Structural Analysis Library

1.0

Generated by Doxygen 1.5.6

Thu Oct 23 09:41:28 2008

Contents

Chapter 1

Structural Analysis Library

This document describes the application programming interface (API) of LibLA and LibStructural an open source (BSD) library for computing structural characteristics of cellular networks.

LibLA is a linear algebra library derives much of its functionality from the standard CLAPACK library with additional linear algebra functions not directly supported by CLAPACK. The libStructural library supports a range of methods for the structural analysis of cellular networks (derived either from SBML or stoichiometry matrices) and utilizes LibLA for some of its internal computations.

Installing

To make the Structural Analysis Library easily accessible we have created binary installers for Windows as wel as OS X (version 10.4 and above). We also habe a source distribution, complete with Visual Studio, XCode, Scons and Qt project files that allow to build the library on Windows, Linux and OS X. For detailed instructions on how to build the library see the file INSTALL included with the source distribution.

Dependencies

These libraries depend on two third-party libraries, LAPACK and libSBML. Both are provided with the binary installation where necessary.

This work was supported by a grant from the NIH (1R01GM0819070-01).

Author:

Frank T. Bergmann (fbergman@u.washington.edu)
Herbert M. Sauro
Ravishankar Rao Vallabhajosyula (developed a previous version of the sructural analysis code)

License

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 LIB_LA Namespace Reference

The LIB_LA namespace contains all functions and classes enabling the Linear Algebra functions.

Classes

- class Complex

  LIB_LA::Complex is the complex class used by LIB_LA::LibLA.

- class LibLA

  The LIB_LA::LibLA class represents the entry point to a variety of useful functionality operating on double and complex matrices.

- class LU_Result

  LUResult is intended to hold the return values of the Clapack LU decomposition methods.

- class Matrix

  LIB_LA::Matrix is the matrix class used by LIB_LA::LibLA and LIB_STRUTURAL::LibStructural.

Typedefs

- typedef Matrix< Complex > ComplexMatrix

  defines a complex matrix (hides the templates in signatures)

- typedef Matrix< double > DoubleMatrix

  defines a real matrix (hides the templates in signatures)

- typedef Matrix< int > IntMatrix

  defines a integer matrix (hides the templates in signatures)

Functions

- std::ostream & operator<< (std::ostream &os, const Complex &complex)
  overload that allows to print a complex number on a std::stream

5.1.1 Detailed Description

The LIB_LA namespace contains all functions and classes enabling the Linear Algebra functions.

The namespace consists mainly of three classes LIB_LA::Complex, a straight forward implementation of a complex type, LIB_LA::Matrix a template matrix class used by the C++ API and of course LIB_LA:LibLA the entry point of the LA library which encapsulates all functionality.

5.1.2 Typedef Documentation

5.1.2.1 typedef Matrix< Complex > LIB_LA::ComplexMatrix

defines a complex matrix (hides the templates in signatures)

5.1.2.2 typedef Matrix< double > LIB_LA::DoubleMatrix

defines a real matrix (hides the templates in signatures)

Examples:

examples/cpp/loadstoichiometry.cpp.

5.1.2.3 typedef Matrix< int > LIB_LA::IntMatrix

defines a integer matrix (hides the templates in signatures)

5.1.3 Function Documentation

5.1.3.1 std::ostream& LIB_LA::operator<< (std::ostream & *os*, const Complex & *complex*)

overload that allows to print a complex number on a std::stream

This function enables a complex number to be displayed on a stream. It will be formatted as: '(' + realpart + ' + ' + imaginaryPart + 'i)'. To use it invoke for example:

Complex number(1.0, 0.5); cout << number << endl;

Parameters:

os output stream to print on

complex the complex number to be printed

Returns:

the output stream containing the printed complex number

5.2 LIB_STRUCTURAL Namespace Reference

The LIB_STRUCTURAL namespace contains all functions and classes directly related to Structural Analysis.

Classes

- class LibStructural

  Entrypoint for the C++ API of the Structural Analysis Library.

5.2.1 Detailed Description

The LIB_STRUCTURAL namespace contains all functions and classes directly related to Structural Analysis.

The namespace consists mainly of two classes LIB_STRUCTURAL::LibStructural, the class performing all the structural analysis of SBML models, or Stoichiometry matrices, and LIB_STRUCTURAL::SBMLmodel, a small utility class for easy access of the needed information.

Chapter 6

Class Documentation

6.1 LIB_LA::Complex Class Reference

LIB_LA::Complex is the complex class used by LIB_LA::LibLA.

`#include <complex.h>`

Public Member Functions

- Complex (double real=0.0, double imag=0.0)
  constructs a new complex number with given real and imaginary part
- double getImag ()
  return the complex part of the complex number
- double getReal ()
  returns the real part of the complex number
- virtual Complex & operator∗ (const Complex &rhs)
  implements multiplication of complex numbers
- virtual Complex & operator+ (const Complex &rhs)
  implements addition of complex numbers
- virtual Complex & operator- (const Complex &rhs)
  implements subtraction of complex numbers
- virtual Complex & operator/ (const Complex &rhs)
  implements complex division
- virtual std::basic_ostream< char > & operator<< (std::basic_ostream< char > &os)
  print the complex number on an output stream
- virtual Complex & operator= (const Complex &rhs)
  assignment operator

- virtual Complex & operator= (const double rhs)
  assignment operator (sets the real part only)
- void set (double real, double imag)
  sets real and imaginary part of the complex number
- void setImag (double imag)
  sets the imaginary part of the complex number
- void setReal (double real)
  sets the real part of the complex number

Public Attributes

- double Imag
  imaginary part of the complex number
- double Real
  real part of the complex number

6.1.1 Detailed Description

LIB_LA::Complex is the complex class used by LIB_LA::LibLA.

