RIT MKS Fault Detection Library

0.2.1

Generated by Doxygen 1.8.10

Tue Jul 14 2015 15:34:23

Contents

Chapter 1

RIT MKS Fault Detection Library

This C++ library was developed by the Multi-Agent BioRobotics Laboratory (MABL) at Rochester Institute of Technology (RIT) in collaboration with MKS Instruments and the MKS ENI Products divison.

This library is an implementation of a fault detection library for RF Power Generators. The library provides functionality of a one-class classifier that uses Mixture of Gaussians to identify time-series system signatures as normal or faulty. The code was devloped in compliance with MKS embedded requirements. No external thrid-party libraries were used and C++ 98 compiler standard were used. All required matrix operations and other non-fundamental mathematical operations/algorithms have been implemented.

Through the use of pre-processor definitions the library may be used within linux and windows based operating systems. Based on computational power of the target platform, matrix precision may be represented as float or double precision. Focus of the documentation is for a target platform that is linux-based.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Dr. Ferat Sahin (feseee@rit.edu)

Date

2013-2015

Chapter 2

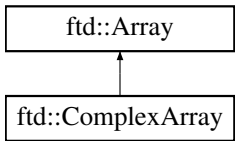
Class Documentation

2.1 ftd::Array Class Reference

Container object for array of real values.

`#include <Array.h>`

Inheritance diagram for ftd::Array:



Public Member Functions

- Array (int size)

  Constructor, Empty Array object with specified size.
- Array (int size, data_type *data)

  Constructor, Set values.
- Array (const Array &c)

  Copy Constructor.
- virtual ~Array ()

  Destructor.
- data_type at (int index) const

  Retrieve value in array at specified index.
- int getSize (void) const

  Return the number of elements in the array.

Protected Member Functions

- Array ()

  Default Constructor, Empty array, no size.

Protected Attributes

- data_type * data

  Pointer to storage of real values.
- int size

  Number of elements in the Array object.

Friends

- class ComplexArray

  Friend class ComplexArray.

2.1.1 Detailed Description

Container object for array of real values.

Container object to hold contents of an array of real numbers. Intended for use of insertion and fetch from Matrix objects.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Array::Array (int *size*)

Constructor, Empty Array object with specified size.

Parameters

size	Desired number of elements in array object.

2.1.2.2 Array::Array (int *size*, data_type * *data*)

Constructor, Set values.

Parameters

size	Number of elements to place in array.
data	Values to store in the array, must be length of size.

Warning

Copies values pointed to, does not free data. Memory management must be done outside scope of constructor call.

2.1.2.3 Array::Array (const Array & *c*)

Copy Constructor.

Parameters

c	Array to be copied.

2.1.3 Member Function Documentation

2.1.3.1 data_type ftd::Array::at (int *index*) const [inline]

Retrieve value in array at specified index.

Parameters

index	Index of the item to be retrived (0 based).

Returns

Value at specified index

2.1.3.2 int Array::getSize (void) const

Return the number of elements in the array.

Returns

number of elements in the array.

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

- Array.h
- Array.cpp

2.2 ftd::Complex Struct Reference

Structure for complex number representation.

`#include <Array.h>`

Public Member Functions

- Complex (data_type real, data_type img)
  Constructor.
- Complex operator* (Complex c)
  Multiplication operation overloaded.

Public Attributes

- data_type img
  Imaginary component of complex number.
- data_type real
  Real component of complex number.

2.2.1 Detailed Description

Structure for complex number representation.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 ftd::Complex::Complex (data_type *real*, data_type *img*) [inline]

Constructor.

Parameters

real	Real value of complex number.
img	Imaginary value of complex number.

2.2.3 Member Function Documentation

2.2.3.1 Complex ftd::Complex::operator* (Complex *c*) [inline]

Multiplication operation overloaded.

Implement complex multiplication of two complex numbers.

Parameters

c	Structure of complex number to multiply existing complex number with.

Returns

Copy of the result of the complex multiplication.

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

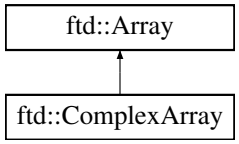
- Array.h

2.3 ftd::ComplexArray Class Reference

Container object for array of real or complex values.

`#include <Array.h>`

Inheritance diagram for ftd::ComplexArray:



Public Member Functions

- ComplexArray (int size)

Constructor, Empty container with specified size.

- ComplexArray (int size, data_type *real, data_type *img)

  Constructor, Set values.
- ComplexArray (int size, data_type *real)

  Constructor, copy real values from memory specified by pointer.
- ComplexArray (const ComplexArray &c)
- ComplexArray (const Array &arr)

  Constructor, copy real values from existing Array.
- virtual ~ComplexArray ()

  Destructor.
- data_type at (int index) const

  Retrieve value in array at specified index.
- int getSize (void) const

  Return the number of elements in the array.
- ComplexArray & operator= (const ComplexArray &c)

  Assignment operator overloaded.
- void padZeros (int n)

  Pads array with zeros at end.
- void trunc (int n)

  Truncate array to specified size.

Protected Attributes

- data_type * data

  Pointer to storage of real values.
- int size

  Number of elements in the Array object.

Friends

- class ComplexMatrix

  Friend class ComplexMatrix.
- void fft (ComplexArray &c)

  Friend function fft, performs FFT on ComplexArray.
- ComplexArray fft_pad (const ComplexArray &)

  Friend function fft_pad performs FFT on ComplexArray; padding first.

2.3.1 Detailed Description

Container object for array of real or complex values.

Container object to hold contents of an array of real or complex numbers. Intended for use of insertion and fetch from Matrix objects. Also, created for ease for implementation of FFT.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 ComplexArray::ComplexArray (int *size*)

Constructor, Empty container with specified size.

Parameters

size	Number of elements to initialize.

2.3.2.2 ComplexArray::ComplexArray (int *size*, data_type * *real*, data_type * *img*)

Constructor, Set values.

Copies real and imaginary values from locations specified by input pointers.

Parameters

size	Number of complex elements to place in array.
real	Pointer to memory where real values are stored.
img	Pointer to memory where complex values are stored.

Warning

Copies values pointed to, does not free data. Memory management must be done outside scope of constructor call.

2.3.2.3 ComplexArray::ComplexArray (int *size*, data_type * *real*)

Constructor, copy real values from memory specified by pointer.

Real values are copied into new ComplexArray, and imaginary values are initialized to 0.

Parameters

size	Number of real values in memory to copy.
real	Pointer to memory where real values are stored.

Warning

Copies values pointed to, does not free data. Memory management must be done outside scope of constructor call.

2.3.2.4 ComplexArray::ComplexArray (const ComplexArray & *c*)

Copy Constructor

Parameters

c	ComplexArray to be copied.

2.3.2.5 ComplexArray::ComplexArray (const Array & *arr*)

Constructor, copy real values from existing Array.

Real values are copied into new ComplexArray, and imaginary values are initialized to 0.

Parameters

arr	Array of real values to copy into new ComplexArray.

2.3.3 Member Function Documentation

2.3.3.1 data_type ftd::Array::at (int *index*) const [inline], [inherited]

Retrieve value in array at specified index.

Parameters

index	Index of the item to be retrived (0 based).

Returns

Value at specified index

2.3.3.2 int Array::getSize (void) const [inherited]

Return the number of elements in the array.

Returns

number of elements in the array.

2.3.3.3 ComplexArray & ComplexArray::operator= (const ComplexArray & *c*)

Assignment operator overloaded.

Parameters

c	Reference to ComplexArray object assign current ComplexArray object.

Returns

Reference to newly assigned ComplexArray object.

2.3.3.4 void ComplexArray::padZeros (int *n*)

Pads array with zeros at end.

Parameters

n	Number of zeros to pad at end of array.

2.3.3.5 void ComplexArray::trunc (int *n*)

Truncate array to specified size.

Parameters

n	Final size of the array.

Warning

n must be $<$ size of array.

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

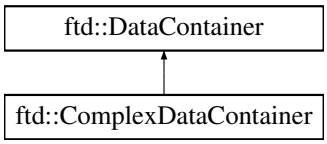
- Array.h
- Array.cpp

2.4 ftd::ComplexDataContainer Class Reference

Container Object for data file with complex values.

`#include <DataContainer.h>`

Inheritance diagram for ftd::ComplexDataContainer:



Public Member Functions

- ComplexDataContainer (int nVars, int nEntries)

  Constructor, creates empty data container.
- ComplexDataContainer (const ComplexDataContainer &m)

  Copy Constructor with complex values.
- ComplexDataContainer (const DataContainer &m)

  Copy Constructor with real values.
- virtual ~ComplexDataContainer ()

  Destructor.
- const data_type * getData (void) const

  Copies the existing data in the DataContainer into a new memory location then returns the pointer to the memory location.
- data_type getData (int row, int col) const

  Retrieve a specific value from the DataContainer.
- const data_type * getImgData (void) const

  Returns pointer to imaginary data.
- Matrix getMatrix (void)

  Returns Matrix representation of DaraContainer.
- ComplexMatrix getMatrix (void) const

  Returns a copy of the complex data container in a ComplexMatrix object.
- int getNumberEntries (void) const

  Returns the number of entries in the container.

