamformerData.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, Beams_t const &b)

7.5 BeamformerData.h File Reference

Data Structures

- struct bf::Beams_tt

Namespaces

- namespace bf

Typedefs

- typedef struct bf::Beams_tt bf::Beams_t

Functions

- std::ostream & bf::operator<< (std::ostream &os, Beams_t const &b)

7.6 BeamformerTypeDefs.h File Reference

Namespaces

- namespace bf

Typedefs

- typedef arma::cx_double bf::complex
- typedef double bf::real

7.7 BeamformerWeights.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, BeamformerWeights const &w)

7.8 BeamformerWeights.h File Reference

Data Structures

- class bf::BeamformerWeights

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, BeamformerWeights const &w)

7.9 Covariance.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, Covariance const &c)

7.10 Covariance.h File Reference

Data Structures

- class bf::Covariance

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, Covariance const &c)

7.11 CovarianceModifier.cpp File Reference

Namespaces

- namespace bf

7.12 CovarianceModifier.h File Reference

Data Structures

- class bf::CovarianceModifier

Namespaces

- namespace bf

7.13 Decomposition.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, Decomposition const &d)

7.14 Decomposition.h File Reference

Data Structures

- class bf::Decomposition

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, Decomposition const &d)

7.15 DecompositionAnalyzer.cpp File Reference

Namespaces

- namespace bf

7.15.1 Detailed Description

Class for analyzing array covariance matrix decompositions.

The rank of the signal (RFI) subspace i.e. the unknown number of interferers is estimated using some information criterion. The same AIC, MDL work on the decomposition of time-integrated covariance data from both time domain as well as Fourier domain.

[1] M. Wax, T. Kailath, "Detection of Signals by Information Theoretic Criteria", IEEE, Vol. ASSP-33, No. 2, April 1985, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1164557

7.16 DecompositionAnalyzer.h File Reference

Data Structures

- class bf::DecompositionAnalyzer

Namespaces

- namespace bf

7.17 DecompositionModifier.cpp File Reference

Namespaces

- namespace bf

7.18 DecompositionModifier.h File Reference

Data Structures

- class bf::DecompositionModifier

Namespaces

- namespace bf

7.19 Decompositions.cpp File Reference

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, QRDecomposition const &d)
- std::ostream & bf::operator<< (std::ostream &os, EVDecomposition const &d)
- std::ostream & bf::operator<< (std::ostream &os, SVDecomposition const &d)

7.20 Decompositions.h File Reference

Data Structures

- class bf::QRDecomposition
- class bf::EVDecomposition
- class bf::SVDecomposition

Namespaces

- namespace bf

Functions

- std::ostream & bf::operator<< (std::ostream &os, QRDecomposition const &d)
- std::ostream & bf::operator<< (std::ostream &os, EVDecomposition const &d)
- std::ostream & bf::operator<< (std::ostream &os, SVDecomposition const &d)

Index