This class implements a basic complex type along with basic operations on it.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 LIB_LA::Complex::Complex (double *real* = `0.0`, double *imag* = `0.0`) `[inline]`

constructs a new complex number with given real and imaginary part

6.1.3 Member Function Documentation

6.1.3.1 double LIB_LA::Complex::getImag () `[inline]`

return the complex part of the complex number

6.1.3.2 double LIB_LA::Complex::getReal () `[inline]`

returns the real part of the complex number

6.1.3.3 virtual Complex& LIB_LA::Complex::operator* (const Complex & *rhs*) `[inline, virtual]`

implements multiplication of complex numbers

6.1.3.4 virtual Complex& LIB_LA::Complex::operator+ (const Complex & *rhs*) [inline, virtual]

implements addition of complex numbers

6.1.3.5 virtual Complex& LIB_LA::Complex::operator- (const Complex & *rhs*) [inline, virtual]

implements subtraction of complex numbers

6.1.3.6 virtual Complex& LIB_LA::Complex::operator/ (const Complex & *rhs*) [inline, virtual]

implements complex division

6.1.3.7 virtual std::basic_ostream<char>& LIB_LA::Complex::operator<< (std::basic_ostream< char > & *os*) [virtual]

print the complex number on an output stream

6.1.3.8 virtual Complex& LIB_LA::Complex::operator= (const Complex & *rhs*) [inline, virtual]

assignment operator

6.1.3.9 virtual Complex& LIB_LA::Complex::operator= (const double *rhs*) [inline, virtual]

assignment operator (sets the real part only)

6.1.3.10 void LIB_LA::Complex::set (double *real*, double *imag*) [inline]

sets real and imaginary part of the complex number

6.1.3.11 void LIB_LA::Complex::setImag (double *imag*) [inline]

sets the imaginary part of the complex number

6.1.3.12 void LIB_LA::Complex::setReal (double *real*) [inline]

sets the real part of the complex number

6.1.4 Member Data Documentation

6.1.4.1 double LIB_LA::Complex::Imag

imaginary part of the complex number

6.1.4.2 double LIB_LA::Complex::Real

real part of the complex number

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

- complex.h

6.2 LIB_LA::LibLA Class Reference

The LIB_LA::LibLA class represents the entry point to a variety of useful functionality operating on double and complex matrices.

`#include <libla.h>`

Public Member Functions

- void fullyPivotedGaussJordan (DoubleMatrix &oMatrix, std::vector< int > &rowPivots, std::vector< int > &colPivots)

  This method calculates the fully pivoted Gauss Jordan form of the given matrix.

- std::vector< int > gaussJordan (DoubleMatrix &oMatrix)

  This method calculates the Gauss Jordan or row echelon form of the given matrix.

- std::vector< Complex > getEigenValues (DoubleMatrix &oMatrix)

  Calculates the eigen-values of a square real matrix.

- DoubleMatrix * getLeftNullSpace (DoubleMatrix &oMatrix)

  This function calculates the left null space of a given real matrix.

- LU_Result * getLU (DoubleMatrix &oMatrix)

  This function computes the LU factorization of the given real M-by-N matrix A.

- LU_Result * getLUwithFullPivoting (DoubleMatrix &oMatrix)

  This function computes the LU factorization of the given real N-by-N matrix A using complete pivoting (with row and column interchanges).

- std::vector< DoubleMatrix * > getQR (DoubleMatrix &oMatrix)

  This function computes the QR factorization of the given real M-by-N matrix A.

- std::vector< DoubleMatrix * > getQRWithPivot (DoubleMatrix &oMatrix)

  This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

- int getRank (DoubleMatrix &oMatrix)

  This method computes the rank of the given matrix.

- DoubleMatrix * getRightNullSpace (DoubleMatrix &oMatrix)

  This function calculates the right null space of a given real matrix.

- DoubleMatrix * getScaledLeftNullSpace (DoubleMatrix &oMatrix)

  This function calculates the scaled left null space of a given real matrix.

- DoubleMatrix * getScaledRightNullSpace (DoubleMatrix &oMatrix)

  This function calculates the scaled right null space of a given real matrix.

- std::vector< double > getSingularValsBySVD (DoubleMatrix &oMatrix)

  This method performs the Singular Value Decomposition of the given real matrix, returning only the singular values.

- double getTolerance ()

  Returns the currently used tolerance.

- DoubleMatrix * inverse (DoubleMatrix &oMatrix)

  This function calculates the inverse of a square real matrix.

- LibLA ()

  Constructor of a new instance of LIB_LA::LibLA with a default tolerance of 1E-12.

- void setTolerance (double dTolerance)

  Set user specified tolerance.