- int getNumberVars (void) const

  Returns the number of variable in the container.
- const data_type * getRealData (void) const

  Returns pointer to real values of the container.
- void insertEntry (const data_type *values, int row)

  Inserts an entire entry into the container.
- void insertItem (data_type value, int row, int col)

  Inserts a value into specific location in the container.
- void insertItem (data_type real, data_type img, int row, int col)

  Insert a single item into the complex data container.
- ComplexDataContainer & operator= (const ComplexDataContainer &dc)

  Overloaded Assignment operator.
- virtual void print (FILE *fOut=stderr) const

  Prints the complex data container to a file.
- void printCLI (void) const

  Function that prints the data container out to the command line.
- void setData (const data_type *values, int nVars, int nEntries)

  Sets the data of the DataContainer to new values.

Protected Attributes

- data_type * data

  Storage of the data for the DataContainer.
- int nEntries

  Number of entries (columns) in the DataContainer.
- int nVars

  Number variables (rows) in the DataContainer.

2.4.1 Detailed Description

Container Object for data file with complex values.

Container Object for a data file with complex values only. Access is similar to a 2D structure with entries(row) and variables (rows). However fundatmental implementation uses flat struture for memory access optimization.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 ComplexDataContainer::ComplexDataContainer (int *nVars*, int *nEntries*)

Constructor, creates empty data container.

Parameters

nVars	Number of variables (ie columns) in complex data container.
nEntries	Number of entries (ie rows) in complex data container.

2.4.2.2 ComplexDataContainer::ComplexDataContainer (const ComplexDataContainer & *m*)

Copy Constructor with complex values.

Parameters

m	Existing complex data container to be copied.

2.4.2.3 ComplexDataContainer::ComplexDataContainer (const DataContainer & *m*)

Copy Constructor with real values.

Parameters

m	Existing non-complex data container to be copied into a complex container.

2.4.3 Member Function Documentation

2.4.3.1 const data_type * DataContainer::getData (void) const [inherited]

Copies the existing data in the DataContainer into a new memory location then returns the pointer to the memory location.

Warning

DOES NOT CLEAN UP MEMORY, MUST BE DONE OUTSIDE OF THIS FUNCTION SCOPE

Returns

Pointer to memory location of copy of data in the DataContainer.

2.4.3.2 data_type DataContainer::getData (int *row*, int *col*) const [inherited]

Retrieve a specific value from the DataContainer.

Parameters

row	Row (entry location) of the data to retrieve.
col	Column (variable location) of the data to retrieve.

Returns

Data value corresponding to (row,col) ie (entry,variable)

2.4.3.3 const data_type * ComplexDataContainer::getImgData (void) const

Returns pointer to imaginary data.

Returns

Pointer to imaginary values of the container.

2.4.3.4 Matrix DataContainer::getMatrix (void) [inherited]

Returns Matrix representation of DaraContainer.

Returns

DataContainer object as a Matrix object.

2.4.3.5 ComplexMatrix ComplexDataContainer::getMatrix (void) const

Returns a copy of the complex data container in a ComplexMatrix object.

Returns

Copy of the complex data container in a ComplexMatrix object.

2.4.3.6 int DataContainer::getNumberEntries (void) const [inherited]

Returns the number of entries in the container.

Returns

Number of data entries in the container.

2.4.3.7 int DataContainer::getNumberVars (void) const [inherited]

Returns the number of variable in the container.

Returns

Number of variable in the container.

2.4.3.8 const data_type * ComplexDataContainer::getRealData (void) const

Returns pointer to real values of the container.

Returns

Pointer to real values of the container.

2.4.3.9 void DataContainer::insertEntry (const data_type * *values*, int *row*) [inherited]

Inserts an entire entry into the container.

Inserts an entire entry into the container, replacing existing values. If the row location $>$ number entries the DataContainer is resized and padded with zeros.

Parameters

values	Pointer to memory location storing values to be inserted.
row	Row (entry location) to insert the entry values.

Warning

There is no check on the number of items passed to be.

2.4.3.10 void DataContainer::insertItem (data_type *value*, int *row*, int *col*) [inherited]

Inserts a value into specific location in the container.

Inserts a value into specific location in the conntainer object. If row location is outside the current number of entries, the DataContainer is resized, padded with zeros.

Parameters

value	Real value to be inserted.
row	Row (entry location) to place value.
col	Column (variable location) to place value.

Warning

The column location must not exceed the number of variables.

2.4.3.11 void ComplexDataContainer::insertItem (data_type *real*, data_type *img*, int *row*, int *col*)

Insert a single item into the complex data container.

Parameters

real	Real component of the item to be inserted.
img	Imaginary component of the item to be inserted.
row	Row (entry location) to place value.
col	Column (variable location) to place value.

2.4.3.12 ComplexDataContainer & ComplexDataContainer::operator= (const ComplexDataContainer & *dc*)

Overloaded Assignment operator.

Parameters

dc	Reference to ComplexDataContainer object to assign this ComplexDataContainer to.

Returns

Reference to this ComplexDataConatiner object after assignment.

2.4.3.13 void ComplexDataContainer::print (FILE * *fOut* = stderr) const [virtual]

Prints the complex data container to a file.

Parameters

fOut	File object pointer to print complex data container. Default is stderr if argument is not specified.

Reimplemented from ftd::DataContainer.

2.4.3.14 void DataContainer::setData (const data_type * *values*, int *nVars*, int *nEntries*) [inherited]

Sets the data of the DataContainer to new values.

Parameters

values	Pointer to memory location of new data.
nVars	Number of variable (ie columns)
nEntries	Number of rows (ie rows)

Warning

ALL OLD VALUES ARE DESTROYED, NEW MEMORY ALLOCATED AND SET.

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

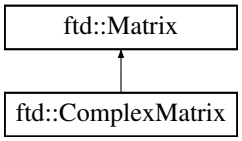
- DataContainer.h
- DataContainer.cpp

2.5 ftd::ComplexMatrix Class Reference

Container for matrix of complex values.

`#include <Matrix.h>`

Inheritance diagram for ftd::ComplexMatrix:



Public Member Functions

- ComplexMatrix (int r, int c)

  Constructor, initialize all real/imaginary values to zero.
- ComplexMatrix (int r, int c, data_type *real)

  Constructor, initialize real values as specified, imaginary initialized to 0.
- ComplexMatrix (int r, int c, data_type *real, data_type *imag)

  Constructor, initialize real and imaginary values as specified.
- ComplexMatrix (const ComplexMatrix &cm)

  Copy Constructor.
- ComplexMatrix (const Matrix &m)

  Copy Constructor for non-complex Matrix.

- ~ComplexMatrix ()

  Destructor.
- ComplexArray getCol (int c) const

  Return a column from the ComplexMatrix.
- data_type getData (int r, int c) const

  Get a value from a Matrix.
- int getNCols () const

  Return the number of columns in the Matrix.
- int getNRows () const

  Return the number of rows in the Matrix.
- ComplexArray getRow (int r) const

  Return a row from the ComplexMatrix.
- Matrix operator* (const Matrix &m) const

  Overloaded operator (*), performs matrix multiplication.
- Matrix operator* (data_type d) const

  Overloaded operator (*), Multiplies all values of Matrix by a single value.
- Matrix & operator*= (const Matrix &m)

  Overloaded operator (=), performs matrix multiplication and assigns to LHS.*
- Matrix & operator*= (data_type d)

  Overloaded operator (=), Multiplies all values of Matrix by a single value, assigns to LHS.*
- Matrix operator+ (const Matrix &m) const

  Overloaded operator (+), Add element-wise two matrices.
- Matrix operator+ (data_type d) const

  Overloaded operator (+), Add each value in Matrix by single value.
- Matrix & operator+= (const Matrix &m)

  Overloaded operator (+=), Add element-wise two matrices assign result to LHS.
- Matrix & operator+= (data_type d)

  Overloaded operator (+=), Add each value in Matrix by single value and assign to LHS.
- Matrix operator- (const Matrix &m) const

  Overloaded operator (-), Subtract element-wise two matrices.
- Matrix operator- (data_type d) const

  Overloaded operator (-), Subtract each value in Matrix by single value.
- Matrix & operator-= (const Matrix &m)

  Overloaded operator (-=), Subtract element-wise two matrices assign result to LHS.
- Matrix & operator-= (data_type d)

  Overloaded operator (-=), Subtract each value in Matrix by single value and assign to LHS.
- Matrix operator/ (data_type d) const

  Overloaded operator (/), Divided all values of Matrix by a single value.
- Matrix & operator/= (data_type d)

  Overloaded operator (/=), Divides all values of Matrix by a single value, assigns to LHS.
- const ComplexMatrix & operator= (const ComplexMatrix &cm)

  Overloaded assignment operator (=)
- const ComplexMatrix & operator= (const Matrix &m)

  Overloaded assignment operator (=), real values assigned, imaginary set to 0.
- virtual void print (FILE *fOut=stderr) const

  Print the values into a FILE object.
- void setCol (int c, const ComplexArray &ca)

Set a column in the ComplexMatrix.