- std::vector< Complex > ZgetEigenValues (ComplexMatrix &oMatrix)

  Calculates the eigen-values of a square complex matrix.

- ComplexMatrix * Zinverse (ComplexMatrix &oMatrix)

  This function calculates the inverse of a square complex matrix.

Static Public Member Functions

- static LibLA * getInstance ()

  Provides access to a singleton of this class.

6.2.1 Detailed Description

The LIB_LA::LibLA class represents the entry point to a variety of useful functionality operating on double and complex matrices.

The current scope of the library encompasses matrix factorizations (QR and LU factorization) as well as commonly needed matrix operations, such as calculating the inverse of a matrix, computing eigen values and singular values as well as the null space of a matrix (both left and right null space) along with a method for the computation of the row echelon or Gauss Jordan form of a matrix.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 LIB_LA::LibLA::LibLA () [inline]

Constructor of a new instance of LIB_LA::LibLA with a default tolerance of 1E-12.

See also LIB_LA::LibLA::setTolerance

6.2.3 Member Function Documentation

6.2.3.1 void LIB_LA::LibLA::fullyPivotedGaussJordan (DoubleMatrix & *oMatrix*, std::vector< int > & *rowPivots*, std::vector< int > & *colPivots*)

This method calculates the fully pivoted Gauss Jordan form of the given matrix.

Fully pivoted means, that rows as well as column swaps will be used. These permutations are captured in the integer vectors rowPivots and colPivots.

If no permutations have occurred those vectors will be in ascending form [0, 1, 2, 3]; However if say row one and three would be swapped this vector would look like: [0, 3, 2, 1];

6.2.3.2 std::vector<int> LIB_LA::LibLA::gaussJordan (DoubleMatrix & *oMatrix*)

This method calculates the Gauss Jordan or row echelon form of the given matrix.

Only row swaps are used. These permutations will be returned in the 'pivots' vector.

If no permutations have occurred this vector will be in ascending form [0, 1, 2, 3]; However if say row one and three would be swapped this vector would look like: [0, 3, 2, 1];

Returns:

the pivots vector

6.2.3.3 std::vector< Complex > LIB_LA::LibLA::getEigenValues (DoubleMatrix & *oMatrix*)

Calculates the eigen-values of a square real matrix.

This function calculates the complex eigenvalues of the given real matrix. The complex vector of eigenvalues will be returned in two real vectors, one for the real and one for the imaginary part.

Parameters:

oMatrix a real matrix

Returns:

a vector of LIB_LA::Complex numbers representing the eigen-values of the matrix

6.2.3.4 static LibLA* LIB_LA::LibLA::getInstance () [static]

Provides access to a singleton of this class.

6.2.3.5 DoubleMatrix* LIB_LA::LibLA::getLeftNullSpace (DoubleMatrix & *oMatrix*)

This function calculates the left null space of a given real matrix.

That is:

null(A)*A = 0

Remarks:

This function is equivalent to returning the right null space of the transposed matrix. See LIB_LA::LibLA::getRightNullSpace. It should also be noted that the values are unscaled yielding rational numbers. For a scaled version see LIB_LA::LibLA::getScaledLeftNullSpace

Parameters:

oMatrix a real matrix

Returns:

a matrix representing the left null space

6.2.3.6 LU_Result* LIB_LA::LibLA::getLU (DoubleMatrix & *oMatrix*)

This function computes the LU factorization of the given real M-by-N matrix A.

using partial pivoting with row interchanges. This procedure is carried out by the LAPACK method dgetrf . A is factorized into:

A = P * L * U

Here P is the row permutation matrix.

Parameters:

oMatrix a real matrix

Returns:

a LIB_LA::LU_Result object with empty LIB_LA::LU_Result::Q matrix

Remarks:

The LU factorization is unstable, please check the status flag: LIB_LA::LU_Result::nInfo

6.2.3.7 LU_Result* LIB_LA::LibLA::getLUwithFullPivoting (DoubleMatrix & *oMatrix*)

This function computes the LU factorization of the given real N-by-N matrix A using complete pivoting (with row and column interchanges).

This procedure is carried out by the LAPACK method dgetc2.

A is factorized into:

A = P * L * U * Q

Here P and Q are permutation matrices for the rows and columns respectively.

Remarks:

This function supports only square matrices (N-by-N), choose LibLA_getQRWithPivot for a stable method operating on N-by-M matrices.

Parameters:

oMatrix a real matrix

Returns:

a LIB_LA::LU_Result object

Remarks:

The LU factorization is unstable, please check the status flag: LIB_LA::LU_Result::nInfo

6.2.3.8 std::vector< DoubleMatrix* > LIB_LA::LibLA::getQR (DoubleMatrix & *oMatrix*)

This function computes the QR factorization of the given real M-by-N matrix A.

The LAPACK method dgeqp3 is used followed by an orthonormalization of Q through the use of DORGQR.

The factorized form is:

$$A = Q * R$$

In order to also perform column pivoting use LIB_LA::LibLA::getQRWithPivot

Returns:

This call yields a vector of matrices. These matrices are (in order):

- Q an orthogonal matrix.
- R an upper triangular matrix

Remarks:

free all matrices using 'delete'.

6.2.3.9 std::vector< DoubleMatrix* > LIB_LA::LibLA::getQRWithPivot (DoubleMatrix & *oMatrix*)

This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

The LAPACK method dgeqp3 is used followed by an orthonormalization of Q through the use of DORGQR.

The factorized form is:

$$A = Q * R$$

Returns:

This call yields a vector of matrices. These matrices are (in order):

- Q an orthogonal matrix.
- R an upper triangular matrix
- P a permutation matrix,

Remarks:

free all matrices using 'delete'.

6.2.3.10 int LIB_LA::LibLA::getRank (DoubleMatrix & *oMatrix*)

This method computes the rank of the given matrix.

The singular values of the matrix are calculated and the rank is determined by the number of non-zero values.

Note that zero here is defined as any value whose absolute value is bigger than the set tolerance (see LIB_LA::LibLA::setTolerance)

Parameters:

oMatrix a real matrix

Returns:

the rank of the matrix

6.2.3.11 DoubleMatrix* LIB_LA::LibLA::getRightNullSpace (DoubleMatrix & *oMatrix*)

This function calculates the right null space of a given real matrix.

That is:

$$A * null(A) = 0$$

In order to calculate the (right) null space, we first calculate the full singular value decomposition (employing dgesdd) of the matrix:

```
[U,S,V] = svd(A');
```

then calculate the rank:

```
r = rank(A)
```

and finally return the last columns of the U matrix (r+1...n) as the null space matrix.

Remarks:

It should also be noted that the values are unscaled yielding rational numbers. For a scaled version see LIB_LA::LibLA::getScaledRightNullSpace

Parameters:

oMatrix a real matrix

Returns:

a matrix representing the right null space

6.2.3.12 DoubleMatrix* LIB_LA::LibLA::getScaledLeftNullSpace (DoubleMatrix & *oMatrix*)

This function calculates the scaled left null space of a given real matrix.

This function is equivalent to calling LIB_LA::LibLA::getLeftNullSpace however the resulting matrix will be scaled (employing Gauss Jordan factorization) to yield whole numbered entries wherever possible.

Parameters:

oMatrix a real matrix

Returns:

a matrix representing the scaled left null space

6.2.3.13 DoubleMatrix* LIB_LA::LibLA::getScaledRightNullSpace (DoubleMatrix & *oMatrix*)

This function calculates the scaled right null space of a given real matrix.

This function is equivalent to calling LIB_LA::LibLA::getRightNullSpace however the resulting matrix will be scaled (employing Gauss Jordan factorization) to yield whole numbered entries wherever possible.

Parameters:

oMatrix a real matrix

Returns:

a matrix representing the scaled right null space

6.2.3.14 std::vector< double > LIB_LA::LibLA::getSingularValsBySVD (DoubleMatrix & *oMatrix*)

This method performs the Singular Value Decomposition of the given real matrix, returning only the singular values.

This procedure is carried out by the LAPACK method dgesdd.

Parameters:

oMatrix a real matrix

Returns:

a vector of (real) singular values

6.2.3.15 double LIB_LA::LibLA::getTolerance () `[inline]`

Returns the currently used tolerance.

This function returns the tolerance currently used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered zero and will be neglected.

6.2.3.16 DoubleMatrix* LIB_LA::LibLA::inverse (DoubleMatrix & *oMatrix*)

This function calculates the inverse of a square real matrix.

This procedure is carried out by the LAPACK methods dgetrf and dgetri. This means that the matrix will be factorized using LU decomposition first, followed by the calculation of the inverse based on:

inv(A)*L = inv(U) for inv(A).

Parameters:

oMatrix a real matrix

Returns:

the inverse of the real matrix

6.2.3.17 void LIB_LA::LibLA::setTolerance (double *dTolerance*) [inline]

Set user specified tolerance.

This function sets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

Parameters:

dTolerance Sets the tolerance used by the library to determine a value close to zero

6.2.3.18 std::vector< Complex > LIB_LA::LibLA::ZgetEigenValues (ComplexMatrix & *oMatrix*)

Calculates the eigen-values of a square complex matrix.

This function calculates the complex eigenvalues of the given complex matrix. The input matrix should be broken up into two matrices representing the real and imaginary parts respectively.

Parameters:

oMatrix a complex matrix

Returns:

a vector of LIB_LA::Complex numbers representing the eigen-values of the matrix

6.2.3.19 ComplexMatrix* LIB_LA::LibLA::Zinverse (ComplexMatrix & *oMatrix*)

This function calculates the inverse of a square complex matrix.

This procedure is carried out by the LAPACK methods: zgetrf and zgetri. This means that the matrix will be factorized using LU decomposition first, followed by the calculation of the inverse based on:

inv(A)*L = inv(U) for inv(A).

Parameters:

oMatrix a complex matrix

Returns:

the inverse of the complex matrix

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

- libla.h

6.3 LIB_LA::LU_Result Class Reference

LUResult is intended to hold the return values of the Clapack LU decomposition methods.

`#include <libla.h>`

Public Member Functions

- LU_Result ()

  Constructor of a new result object (all result variables are NULL).

- ~LU_Result ()

  Destructor deletes all non-NULL matrices.

Public Attributes

- DoubleMatrix * L

  L is a lower triangular matrix.