- void setData (data_type d, int r, int c)

  Set a single value within a Matrix specified by row, column.
- void setData (data_type d, int ind)

  Set a single value within a Matrix specified by raw index.
- void setData (data_type *values)

  Set all values of the Matrix.
- void setRow (int r, const ComplexArray &ca)

  Set a row in the ComplexMatrix.
- Array toArray (void) const

  Return values from the Matrix in a single Array object.
- void trunc (int r, int c)

  Truncate Matrix to specified number of rows/columns.

Protected Attributes

- int cols

  Number of columns in Matrix.
- data_type * data

  Pointer to data for real values of the Matrix.
- data_type * imgData

  Pointer to imaginary values.
- int rows

  Number of rows in Matrix.

Friends

- Matrix absm (ComplexMatrix &cm)

  Compute the absolute value of the elements in the ComplexMatrix.
- ComplexMatrix dft2 (const ComplexMatrix &m)

  Computes N-point DFT for of the ComplexMatrix.
- ComplexMatrix fft1_col (const ComplexMatrix &cm)

  Computes N-point FFT for columns of ComplexMatrix.
- ComplexMatrix fft1_row (const ComplexMatrix &cm)

  Computes N-point FFT for rows of ComplexMatrix.
- ComplexMatrix fft2 (const ComplexMatrix &cm)

  Compute the 2D N-point FFT.

2.5.1 Detailed Description

Container for matrix of complex values.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 ComplexMatrix::ComplexMatrix (int *r*, int *c*)

Constructor, initialize all real/imaginary values to zero.

Parameters

r	Number of rows.
c	Number of columns.

2.5.2.2 ComplexMatrix::ComplexMatrix (int *r,* int *c,* data_type * *real*)

Constructor, initialize real values as specified, imaginary initialized to 0.

Parameters

r	Number of rows.
c	Number of columns.
real	Pointer to real values to copy into new ComplexMatrix.

Warning

The real values must match the number of elements specified by r and c. Memory management of input real values must be done outside scope of the constructor call.

2.5.2.3 ComplexMatrix::ComplexMatrix (int *r,* int *c,* data_type * *real,* data_type * *imag*)

Constructor, initialize real and imaginary values as specified.

Parameters

r	Number of rows.
c	Number of columns.
real	Pointer to real values to copy into new ComplexMatrix.
imag	Pointer to imaginary values to copy into new ComplexMatrix.

Warning

The real/imaginary values must match the number of elements specified by r and c. Memory management of input real/imaginary values must be done outside scope of the constructor call.

2.5.2.4 ComplexMatrix::ComplexMatrix (const ComplexMatrix & *cm*)

Copy Constructor.

Parameters

cm	ComplexMatrix to be copied.

2.5.2.5 ComplexMatrix::ComplexMatrix (const Matrix & *m*)

Copy Constructor for non-complex Matrix.

Real values are copied and imaginary are set to zero.

Parameters

m	Real valued Matrix to be copied

2.5.3 Member Function Documentation

2.5.3.1 ComplexArray ComplexMatrix::getCol (int *c*) const

Return a column from the ComplexMatrix.

Parameters

c	Column to return.

Returns

ComplexArray of value from specified column.

2.5.3.2 data_type Matrix::getData (int *r*, int *c*) const [inherited]

Get a value from a Matrix.

Parameters

r	Row of element.
c	Column of element.

Returns

Value of item at specified row and column.

2.5.3.3 ComplexArray ComplexMatrix::getRow (int *r*) const

Return a row from the ComplexMatrix.

Parameters

r	Row to return.

Returns

ComplexArray of value from specified row.

2.5.3.4 Matrix Matrix::operator∗ (const Matrix & *m*) const [inherited]

Overloaded operator (∗), performs matrix multiplication.

Matrix multiplication resulting in matrix of size [LHS_rows x RHS_columns].

Parameters

m	RHS Matrix for multiplication.

Returns

Copy of Matrix with values from the Matrix multiplication.

Warning

Proper Matrix sizes must be provided LHS_cols==RHS_rows!

2.5.3.5 Matrix Matrix::operator∗ (data_type *d*) const [inherited]

Overloaded operator (∗), Multiplies all values of Matrix by a single value.

Parameters

d	Value to multiply all Matrix values with.

Returns

Copy of Matrix resulting from the scalar multiplication.

2.5.3.6 Matrix & Matrix::operator∗= (const Matrix & *m*) [inherited]

Overloaded operator (∗=), performs matrix multiplication and assigns to LHS.

Matrix multiplication resulting in matrix of size [LHS_rows x RHS_columns].

Parameters

m	RHS Matrix for multiplication.

Returns

Reference to Matrix with values from the Matrix multiplication.

Warning

Proper Matrix sizes must be provided LHS_cols==RHS_rows!

2.5.3.7 Matrix & Matrix::operator∗= (data_type *d*) [inherited]

Overloaded operator (∗=), Multiplies all values of Matrix by a single value, assigns to LHS.

Parameters

d	Value to multiply all Matrix values with.

Returns

Reference of Matrix resulting from the scalar multiplication.

2.5.3.8 Matrix Matrix::operator+ (const Matrix & *m*) const [inherited]

Overloaded operator (+), Add element-wise two matrices.

Parameters

m	RHS of the addition.

Returns

Copy of the result of the matrix addition.

Warning

RHS and LHS must have the same number of elements.

2.5.3.9 Matrix Matrix::operator+ (data_type *d*) const [inherited]

Overloaded operator (+), Add each value in Matrix by single value.

Parameters

d	Single value to addition from all values.

Returns

Copy of the resulting scalar addition.

2.5.3.10 Matrix & Matrix::operator+= (const Matrix & *m*) [inherited]

Overloaded operator (+=), Add element-wise two matrices assign result to LHS.

Parameters

m	RHS of the addition.

Returns

Reference to the result of the matrix addition.

Warning

RHS and LHS must have the same number of elements.

2.5.3.11 Matrix & Matrix::operator+= (data_type *d*) [inherited]

Overloaded operator (+=), Add each value in Matrix by single value and assign to LHS.

Parameters

d	Single value to addition from all values.

Returns

Reference to the resulting scalar addition.

2.5.3.12 Matrix Matrix::operator- (const Matrix & *m*) const [inherited]

Overloaded operator (-), Subtract element-wise two matrices.

Parameters

m	RHS of the subtraction.

Returns

Copy of the result of the matrix subtraction.

Warning

RHS and LHS must have the same number of elements.

2.5.3.13 Matrix Matrix::operator- (data_type *d*) const [inherited]

Overloaded operator (-), Subtract each value in Matrix by single value.

Parameters

d	Single value to subtract from all values.

Returns

Copy of the resulting scalar subtraction.

2.5.3.14 Matrix & Matrix::operator-= (const Matrix & *m*) [inherited]

Overloaded operator (-=), Subtract element-wise two matrices assign result to LHS.

Parameters

m	RHS of the subtraction.

Returns

Reference to the result of the matrix subtraction.

Warning

RHS and LHS must have the same number of elements.

2.5.3.15 Matrix & Matrix::operator-= (data_type *d*) [inherited]

Overloaded operator (-=), Subtract each value in Matrix by single value and assign to LHS.

Parameters

d	Single value to subtract from all values.

Returns

Reference to the resulting scalar subtraction.

2.5.3.16 Matrix Matrix::operator/ (data_type *d*) const [inherited]

Overloaded operator (/), Divided all values of Matrix by a single value.

Parameters

d	Value to divide all Matrix values with.

Returns

Copy of Matrix resulting from the scalar division.

2.5.3.17 Matrix & Matrix::operator/= (data_type *d*) [inherited]

Overloaded operator (/=), Divides all values of Matrix by a single value, assigns to LHS.

Parameters

d	Value to divide all Matrix values with.

Returns

Reference of Matrix resulting from the scalar division.

2.5.3.18 const ComplexMatrix & ComplexMatrix::operator= (const ComplexMatrix & *cm*)

Overloaded assignment operator (=)

Parameters

cm	RHS of assignment with type ComplexMatrix

2.5.3.19 const ComplexMatrix & ComplexMatrix::operator= (const Matrix & *m*)

Overloaded assignment operator (=), real values assigned, imaginary set to 0.

Parameters

m	RHS of assignment with type Matrix

2.5.3.20 void ComplexMatrix::print (FILE * *fOut* = stderr) const [virtual]

Print the values into a FILE object.

Parameters

fOut	Pointer to FILE object to print values, default is stderr.

Warning

FILE pointer fOut must be valid and opened for write then closed outside scope.

Reimplemented from ftd::Matrix.

2.5.3.21 void ComplexMatrix::setCol (int *c*, const ComplexArray & *ca*)

Set a column in the ComplexMatrix.