- int nInfo

  Info represents status information about the LU factorization its value is to be interpreted as:.

- IntMatrix * P

  P is a permutation matrix representing row permutations.

- IntMatrix * Q

  Q is a permutation matrix representing column permutations.

- DoubleMatrix * U

  U is an upper triangular matrix.

6.3.1 Detailed Description

LUResult is intended to hold the return values of the Clapack LU decomposition methods.

This class will hold the result from the methods LIB_LA::LibLA::getLU and LIB_LA::LibLA::getLUwithFullPivoting.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 LIB_LA::LU_Result::LU_Result () [inline]

Constructor of a new result object (all result variables are NULL).

6.3.2.2 LIB_LA::LU_Result::~LU_Result () [inline]

Destructor deletes all non-NULL matrices.

6.3.3 Member Data Documentation

6.3.3.1 DoubleMatrix* LIB_LA::LU_Result::L

L is a lower triangular matrix.

6.3.3.2 int LIB_LA::LU_Result::nInfo

Info represents status information about the LU factorization its value is to be interpreted as:.

- 0: successful exit
- < 0: if INFO = -i, the i-th argument had an illegal value
- > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

6.3.3.3 IntMatrix* LIB_LA::LU_Result::P

P is a permutation matrix representing row permutations.

6.3.3.4 IntMatrix* LIB_LA::LU_Result::Q

Q is a permutation matrix representing column permutations.

Remarks:

and is only available after a call to LIB_LA::LibLA::getLUwithFullPivoting and NULL otherwise

6.3.3.5 DoubleMatrix* LIB_LA::LU_Result::U

U is an upper triangular matrix.

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

- libla.h

6.4 LIB_LA::Matrix< T > Class Template Reference

LIB_LA::Matrix is the matrix class used by LIB_LA::LibLA and LIB_STRUTURAL::LibStructural.

`#include <matrix.h>`

Public Types

- typedef T _ElementType
  the element type for this matrix, will be real, LIB_LA::Complex or integer.

Public Member Functions

- T ** get2DMatrix (int &nRows, int &nCols)
  returns a 2D data array
- T * getArray ()
  returns a pointer to the underlying 1D array
- T * getCopy (bool transpose=false)
  returns a copy of the data, optionally transposing it
- virtual Matrix< T > * getTranspose ()
  creates a new matrix holding the transpose
- void initializeFrom2DMatrix (T **&oRawData, int nRows, int nCols)
  initializes the matrix from 2D data
- void initializeFromConst2DMatrix (const T **oRawData, int nRows, int nCols)
  initializes the matrix from 2D const data
- Matrix (const T **oRawData, int nRows, int nCols)
  constructs a matrix from 2D const data
- Matrix (T **&oRawData, int nRows, int nCols)
  constructs a matrix from 2D data
- Matrix (T *&oRawData, int nRows, int nCols, bool transpose=true)
  Constructor taking a matrix mapped to a vector and reconstructing the 2D form.
- Matrix (const Matrix< T > &src)
  Copy constructor.
- Matrix (unsigned int rows=0, unsigned int cols=0)
  Creates a new matrix with the given numbers of rows and columns.
- virtual unsigned int numCols () const
  returns the number of columns

- virtual unsigned int numRows () const

  returns the number of rows

- virtual const _ElementType & operator() (const unsigned int &row, const unsigned int &col) const

  returns the selected matrix element (const)

- virtual _ElementType & operator() (const unsigned int &row, const unsigned int &col)

  returns the selected matrix element

- virtual Matrix< T > & operator= (const T &value)

  scalar assignment operator

- virtual Matrix< T > & operator= (const Matrix< T > &rhs)

  assignment operator

- virtual const T * operator[ ] (unsigned int row) const

  returns the selected row

- virtual T * operator[ ] (unsigned int row)

  returns the selected row

- virtual void resize (unsigned int rows, unsigned int cols)

  resizes the matrix to the given number of rows and columns

- virtual unsigned int size () const

  returns the size of the matrix

- virtual void swapCols (unsigned int col1, unsigned int col2)

  swaps the given columns

- virtual void swapRows (unsigned int row1, unsigned int row2)

  swaps the given rows

- virtual ~Matrix ()

  virtual destructor

Protected Attributes

- T * _Array
- unsigned int _Cols
- unsigned int _Rows

6.4.1 Detailed Description

template<class T> class LIB_LA::Matrix< T >

LIB_LA::Matrix is the matrix class used by LIB_LA::LibLA and LIB_STRUTURAL::LibStructural.

This class implements a template to hold real, LIB_LA::Complex and integer matrices. It also implements basic operations on matrices.

Examples:

examples/cpp/loadstoichiometry.cpp, and examples/cpp/printmatrices.cpp.

6.4.2 Member Typedef Documentation

6.4.2.1 template<class T> typedef T LIB_LA::Matrix< T >::_ElementType

the element type for this matrix, will be real, LIB_LA::Complex or integer.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 template<class T> LIB_LA::Matrix< T >::Matrix (unsigned int *rows* = 0, unsigned int *cols* = 0) [inline]

Creates a new matrix with the given numbers of rows and columns.

6.4.3.2 template<class T> LIB_LA::Matrix< T >::Matrix (const Matrix< T > & *src*) [inline]

Copy constructor.

6.4.3.3 template<class T> LIB_LA::Matrix< T >::Matrix (T *& *oRawData*, int *nRows*, int *nCols*, bool *transpose* = true) [inline]

Constructor taking a matrix mapped to a vector and reconstructing the 2D form.

6.4.3.4 template<class T> LIB_LA::Matrix< T >::Matrix (T **& *oRawData*, int *nRows*, int *nCols*) [inline]

constructs a matrix from 2D data

6.4.3.5 template<class T> LIB_LA::Matrix< T >::Matrix (const T ** *oRawData*, int *nRows*, int *nCols*) [inline]

constructs a matrix from 2D const data

6.4.3.6 template<class T> virtual LIB_LA::Matrix< T >::~Matrix () [inline, virtual]

virtual destructor

6.4.4 Member Function Documentation

6.4.4.1 template<class T> T** LIB_LA::Matrix< T >::get2DMatrix (int & *nRows*, int & *nCols*)

returns a 2D data array

6.4.4.2 template<class T> T* LIB_LA::Matrix< T >::getArray () [inline]

returns a pointer to the underlying 1D array

6.4.4.3 template<class T> T* LIB_LA::Matrix< T >::getCopy (bool *transpose* = false) [inline]

returns a copy of the data, optionally transposing it

6.4.4.4 template<class T> virtual Matrix<T>* LIB_LA::Matrix< T >::getTranspose () [inline, virtual]

creates a new matrix holding the transpose

6.4.4.5 template<class T> void LIB_LA::Matrix< T >::initializeFrom2DMatrix (T **& *oRawData*, int *nRows*, int *nCols*)

initializes the matrix from 2D data

6.4.4.6 template<class T> void LIB_LA::Matrix< T >::initializeFromConst2DMatrix (const T ** *oRawData*, int *nRows*, int *nCols*)

initializes the matrix from 2D const data

6.4.4.7 template<class T> virtual unsigned int LIB_LA::Matrix< T >::numCols () const [inline, virtual]

returns the number of columns

Examples:

examples/cpp/printmatrices.cpp.

6.4.4.8 template<class T> virtual unsigned int LIB_LA::Matrix< T >::numRows () const [inline, virtual]

returns the number of rows

Examples:

examples/cpp/printmatrices.cpp.

6.4.4.9 template<class T> virtual const _ElementType& LIB_LA::Matrix< T >::operator() (const unsigned int & *row*, const unsigned int & *col*) const [inline, virtual]

returns the selected matrix element (const)

6.4.4.10 template<class T> virtual _ElementType& LIB_LA::Matrix< T >::operator() (const unsigned int & *row*, const unsigned int & *col*) [inline, virtual]

returns the selected matrix element

6.4.4.11 template<class T> virtual Matrix<T>& LIB_LA::Matrix< T >::operator= (const T & *value*) [inline, virtual]

scalar assignment operator

6.4.4.12 template<class T> virtual Matrix<T>& LIB_LA::Matrix< T >::operator= (const Matrix< T > & *rhs*) [inline, virtual]

assignment operator

6.4.4.13]

template<class T> virtual const T* **LIB_LA::Matrix**< T >::operator[ ] (unsigned int *row*) const [inline, virtual]

returns the selected row

6.4.4.14]

template<class T> virtual T* **LIB_LA::Matrix**< T >::operator[ ] (unsigned int *row*) [inline, virtual]

returns the selected row

6.4.4.15 template<class T> virtual void LIB_LA::Matrix< T >::resize (unsigned int *rows*, unsigned int *cols*) [inline, virtual]

resizes the matrix to the given number of rows and columns

6.4.4.16 template<class T> virtual unsigned int LIB_LA::Matrix< T >::size () const [inline, virtual]

returns the size of the matrix

6.4.4.17 template<class T> virtual void LIB_LA::Matrix< T >::swapCols (unsigned int *col1*, unsigned int *col2*) [inline, virtual]

swaps the given columns

6.4.4.18 template<class T> virtual void LIB_LA::Matrix< T >::swapRows (unsigned int *row1*, unsigned int *row2*) [inline, virtual]

swaps the given rows

6.4.5 Member Data Documentation

6.4.5.1 template<class T> T* LIB_LA::Matrix< T >::_Array [protected]

6.4.5.2 template<class T> unsigned int LIB_LA::Matrix< T >::_Cols [protected]

6.4.5.3 template<class T> unsigned int LIB_LA::Matrix< T >::_Rows [protected]

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

- matrix.h

6.5 LIB_STRUCTURAL::LibStructural Class Reference

Entrypoint for the C++ API of the Structural Analysis Library.

`#include <libstructural.h>`

Public Types

- typedef LIB_LA::Matrix< LIB_LA::Complex > ComplexMatrix
- typedef LIB_LA::Matrix< double > DoubleMatrix
- typedef LIB_LA::Matrix< int > IntMatrix

Public Member Functions

- std::string analyzeWithFullyPivotedLU ()

  Uses fully pivoted LU Decomposition for Conservation analysis.

- std::string analyzeWithFullyPivotedLUwithTests ()

  Uses fully pivoted LU Decomposition for Conservation analysis.