Parameters

r	Column to be set.
ca	ComplexArray of values to set in column.

2.5.3.22 void Matrix::setData (data_type *d*, int *r*, int *c*) [inherited]

Set a single value within a Matrix specified by row, column.

Parameters

d	Value to be set.
r	Row of the value.
c	Column of the value.

2.5.3.23 void Matrix::setData (data_type *d*, int *ind*) [inherited]

Set a single value within a Matrix specified by raw index.

Parameters

d	Value to be set.
ind	Raw index to store value in Matrix.

2.5.3.24 void Matrix::setData (data_type * *values*) [inherited]

Set all values of the Matrix.

Parameters

values	Pointer to values to set in Matrix.

Warning

Input values must be created and initialized outside scope and number of elements must match that of the Matrix.

2.5.3.25 void ComplexMatrix::setRow (int *r,* const ComplexArray & *ca*)

Set a row in the ComplexMatrix.

Parameters

r	Row to be set.
ca	ComplexArray of values to set in row.

2.5.3.26 Array Matrix::toArray (void) const [inherited]

Return values from the Matrix in a single Array object.

Returns

Array with all Matrix values, number elements are rows∗cols.

2.5.3.27 void Matrix::trunc (int *r*, int *c*) [inherited]

Truncate Matrix to specified number of rows/columns.

The truncation process keeps the first r rows and c columns.

Parameters

r	Number of rows to keep.
c	Number of columns to keep.

Warning

The number of rows/columns desired must be less than current counts.

2.5.4 Friends And Related Function Documentation

2.5.4.1 Matrix absm (ComplexMatrix & *cm*) [friend]

Compute the absolute value of the elements in the ComplexMatrix.

Parameters

cm	ComplexMatrix to compute absolute value of the elements.

Returns

Real-valued Matrix of the absolute values.

2.5.4.2 ComplexMatrix dft2 (const ComplexMatrix & *m*) [friend]

Computes N-point DFT for of the ComplexMatrix.

The N-point DFT is taken column-wise then row-wise for 2D implementation.

Parameters

cm	ComplexMatrix to compute 2D DFT.

Returns

Result of the 2D N-Point DFT.

2.5.4.3 ComplexMatrix fft1_col (const ComplexMatrix & *cm*) [friend]

Computes N-point FFT for columns of ComplexMatrix.

N-point FFT using the Cooley-Tukey FFT. N is set as the next power of 2 and zero padding performed.

Parameters

cm	ComplexMatrix to compute FFT.

Returns

Result of the column-wise N-point FFT.

2.5.4.4 ComplexMatrix fft1_row (const ComplexMatrix & *cm*) [friend]

Computes N-point FFT for rows of ComplexMatrix.

N-point FFT using the Cooley-Tukey FFT. N is set as the next power of 2 and zero padding performed.

Parameters

cm	ComplexMatrix to compute FFT.

Returns

Result of the row-wise N-point FFT.

2.5.4.5 ComplexMatrix fft2 (const ComplexMatrix & *cm*) [friend]

Compute the 2D N-point FFT.

The N-point FFT is taken column-wise then row-wise for 2D implementation.

Parameters

cm	ComplexMatrix to compute 2D FFT.

Returns

Result of the 2D N-Point FFT.

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

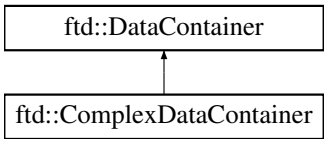
- Matrix.h
- Matrix.cpp

2.6 ftd::DataContainer Class Reference

Container class for real-values data files.

`#include <DataContainer.h>`

Inheritance diagram for ftd::DataContainer:



Public Member Functions

- DataContainer (int nVars, int nEntries)

  Constructor, creates empty data container.
- DataContainer (const DataContainer &dc)

  Copy Constructor.
- DataContainer (data_type *data, int nVars, int nEntries)

  Constructor given existing data array.
- virtual ~DataContainer ()

  Destructor.
- const data_type * getData (void) const

  Copies the existing data in the DataContainer into a new memory location then returns the pointer to the memory location.
- data_type getData (int row, int col) const

  Retrieve a specific value from the DataContainer.
- Matrix getMatrix (void)

  Returns Matrix representation of DaraContainer.
- int getNumberEntries (void) const

  Returns the number of entries in the container.
- int getNumberVars (void) const

  Returns the number of variable in the container.
- void insertEntry (const data_type *values, int row)

  Inserts an entire entry into the container.
- void insertItem (data_type value, int row, int col)

  Inserts a value into specific location in the container.
- DataContainer & operator= (const DataContainer &dc)

  Overloaded Assigment operator.
- virtual void print (FILE *fOut=stderr) const

  Prints the data container to a file.
- void printCLI (void) const

  Function that prints the data container out to the command line.
- void setData (const data_type *values, int nVars, int nEntries)

  Sets the data of the DataContainer to new values.

Protected Member Functions

- DataContainer ()

  Default Constructor - Initialize vars, entries and data to 0.
- void resizeCopyOld (int row)

  Helper function to resize the DataContiner.

Protected Attributes

- data_type * data

  Storage of the data for the DataContainer.
- int nEntries

  Number of entries (columns) in the DataContainer.
- int nVars

  Number variables (rows) in the DataContainer.

2.6.1 Detailed Description

Container class for real-values data files.

Data Container Object for a data file with real values only. Access is similar to a 2D structure with entries(row) and variables (rows). However fundamental implementation uses flat structure for memory access optimization.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 DataContainer::DataContainer (int *nVars*, int *nEntries*)

Constructor, creates empty data container.

Parameters

nVars	Number of variables (ie columns)
nEntries	Number of data entries (ie rows)

2.6.2.2 DataContainer::DataContainer (const DataContainer & *dc*)

Copy Constructor.

Parameters

dc	Refernce to DataContainer object to be copied.

2.6.2.3 DataContainer::DataContainer (data_type * *data*, int *nVars*, int *nEntries*)

Constructor given existing data array.

Parameters

data	Data to be stored in the container.
nVars	Number of variables (ie columns).
nEntries	Number of data entries (ie rows).

2.6.3 Member Function Documentation

2.6.3.1 const data_type * DataContainer::getData (void) const

Copies the existing data in the DataContainer into a new memory location then returns the pointer to the memory location.

Warning

DOES NOT CLEAN UP MEMORY, MUST BE DONE OUTSIDE OF THIS FUNCTION SCOPE

Returns

Pointer to memory location of copy of data in the DataContainer.

2.6.3.2 data_type DataContainer::getData (int *row*, int *col*) const

Retrieve a specific value from the DataContainer.

Parameters

row	Row (entry location) of the data to retrieve.
col	Column (variable location) of the data to retrieve.

Returns

Data value corresponding to (row,col) ie (entry,variable)

2.6.3.3 Matrix DataContainer::getMatrix (void)

Returns Matrix representation of DaraContainer.

Returns

DataContainer object as a Matrix object.

2.6.3.4 int DataContainer::getNumberEntries (void) const

Returns the number of entries in the container.

Returns

Number of data entries in the container.

2.6.3.5 int DataContainer::getNumberVars (void) const

Returns the number of variable in the container.

Returns

Number of variable in the container.

2.6.3.6 void DataContainer::insertEntry (const data_type * *values*, int *row*)

Inserts an entire entry into the container.

Inserts an entire entry into the container, replacing existing values. If the row location > number entries the DataContainer is resized and padded with zeros.

Parameters

values	Pointer to memory location storing values to be inserted.
row	Row (entry location) to insert the entry values.

Warning

There is no check on the number of items passed to be.

2.6.3.7 void DataContainer::insertItem (data_type *value*, int *row*, int *col*)

Inserts a value into specific location in the container.

Inserts a value into specific location in the conntainer object. If row location is outside the current number of entries, the DataContainer is resized, padded with zeros.

Parameters

value	Real value to be inserted.
row	Row (entry location) to place value.
col	Column (variable location) to place value.

Warning

The column location must not exceed the number of variables.

2.6.3.8 DataContainer & DataContainer::operator= (const DataContainer & *dc*)

Overloaded Assigment operator.

Parameters

dc	Reference to DataContainer object to assign this DataContainer to.

Returns

Reference to this DataConatiner object after assignment.

2.6.3.9 void DataContainer::print (FILE * *fOut* = stderr) const [virtual]

Prints the data container to a file.

Parameters

fOut	File object pointer to print data container. Default is stderr if argument is not specified.

Reimplemented in ftd::ComplexDataContainer.

2.6.3.10 void DataContainer::resizeCopyOld (int *row*) [protected]

Helper function to resize the DataContiner.

Resizes DataContainer to a new number of rows (entries). Retains the old values and pads with zeros if needed.

Parameters

row	New number of rows (entries).

Warning

CURRENTLY ASSUMES rows > old rows!!!!

2.6.3.11 void DataContainer::setData (const data_type * *values*, int *nVars*, int *nEntries*)

Sets the data of the DataContainer to new values.