- std::string analyzeWithLU ()

  Uses LU Decomposition for Conservation analysis.

- std::string analyzeWithLUandRunTests ()

  Uses LU Decomposition for Conservation analysis.

- std::string analyzeWithQR ()

  Uses QR factorization for structural analysis.

- DoubleMatrix * getColumnReorderedNrMatrix ()

  Returns the Nr Matrix repartitioned into NIC (independent columns) and NDC (dependent columns).

- void getColumnReorderedNrMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the Nr Matrix row and column labels (repartitioned into NIC and NDC).

- std::vector< std::string > getConservedLaws ()

  Returns algebraic expressions for conserved cycles.

- std::vector< double > getConservedSums ()

  Returns values for conservation laws using the current initial conditions.

- std::vector< std::string > getDependentReactionIds ()

  Returns the list of dependent reactions.

- std::vector< std::string > getDependentSpecies ()

  Returns the list of dependent species.

- std::vector< std::string > getDependentSpeciesNamesList ()

Returns the actual names of the dependent species.

- DoubleMatrix * getFullyReorderedStoichiometryMatrix ()

  Returns the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

- void getFullyReorderedStoichiometryMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the row and column labels for the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

- DoubleMatrix * getGammaMatrix ()

  Returns Gamma, the conservation law array.

- void getGammaMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the row and column labels for Gamma, the conservation law array.

- std::vector< std::string > getIndependentReactionIds ()

  Returns the list of independent reactions.

- std::vector< std::string > getIndependentSpecies ()

  Returns the list of independent species.

- std::vector< std::string > getIndependentSpeciesNamesList ()

  Returns the actual names of the independent species.

- std::vector< std::pair< std::string, double > > getInitialConditions ()

  Returns Initial Conditions used in the model.

- DoubleMatrix * getK0Matrix ()

  Returns the K0 Matrix.

- void getK0MatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the K0 Matrix row and column labels.

- DoubleMatrix * getKMatrix ()

  Returns the K matrix (right nullspace of Nr).

- void getKMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the K matrix row and column labels.

- DoubleMatrix * getL0Matrix ()

  Returns the L0 Matrix.

- void getL0MatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the L0 Matrix row and column labels.

- DoubleMatrix * getLinkMatrix ()

  Returns L, the Link Matrix, left nullspace (aka nullspace of the transpose Nr).

- void getLinkMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)
  Returns the row and column labels for the Link Matrix, L.
- std::string getModelName ()
  Returns the name of the model.
- DoubleMatrix * getN0Matrix ()
  Returns the N0 Matrix.
- void getN0MatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)
  Returns the N0 Matrix row and column labels.
- DoubleMatrix * getNDCMatrix ()
  Returns the NDC Matrix (the set of linearly dependent columns of Nr).
- void getNDCMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)
  Returns the NDC Matrix row and column labels.
- DoubleMatrix * getNICMatrix ()
  Returns the NIC Matrix (the set of linearly independent columns of Nr).
- void getNICMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)
  Returns the NIC Matrix row and column labels.
- double getNmatrixSparsity ()
  Returns the number of nonzero values in Stoichiometry matrix.
- DoubleMatrix * getNrMatrix ()
  Returns the Nr Matrix.
- void getNrMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)
  Returns the Nr Matrix row and column labels.
- int getNumDepReactions ()
  Returns the number of dependent reactions.
- int getNumDepSpecies ()
  Returns the number of dependent species.
- int getNumIndReactions ()
  Returns the number of independent reactions.
- int getNumIndSpecies ()
  Returns the number of independent species.
- int getNumReactions ()
  Returns the total number of reactions.
- int getNumSpecies ()

Returns the total number of species.

- int getRank ()

  Returns rank of stoichiometry matrix.

- std::vector< std::string > getReactions ()

  Returns the list of Reactions.

- std::vector< std::string > getReactionsNamesList ()

  Returns actual names of the Reactions.

- std::vector< std::string > getReorderedReactions ()

  Returns the reordered list of reactions.

- std::vector< std::string > getReorderedSpecies ()

  Returns the reordered list of molecular species. (choosing the SBML Id if possible).

- std::vector< std::string > getReorderedSpeciesNamesList ()

  Returns the reordered list of molecular species. (choosing the SBML Name if possible).

- DoubleMatrix * getReorderedStoichiometryMatrix ()

  Returns the reordered stoichiometry matrix (row reordered stoichiometry matrix, columns are not reordered!).

- void getReorderedStoichiometryMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix).

- std::vector< std::string > getSpecies ()

  Returns the unordered list of species Ids.

- DoubleMatrix * getStoichiometryMatrix ()

  Returns the original, unaltered stoichiometry matrix.

- void getStoichiometryMatrixLabels (std::vector< std::string > &oRows, std::vector< std::string > &oCols)

  Returns the row and column labels for the original and unaltered stoichiometry matrix.

- std::string getTestDetails ()

  Return Return Details about validation tests.

- double getTolerance ()

  Returns the currently used tolerance.

- LibStructural ()

  Constructor of a new instance of LibStructural.

- void loadReactionNames (std::vector< std::string > &reactionNames)

  Load reaction names.