Parameters

values	Pointer to memory location of new data.
nVars	Number of variable (ie columns)
nEntries	Number of rows (ie rows)

Warning

ALL OLD VALUES ARE DESTROYED, NEW MEMORY ALLOCATED AND SET.

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

- DataContainer.h
- DataContainer.cpp

2.7 ftd::DataFileParser Class Reference

Object for parsing a fingerprint file.

`#include <DataFileParser.h>`

Public Member Functions

- DataFileParser (int nVars, const bool varUsage[ ], const std::string varNames[ ]=NULL)

  Constructor.
- virtual ~DataFileParser ()

  Destructor.
- uchar close ()

Closes the file.

- void copyVarNamesInUse (std::string varNames[ ]) const

  Copy name of the variables that were/are parsed.
- DataContainer getDataContainer (void) const

  Get the data from the data file.
- const int getNumEntries (void) const

  Get the number of entries that were/are parsed.
- const int getNVarsInUse (void) const

  Get the number of variables were/are parsed.
- uchar open (const char *fileName)

  Opens the file specified by file name.
- uchar parseData (int start, int nEntries)

  Parse out the values from the fingerprint file.
- void setVarNames (const std::string[ ] varNames, int n)

  Sets the names of the variables.
- void setVarUsage (const bool[ ] varUsage, int n)

  Sets bool value for which a variable is to be parsed.

2.7.1 Detailed Description

Object for parsing a fingerprint file.

2.7.2 Constructor & Destructor Documentation

2.7.2.1 ftd::DataFileParser::DataFileParser (int *nVars,* const bool *varUsage[ ],* const std::string *varNames[ ]* = NULL)

Constructor.

Parameters

nVars	Number of variables in the fingerpirint file.
varUsage	An array of bool specifying which variable are to be parsed out.
varaName	An array of strings with each variable's name, Default is NULL.

2.7.3 Member Function Documentation

2.7.3.1 uchar DataFileParser::close ( )

Closes the file.

Returns

Status of the close operation.

See also

fclose.

2.7.3.2 void ftd::DataFileParser::copyVarNamesInUse (std::string *varNames[ ]*) const

Copy name of the variables that were/are parsed.

Parameters

varNames	An array to store the names of the variables were/are parsed.

Warning

Input array must have the same size as returned by getNVarsInUse.

2.7.3.3 DataContainer DataFileParser::getDataContainer (void) const

Get the data from the data file.

Returns

All the parsed data in a DataContainer object.

2.7.3.4 const int DataFileParser::getNumEntries (void) const

Get the number of entries that were/are parsed.

Returns

Number of entries (rows) were/are parsed.

2.7.3.5 const int DataFileParser::getNVarsInUse (void) const

Get the number of variables were/are parsed.

Returns

Number of variables were/are parsed.

2.7.3.6 uchar DataFileParser::open (const char * *fileName*)

Opens the file specified by file name.

Parameters

fileName	Name of the file to be opened.

Returns

1 is successfully opened, 0 otherwise.

2.7.3.7 uchar DataFileParser::parseData (int *start* = 0, int *nEntries* = -1)

Parse out the values from the fingerprint file.

Parameters

start	Line to start parsing values.
nEntries	Number of entries(rows)

Returns

Status of the parse

See also

ftd::ParseResult

Warning

File must first be opened using the open function call prior to parsing.

2.7.3.8 void ftd::DataFileParser::setVarNames (const std::string[ ] *varNames*, int *n*)

Sets the names of the variables.

Parameters

varNames	An array of string variable names.
n	Number of variable names in the array.

2.7.3.9 void DataFileParser::setVarUsage (const bool[ ] *varUsage*, int *n*)

Sets bool value for which a variable is to be parsed.

Parameters

varUsage	An array of bool for variable usage.
n	Number of bool values in the array.

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

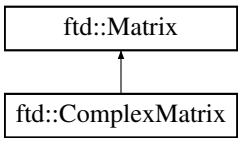
- DataFileParser.h
- DataFileParser.cpp

2.8 ftd::Matrix Class Reference

Container for real-valued matrix object.

`#include <Matrix.h>`

Inheritance diagram for ftd::Matrix:



Public Member Functions

- Matrix (int r, int c)

  Constructor, create Matrix with values initialized to zero.
- Matrix (int r, int c, data_type *data)

  Constructor, create Matrix and initialize values from memory.
- Matrix (const Array &a)

  Constructor, create Matrix from existing Array object.
- Matrix (const Matrix &m)

  Copy Constructor.
- ~Matrix ()

  Destructor.
- Array getCol (int c) const

  Get a column from a Matrix.
- data_type getData (int r, int c) const

  Get a value from a Matrix.
- int getNCols () const

  Return the number of columns in the Matrix.
- int getNRows () const

  Return the number of rows in the Matrix.
- Array getRow (int r) const

  Get a row from a Matrix.
- Matrix operator* (const Matrix &m) const

  Overloaded operator (*), performs matrix multiplication.
- Matrix operator* (data_type d) const

  Overloaded operator (*), Multiplies all values of Matrix by a single value.
- Matrix & operator*= (const Matrix &m)

  Overloaded operator (=), performs matrix multiplication and assigns to LHS.*
- Matrix & operator*= (data_type d)

  Overloaded operator (=), Multiplies all values of Matrix by a single value, assigns to LHS.*
- Matrix operator+ (const Matrix &m) const

  Overloaded operator (+), Add element-wise two matrices.
- Matrix operator+ (data_type d) const

  Overloaded operator (+), Add each value in Matrix by single value.
- Matrix & operator+= (const Matrix &m)

  Overloaded operator (+=), Add element-wise two matrices assign result to LHS.
- Matrix & operator+= (data_type d)

  Overloaded operator (+=), Add each value in Matrix by single value and assign to LHS.
- Matrix operator- (const Matrix &m) const

  Overloaded operator (-), Subtract element-wise two matrices.
- Matrix operator- (data_type d) const

  Overloaded operator (-), Subtract each value in Matrix by single value.
- Matrix & operator-= (const Matrix &m)

  Overloaded operator (-=), Subtract element-wise two matrices assign result to LHS.
- Matrix & operator-= (data_type d)

  Overloaded operator (-=), Subtract each value in Matrix by single value and assign to LHS.
- Matrix operator/ (data_type d) const

Overloaded operator (/), Divided all values of Matrix by a single value.

- Matrix & operator/= (data_type d)

  Overloaded operator (/=), Divides all values of Matrix by a single value, assigns to LHS.
- const Matrix & operator= (const Matrix &m)

  Overloaded assignment operator (=), non daisy chain allowed.
- virtual void print (FILE *fOut=stderr) const

  Print the values into a FILE object.
- void setData (data_type d, int r, int c)

  Set a single value within a Matrix specified by row, column.
- void setData (data_type d, int ind)

  Set a single value within a Matrix specified by raw index.
- void setData (data_type *values)

  Set all values of the Matrix.
- Array toArray (void) const

  Return values from the Matrix in a single Array object.
- void trunc (int r, int c)

  Truncate Matrix to specified number of rows/columns.

Protected Member Functions

- Matrix ()

  Default Constructor, protected so empty Matrix cannot be created.

Protected Attributes

- int cols

  Number of columns in Matrix.
- data_type * data

  Pointer to data for real values of the Matrix.
- int rows

  Number of rows in Matrix.

Friends

- Matrix bandPower (const Matrix &m, int bands, int dim)

  Bands together neighbouring values in a Matrix row/column-wise based on specified dimension.
- class ComplexMatrix

  Friend class of Matrix.
- Matrix covar (const Matrix &m)

  Scales Matrix values between [0,1] in specified dimension.
- Matrix elemDiv (const Matrix &a, const Matrix &b)

  Divides two Matrix objects elementally.
- Matrix elemMult (const Matrix &a, const Matrix &b)

  Multiplies two Matrix objects elementally.
- Matrix max (const Matrix &m, int dim)

  Calculates max values of a Matrix.

- Matrix mean (const Matrix &m, int)

  Calculates mean of a Matrix row/columnise.
- Matrix min (const Matrix &m, int dim)

  Calculates max values of a Matrix.
- Matrix normMaxMin (const Matrix &m, int dim)

  Scales Matrix values between [0,1] in specified dimension.
- Matrix ones (int r, int c)

  Create a Matrix full of all ones.
- Matrix reshape (const Matrix &m, int r, int c)

  Reshape a Matrix to specified number of rows and columns.
- Matrix sum (const Matrix &m, int dim)

  Calculates sum of a Matrix row/columnise.
- Matrix transpose (const Matrix &m)
- Matrix var (const Matrix &m)

  Calculates variance of a Matrix row/columnise.

2.8.1 Detailed Description

Container for real-valued matrix object.

2.8.2 Constructor & Destructor Documentation

2.8.2.1 Matrix::Matrix (int *r*, int *c*)

Constructor, create Matrix with values initialized to zero.

Parameters

r	Number of rows.
c	Number of columns.

2.8.2.2 Matrix::Matrix (int *r*, int *c*, data_type * *data*)

Constructor, create Matrix and initialize values from memory.

Parameters

r	Number of rows.
c	Number of columns.
data	Pointer to memory where values for Matrix are stored.

Warning

data is a pointer that must be allocated and initialized, number of must be r∗c and memory management must be done outside constructor scope.

2.8.2.3 Matrix::Matrix (const Array & *a*)

Constructor, create Matrix from existing Array object.

Size of the Matrix will be [1xM] where M is the number of elements in the input Array.

Parameters

a	Array to be used to create Matrix.

2.8.2.4 Matrix::Matrix (const Matrix & *m*)

Copy Constructor.

Existing Matrix values copied to New Matrix, size is preserved.

2.8.3 Member Function Documentation

2.8.3.1 Array Matrix::getCol (int *c*) const

Get a column from a Matrix.

Parameters

c	Column index.

Returns

Array object with values from specified column.

2.8.3.2 data_type Matrix::getData (int *r*, int *c*) const

Get a value from a Matrix.

Parameters

r	Row of element.
c	Column of element.

Returns

Value of item at specified row and column.

2.8.3.3 Array Matrix::getRow (int *r*) const

Get a row from a Matrix.

Parameters

r	Row index.

Returns

Array object with values from specified row.

2.8.3.4 Matrix Matrix::operator∗ (const Matrix & *m*) const

Overloaded operator (∗), performs matrix multiplication.

Matrix multiplication resulting in matrix of size [LHS_rows x RHS_columns].

Parameters

m	RHS Matrix for multiplication.

Returns

Copy of Matrix with values from the Matrix multiplication.

Warning

Proper Matrix sizes must be provided LHS_cols==RHS_rows!

2.8.3.5 Matrix Matrix::operator∗ (data_type *d*) const

Overloaded operator (∗), Multiplies all values of Matrix by a single value.

Parameters

d	Value to multiply all Matrix values with.

Returns

Copy of Matrix resulting from the scalar multiplication.

2.8.3.6 Matrix & Matrix::operator∗= (const Matrix & *m*)

Overloaded operator (∗=), performs matrix multiplication and assigns to LHS.

Matrix multiplication resulting in matrix of size [LHS_rows x RHS_columns].

Parameters

m	RHS Matrix for multiplication.

Returns

Reference to Matrix with values from the Matrix multiplication.

Warning

Proper Matrix sizes must be provided LHS_cols==RHS_rows!

2.8.3.7 Matrix & Matrix::operator∗= (data_type *d*)

Overloaded operator (∗=), Multiplies all values of Matrix by a single value, assigns to LHS.

Parameters

d	Value to multiply all Matrix values with.

Returns

Reference of Matrix resulting from the scalar multiplication.

2.8.3.8 Matrix Matrix::operator+ (const Matrix & *m*) const

Overloaded operator (+), Add element-wise two matrices.

Parameters

m	RHS of the addition.

Returns

Copy of the result of the matrix addition.

Warning

RHS and LHS must have the same number of elements.

2.8.3.9 Matrix Matrix::operator+ (data_type *d*) const

Overloaded operator (+), Add each value in Matrix by single value.

Parameters

d	Single value to addition from all values.

Returns

Copy of the resulting scalar addition.

2.8.3.10 Matrix & Matrix::operator+= (const Matrix & *m*)

Overloaded operator (+=), Add element-wise two matrices assign result to LHS.

Parameters

m	RHS of the addition.

Returns

Reference to the result of the matrix addition.

Warning

RHS and LHS must have the same number of elements.

2.8.3.11 Matrix & Matrix::operator+= (data_type *d*)

Overloaded operator (+=), Add each value in Matrix by single value and assign to LHS.

Parameters

d	Single value to addition from all values.

Returns

Reference to the resulting scalar addition.

2.8.3.12 Matrix Matrix::operator- (const Matrix & *m*) const

Overloaded operator (-), Subtract element-wise two matrices.

Parameters

m	RHS of the subtraction.

Returns

Copy of the result of the matrix subtraction.

Warning

RHS and LHS must have the same number of elements.

2.8.3.13 Matrix Matrix::operator- (data_type *d*) const

Overloaded operator (-), Subtract each value in Matrix by single value.

Parameters

d	Single value to subtract from all values.

Returns

Copy of the resulting scalar subtraction.

2.8.3.14 Matrix & Matrix::operator-= (const Matrix & *m*)

Overloaded operator (-=), Subtract element-wise two matrices assign result to LHS.

Parameters

m	RHS of the subtraction.

Returns

Reference to the result of the matrix subtraction.

Warning

RHS and LHS must have the same number of elements.

2.8.3.15 Matrix & Matrix::operator-= (data_type *d*)

Overloaded operator (-=), Subtract each value in Matrix by single value and assign to LHS.

Parameters

d	Single value to subtract from all values.

Returns

Reference to the resulting scalar subtraction.

2.8.3.16 Matrix Matrix::operator/ (data_type *d*) const

Overloaded operator (/), Divided all values of Matrix by a single value.

Parameters

d	Value to divide all Matrix values with.

Returns

Copy of Matrix resulting from the scalar division.

2.8.3.17 Matrix & Matrix::operator/= (data_type *d*)

Overloaded operator (/=), Divides all values of Matrix by a single value, assigns to LHS.

Parameters

d	Value to divide all Matrix values with.

Returns

Reference of Matrix resulting from the scalar division.

2.8.3.18 const Matrix & Matrix::operator= (const Matrix & *m*)

Overloaded assignment operator (=), non daisy chain allowed.

Parameters

m	Matrix to assign LHS.

2.8.3.19 void Matrix::print (FILE * *fOut* = `stderr`) const [virtual]

Print the values into a FILE object.

Parameters

fOut	Pointer to FILE object to print values, default is stderr.

Warning

FILE pointer fOut must be valid and opened for write then closed outside scope.

Reimplemented in ftd::ComplexMatrix.

2.8.3.20 void Matrix::setData (data_type *d*, int *r*, int *c*)

Set a single value within a Matrix specified by row, column.

Parameters

d	Value to be set.
r	Row of the value.
c	Column of the value.

2.8.3.21 void Matrix::setData (data_type *d*, int *ind*)

Set a single value within a Matrix specified by raw index.

Parameters

d	Value to be set.
ind	Raw index to store value in Matrix.

2.8.3.22 void Matrix::setData (data_type * *values*)

Set all values of the Matrix.

Parameters

values	Pointer to values to set in Matrix.

Warning

Input values must be created and initialized outside scope and number of elements must match that of the Matrix.

2.8.3.23 Array Matrix::toArray (void) const

Return values from the Matrix in a single Array object.

Returns

Array with all Matrix values, number elements are rows*cols.

2.8.3.24 void Matrix::trunc (int *r*, int *c*)

Truncate Matrix to specified number of rows/columns.

The truncation process keeps the first r rows and c columns.

Parameters

r	Number of rows to keep.
c	Number of columns to keep.

Warning

The number of rows/columns desired must be less than current counts.

2.8.4 Friends And Related Function Documentation

2.8.4.1 Matrix bandPower (const Matrix & *m*, int *bands*, int *dim*) [friend]

Bands together neighbouring values in a Matrix row/column-wise based on specified dimension.

Intended for calculating the band-power of frequency components a multi-variate signal. Frequencies are banded to generate the desired number of bands. Uneven band distribution is handled by making lower indexed values have larger number of components.

Parameters

m	Matrix containing value to be banded together.
bands	Total number of bands.
dim	Dimension to perform bands. dim==1 banding is across columns, dim==2 banding is across rows.

Returns

Matrix with banded values.

Warning

The number of bands must be <= than the number of elements in specified dimension.

2.8.4.2 Matrix covar (const Matrix & *m*) [friend]

Scales Matrix values between [0,1] in specified dimension.

If dim==1 normalization is done column-wise. If dim==2 normalization is done row-wise.

Returns

Resulting normalized Matrix.

2.8.4.3 Matrix elemDiv (const Matrix & *a*, const Matrix & *b*) [friend]

Divides two Matrix objects elementally.

Parameters

a	Numerator Matrix for elemental division.
b	Denominator Matrix for elemental division.

Returns

Matrix result of the elemental division.

2.8.4.4 Matrix elemMult (const Matrix & *a*, const Matrix & *b*) [friend]

Multiplies two Matrix objects elementally.

Parameters

a	Numerator Matrix for elemental multiplication.
b	Denominator Matrix for elemental multiplication.

Returns

Matrix result of the elemental multiplication.

2.8.4.5 Matrix max (const Matrix & *m*, int *dim*) [friend]

Calculates max values of a Matrix.

If dim==1 max is across columns result [1xM], If dim==2 max is across rows result [Nx1].

Parameters

m	Matrix to calculate max.
dim	Direction of the max operation.

Returns

Matrix of the resulting max operation.

2.8.4.6 Matrix mean (const Matrix & *m*, int *dim*) [friend]

Calculates mean of a Matrix row/columnise.

If dim==1 mean is across columns result [1xM], If dim==2 mean is across rows result [Nx1].

Parameters

m	Matrix to calculate mean.
dim	Direction of the mean operation.

Returns

Matrix of the resulting mean operation.

2.8.4.7 Matrix min (const Matrix & *m*, int *dim*) [friend]

Calculates max values of a Matrix.

If dim==1 min is across columns result [1xM], If dim==2 min is across rows result [Nx1].

Parameters

m	Matrix to calculate min.
dim	Direction of the min operation.

Returns

Matrix of the resulting min operation.

2.8.4.8 Matrix normMaxMin (const Matrix & *m*, int *dim*) [friend]

Scales Matrix values between [0,1] in specified dimension.

If dim==1 normalization is done column-wise. If dim==2 normalization is done row-wise.

Parameters

m	Matrix to be normalized.
dim	Direction of the normalization operation.

Returns

Resulting normalized Matrix.

2.8.4.9 Matrix ones (int *r*, int *c*) [friend]

Create a Matrix full of all ones.

Parameters

r	Number of rows in the Matrix.
c	Number of columns in the Matrix.

Returns

Ones Matrix of size [r x c].

2.8.4.10 Matrix reshape (const Matrix & *m*, int *r*, int *c*) [friend]

Reshape a Matrix to specified number of rows and columns.

Parameters

m	Matrix to be reshaped.
r	Number of rows of resulting reshape.
c	Number of columns of resulting reshape.

Returns

Reshaped Matrix with dimensions [r x c].

Warning

The desired reshaped Matrix must contain the same number elements (r_old∗c_old == r_new∗c_new).

2.8.4.11 Matrix sum (const Matrix & *m*, int *dim*) [friend]

Calculates sum of a Matrix row/columnise.

If dim==1 sum is across columns result [1xM], If dim==2 sum is across rows result [Nx1].

Parameters

m	Matrix to calculate sum.
dim	Direction of the sum operation.

Returns

Matrix of the resulting sum operation.

2.8.4.12 Matrix transpose (const Matrix & *m*) [friend]

Transposes a Matrix.

Parameters

m	Matrix to be transposed.

Returns

Transpose of Matrix m.

2.8.4.13 Matrix var (const Matrix & *m*) [friend]

Calculates variance of a Matrix row/columnise.

If dim==1 variance is across columns result [1xM], If dim==2 variance is across rows result [Nx1].

Parameters

m	Matrix to calculate variance.
dim	Direction of the variance operation.

Returns

Matrix of the resulting variance operation.

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

- Matrix.h
- Matrix.cpp

2.9 ftd::ModelParser Class Reference

Object for parsing a GMM model file.

`#include <ModelParser.h>`

Public Member Functions

- ModelParser ()

  Default constructor.

- virtual ~ModelParser ()

  Destructor.
- void copyVarNames (std::string varNames[ ]) const

  Make of copy of the variable names.
- void copyVarUsage (bool varUsage[ ]) const

  Make of copy of the variable names.
- int getDataStart (void) const

  Get the line number the data for the fingerprint file starts.
- Matrix * getDetRPtr (void) const

  Get the determinates of the GMM covariance matrices.
- int getEntryCount (void) const

  Get the number of entries expected in fingerprint file.
- Matrix * getMaxValuesPtr (void) const

  Get pointer to expected raw max values of variables in fingerprint.
- Matrix * getMeanPCAPtr (void) const

  Get pointer to expected mean feature values prior to PCA.
- Matrix * getMeanPtr (void) const

  Get pointer to GMM mean values.
- Matrix * getMeanValuesPtr (void) const

  Get pointer to expected raw mean values of variables in fingerprint.
- Matrix * getMinValuesPtr (void) const

  Get pointer to expected raw min values of variables in fingerprint.
- int getNBands (void) const

  Get the number of bands used in bandpower.
- int getNFeatures (void) const

  Get the number of features after PCA is performed.
- int getNUsedVars (void) const

  Get number of variables used from the fingerprint during classification.
- PCA_OPTS getPCAOpts (void) const

  Get the PCA option used.
- Matrix * getRInvPtr (void) const

  Get pointer to inverse values of GMM covariance matrices.
- S_OPTS getScalingOpts (void) const

  Get the scaling processing option used.
- Matrix * getStdPCAPtr (void) const

  Get pointer to expected std of feature values prior to PCA.
- data_type getTH (void) const

  Get the threshold log likelihood for normal.
- int getVarCount (void) const

  Get the total number of variables in fingerprint file.
- Matrix * getWPCAPtr (void) const

  Get pointer to PCA matrix.
- Matrix * getWPtr (void) const

  Get the pointer to matrix containing GMM weights.
- ZM_OPTS getZeroMeanOpts (void) const

  Get the zero-mean processing option used.
- uchar open (const char *fileName)

  Opens the model file. fileName Absolute path to the model file to be paresed.
- bool parse (void)

  Parses the model file.

2.9.1 Detailed Description

Object for parsing a GMM model file.

Parses and extracts information contained in a .csv model file for GMM.

2.9.2 Member Function Documentation

2.9.2.1 void ModelParser::copyVarNames (std::string *varNames[ ]*) const

Make of copy of the variable names.

Parameters

varNames	Array that variables will be copied into.

Warning

Array must be the same size as what returns from getVarCount

2.9.2.2 void ModelParser::copyVarUsage (bool *varUsage[ ]*) const

Make of copy of the variable names.

Parameters

varUsage	Array that variables will be copied into.

Warning

Array must be the same size as what returns from getVarCount.

2.9.2.3 Matrix* ftd::ModelParser::getDetRPtr (void) const [inline]

Get the determinates of the GMM covariance matrices.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.4 Matrix* ftd::ModelParser::getMaxValuesPtr (void) const [inline]

Get pointer to expected raw max values of variables in fingerprint.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.5 Matrix* ftd::ModelParser::getMeanPCAPtr (void) const [inline]

Get pointer to expected mean feature values prior to PCA.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.6 Matrix* ftd::ModelParser::getMeanPtr (void) const [inline]

Get pointer to GMM mean values.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.7 Matrix* ftd::ModelParser::getMeanValuesPtr (void) const [inline]

Get pointer to expected raw mean values of variables in fingerprint.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.8 Matrix* ftd::ModelParser::getMinValuesPtr (void) const [inline]

Get pointer to expected raw min values of variables in fingerprint.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.9 Matrix* ftd::ModelParser::getRInvPtr (void) const [inline]

Get pointer to inverse values of GMM covariance matrices.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.10 Matrix* ftd::ModelParser::getStdPCAPtr (void) const [inline]

Get pointer to expected std of feature values prior to PCA.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.11 Matrix* ftd::ModelParser::getWPCAPtr (void) const [inline]

Get pointer to PCA matrix.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.12 Matrix* ftd::ModelParser::getWPtr (void) const [inline]

Get the pointer to matrix containing GMM weights.

Warning

Pointer returned is only valid while ModelParser object is not destroyed.

2.9.2.13 uchar ModelParser::open (const char * *fileName*)

Opens the model file. fileName Absolute path to the model file to be paresed.

Returns

1 if opens successfully, else 0.

2.9.2.14 bool ModelParser::parse (void)

Parses the model file.

Parses the model file and reports any errors/warning or other status during the parsing process based on default logging level (ftd::MAX_LOG_LEVEL).

Returns

True is the model file parses correctly else False.

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

- ModelParser.h
- ModelParser.cpp

2.10 variant Union Reference

Special union for storing any type in the same memory location.

`#include <Types.h>`

Public Attributes

- char **c**
- char * **cptr**
- float32 **f**
- float32 * **fptr**
- int32 **i**
- int32 * **iptr**
- uint32 **u**
- uint32 * **uptr**

2.10.1 Detailed Description

Special union for storing any type in the same memory location.

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

- Types.h

Chapter 3

File Documentation

3.1 Array.h File Reference

Container for Array of Real/Complex data and Complex structure.

```
#include "Types.h"
#include "math_func.h"
#include <math.h>
```

Classes

- class ftd::Array

  Container object for array of real values.
- struct ftd::Complex

  Structure for complex number representation.
- class ftd::ComplexArray

  Container object for array of real or complex values.

3.1.1 Detailed Description

Container for Array of Real/Complex data and Complex structure.

Container object to hold contents of an array of numbers real or complex. Intended for use of insertion and fetch from Matrix} objects. Also, created for ease for implementation of FFT.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

3.2 DataContainer.h File Reference

Header file for DataContainer and ComplexDataConatiner.

```
#include "Types.h"
#include "Matrix.h"
```

Classes

- class ftd::ComplexDataContainer

  Container Object for data file with complex values.
- class ftd::DataContainer

  Container class for real-values data files.

3.2.1 Detailed Description

Header file for DataContainer and ComplexDataConatiner.

Class container object designed to hold the contents of a data file. Data file has variables and entries. Supports both real and complex values.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

3.3 DataFileParser.h File Reference

Header file for DataFileParser class.

```
#include "DataContainer.h"
#include <stdio.h>
#include <stdlib.h>
#include <string>
```

Classes

- class ftd::DataFileParser

  Object for parsing a fingerprint file.

3.3.1 Detailed Description

Header file for DataFileParser class.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

This code was created by Rochester Institute of Technology in research collaboration with MKS Instruments.

3.4 fault_detection.h File Reference

Header includes all other headers required for Fault Detection Library.

```
#include "Types.h"
#include "Array.h"
#include "Matrix.h"
#include "functions.h"
#include "math_func.h"
#include "DataContainer.h"
#include "DataFileParser.h"
#include "ModelParser.h"
```

3.4.1 Detailed Description

Header includes all other headers required for Fault Detection Library.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

This code was created by Rochester Institute of Technology in research collaboration with MKS Instruments.

3.5 functions.h File Reference

Header for various high-level function prototypes.

```
#include "fault_detection.h"
```

Functions

- int **ftd::getNEntries** (void)

  Get the number of entries expected to be in a test fingerprint file.
- int **ftd::getNVars** (void)

  Get the total number of variables expected to be in a test file.
- int **ftd::getNVarsUsed** (void)

  Get the number of variables actually used from a test fingerprint.
- void **ftd::getVarUsage** (bool ∗varUsage)

  Obtain boolean values of the usage of the variables in a test fingerprint.
- bool **ftd::loadModelParser** (const char ∗pathToModel)

  Loads GMM model information into memory.
- void **ftd::report** (const char ∗message, uchar loglevel)

  Method to handle all reporting of errors, warnings, status, etc.
- uchar **ftd::runOneClass** (data_type ∗data)

  Executes the one-class classifier from data parsed into memory.
- uchar **ftd::runOneClassGen** (const char ∗pathToData, const char ∗pathToModel, uchar logLevel=LOG_NOTICE)

  Executes the one-class classifier, intended to be ran within embedded environment.
- void **ftd::unloadModelParser** (void)

3.5.1 Detailed Description

Header for various high-level function prototypes.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

This code was created by Rochester Institute of Technology in research collaboration with MKS Instruments.

3.6 math_func.h File Reference

Various high-level math functions.

```
#include <math.h>
```

Functions

- int **ftd::getFFTPadSize** (int len)

  Returns size of zero padding required for FFT.

3.6.1 Detailed Description

Various high-level math functions.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

3.7 ModelParser.h File Reference

Matrix class implementation, real and complex. Several operations and other Matrix related functions.

```
#include <string.h>
#include "Types.h"
#include "Matrix.h"
```

Classes

- class ftd::ModelParser

  Object for parsing a GMM model file.

Enumerations

- enum **PCA_OPTS** { **PCA_NONE** =0, **PCA_ZM**, **PCA_UV**, **PCA_ZM_UV** }

  Enumeration for the different PCA options.
- enum **S_OPTS** {
  S_NONE =0, **S_MAX_LOCAL**, **S_MAX_GLOBAL**, **S_MAXMIN_LOCAL**,
  S_MAXMIN_GLOBAL }

  Enumeration for the different scaling options.
- enum **ZM_OPTS** { **ZM_NONE** =0, **ZM_LOCAL**, **ZM_GLOBAL** }

  Enumeration for the different zero-mean options.

Variables

- const std::string **ftd::logFileName** = "mog_log.log"

  Default log file name (NOT USED).
- const std::string **ftd::modelFileName** = "mog_model.csv"

  Default model file name.
- const std::string **ftd::rel_mog_dir** = "/../mog_files/"

Default relative path to model file (NOT USED).

- const std::string **ftd::STR_BANDS** = "bands"

  Model file tag for number of bands for FFT.
- const std::string **ftd::STR_DATA_START** = "data_start"

  Model file tag for line that data starts in fingerprint.
- const std::string **ftd::STR_DET_R** = "det_r"

  Model file tag for GMM determinate of covariances.
- const std::string **ftd::STR_MAX_VALUES** = "max_values"

  Model file tag for raw variable max values.
- const std::string **ftd::STR_MEAN** = "mean"

  Model file tag for GMM mean values.
- const std::string **ftd::STR_MEAN_PCA** = "mean_pca"

  Model file tag for mean of features prior to PCA.
- const std::string **ftd::STR_MEAN_VALUES** = "mean_values"

  Model file tag for raw variable mean values.
- const std::string **ftd::STR_MIN_VALUES** = "min_values"

  Model file tag for raw variable min values.
- const std::string **ftd::STR_N_ENTRIES** = "n_entries"

  Model file tag for number of fingerprint entries.
- const std::string **ftd::STR_N_FEATURES** = "n_features"

  Model file tag for number of features after PCA.
- const std::string **ftd::STR_N_VARS** = "n_vars"

  Model file tag for number of fingerprint variables.
- const std::string **ftd::STR_OPT_TRUE** = "true"

  Model file value for a true option.
- const std::string **ftd::STR_OPT_YES** = "yes"

  Model file value for a yes option.
- const std::string **ftd::STR_PROC_SCALING** = "proc_scaling"

  Model file tag for scaling options.
- const std::string **ftd::STR_PROC_ZEROMEAN** = "proc_zeromean"

  Model file tag for zero-mean options.
- const std::string **ftd::STR_R_INV** = "r_inv"

  Model file tag for GMM inverse of covariances.
- const std::string **ftd::STR_RED_PCA** = "red_pca"

  Model file tag for PCA options.
- const std::string **ftd::STR_SCALE_MAX** = "max"

  Model file value for max-scale option.
- const std::string **ftd::STR_SCALE_MAXMIN** = "maxmin"

  Model file value for maxmin-scale option.
- const std::string **ftd::STR_SCOPE_GLOBAL** = "global"

  Model file value for global option.
- const std::string **ftd::STR_SCOPE_LOCAL** = "local"

  Model file value for local option.
- const std::string **ftd::STR_STD_PCA** = "std_pca"

  Model file tag for std of features prior to PCA.
- const std::string **ftd::STR_TH** = "th"

  Model file tag for GMM normal threshold value.

- const std::string **ftd::STR_VAR_NAMES** = "var_names"

  Model file tag for name of the variables in fingerprint.
- const std::string **ftd::STR_VAR_USAGE** = "var_usage"

  Model file tag for boolean array of which variables are used.
- const std::string **ftd::STR_W** = "w"

  Model file tag for GMM weight values.
- const std::string **ftd::STR_WPCA** = "wpca"

  Model file tag for PCA matrix values.
- const uint16 **ftd::STR_ZEROMEAN_COUNT** = 2

3.7.1 Detailed Description

Matrix class implementation, real and complex. Several operations and other Matrix related functions.

Class intended to help parse csv files with model information.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

This code was created by Rochester Institute of Technology in research collaboration with MKS Instruments.

3.8 Types.h File Reference

All required typdefs, namespaces, etc.

```
#include <syslog.h>
```

Classes

- union variant

  Special union for storing any type in the same memory location.

Macros

- #define **infinity**() INFINITY
- #define **infinityf**() INFINITY
- #define **LINUXx86**
- #define **NULL** 0
- #define **variant32** variant

Typedefs

- typedef float **ftd::data_type**

  Typedef controls the data type used in all numerical containers and operations.
- typedef float **float32**
- typedef double **float64**
- typedef long double **float96**
- typedef signed short **int16**
- typedef signed int **int32**
- typedef signed long long **int64**
- typedef signed char **int8**
- typedef unsigned char **ftd::uchar**

  Unsigned character typedef.
- typedef unsigned short **uint16**
- typedef unsigned int **uint32**
- typedef unsigned long long **uint64**
- typedef unsigned char **uint8**

Enumerations

- enum **DataLabel** { **NORMAL**, **FAULTY**, **UNKNOWN** }

  Enumeration for possible labels for a fingerprint.
- enum **OneClassResult** {
  RES_NORMAL, **RES_FAULTY**, **ERROR_DATA**, **ERROR_MODEL**,
  ERROR_ALG, **ERROR_PATH**, **ERROR_LOGFILE**, **ERROR_OTHER** }

  Enumeration for possible results from the one-class classier, including error states.
- enum **ParseResult** { **PARSE_OK**, **ERROR_VARCOUNT**, **ERROR_ENTRYCOUNT** }

  Enumeration for parsing results.

Variables

- const data_type **ftd::EPS** = 1.192092896e-07F

  Definition of what is coisder a the smallest value.
- int **ftd::MAX_LOG_LEVEL** = LOG_INFO

  Default Log Serverity.
- const data_type **ftd::PI** = 3.14159265358979f

  Definition of PI.

3.8.1 Detailed Description

All required typdefs, namespaces, etc.

Author

Ryan M. Bowen (rmb3518@rit.edu)
Ferat Sahin (feseee@rit.edu)

Date

02/20/2014 - 06/30/2015

Copyright

Processor independent fundamental types. Edit the Processor types define below to set the proper type definitions.

Index