- std::string loadSBML (std::string sSBML)
  Load a SBML model.
- std::string loadSBMLFromFile (std::string sFileName)
  Load a SBML model from the specified file.
- std::string loadSBMLwithTests (std::string sSBML)
  Load an SBML model into the library and carry out tests using the internal test suite.
- void loadSpecies (std::vector< std::string > &speciesNames, std::vector< double > &speciesValues)
  Load species names and initial values.
- void loadStoichiometryMatrix (DoubleMatrix &oMatrix)
  Load a new stoichiometry matrix.
- void setTolerance (double dTolerance)
  Set user specified tolerance.
- std::vector< std::string > validateStructuralMatrices ()
  Validates structural matrices.

Static Public Member Functions

- static LibStructural * getInstance ()
  static method to get an instance of LibStructural (allows use as singleton)

6.5.1 Detailed Description

Entrypoint for the C++ API of the Structural Analysis Library.

LIB_STRUCTURAL::LibStructural represents the main class for all structural analyses on either http://sbml.org/ models or directly on a provided stoichiometry matrix.

The model can be either analyzed by employing QR factorization with householder reflections, or with LU(P) factorization. The QR factorization is the superior method.

Further Information

Vallabhajosyula RR, Chickarmane V, Sauro HM. Conservation analysis of large biochemical networks Bioinformatics, 2005 Nov 29 http://bioinformatics.oxfordjournals.org/cgi/content/abstract/bti800v1

Examples

For examples on how to use the library see LIB_STRUCTURAL::LibStructural::loadSBML and LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix

Examples:

examples/cpp/loadsbmlfromfile.cpp, examples/cpp/loadstoichiometry.cpp, and examples/cpp/printmatrices.cpp.

6.5.2 Member Typedef Documentation

6.5.2.1 typedef LIB_LA::Matrix< LIB_LA::Complex > LIB_STRUCTURAL::LibStructural::ComplexMatrix

6.5.2.2 typedef LIB_LA::Matrix< double > LIB_STRUCTURAL::LibStructural::DoubleMatrix

6.5.2.3 typedef LIB_LA::Matrix< int > LIB_STRUCTURAL::LibStructural::IntMatrix

6.5.3 Constructor & Destructor Documentation

6.5.3.1 LIB_STRUCTURAL::LibStructural::LibStructural () [inline]

Constructor of a new instance of LibStructural.

6.5.4 Member Function Documentation

6.5.4.1 std::string LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU ()

Uses fully pivoted LU Decomposition for Conservation analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix or LIB_STRUCTURAL::LibStructural::loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LIB_STRUCTURAL::LibStructural::analyzeWithQR,
- LIB_STRUCTURAL::LibStructural::analyzeWithLU,
- LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests,
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU or
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests

Returns:

a result string with information about the analysis process

6.5.4.2 std::string LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests ()

Uses fully pivoted LU Decomposition for Conservation analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix or LIB_STRUCTURAL::LibStructural::loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LIB_STRUCTURAL::LibStructural::analyzeWithQR,
- LIB_STRUCTURAL::LibStructural::analyzeWithLU,
- LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests,
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU or
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests

This method additionally performs the integrated test suite and returns those results.

Returns:

a result string with information about the analysis process

6.5.4.3 std::string LIB_STRUCTURAL::LibStructural::analyzeWithLU ()

Uses LU Decomposition for Conservation analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix or LIB_STRUCTURAL::LibStructural::loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LIB_STRUCTURAL::LibStructural::analyzeWithQR,
- LIB_STRUCTURAL::LibStructural::analyzeWithLU,
- LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests,
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU or
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests

Returns:

a result string with information about the analysis process

6.5.4.4 std::string LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests ()

Uses LU Decomposition for Conservation analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix or LIB_STRUCTURAL::LibStructural::loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LIB_STRUCTURAL::LibStructural::analyzeWithQR,
- LIB_STRUCTURAL::LibStructural::analyzeWithLU,
- LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests,
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU or

- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests

This method additionally performs the integrated test suite and returns those results.

Returns:

a result string with information about the analysis process

6.5.4.5 std::string LIB_STRUCTURAL::LibStructural::analyzeWithQR ()

Uses QR factorization for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix or LIB_STRUCTURAL::LibStructural::loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LIB_STRUCTURAL::LibStructural::analyzeWithQR,
- LIB_STRUCTURAL::LibStructural::analyzeWithLU,
- LIB_STRUCTURAL::LibStructural::analyzeWithLUandRunTests,
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLU or
- LIB_STRUCTURAL::LibStructural::analyzeWithFullyPivotedLUwithTests

Remarks:

This is the prefered method for structural analysis.

Returns:

a result string with information about the analysis process

Examples:

examples/cpp/loadstoichiometry.cpp.

6.5.4.6 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getColumnReorderedNrMatrix ()

Returns the Nr Matrix repartitioned into NIC (independent columns) and NDC (dependent columns).

6.5.4.7 void LIB_STRUCTURAL::LibStructural::getColumnReorderedNrMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the Nr Matrix row and column labels (repartitioned into NIC and NDC).

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.8 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getConservedLaws ()

Returns algebraic expressions for conserved cycles.

6.5.4.9 std::vector< double > LIB_STRUCTURAL::LibStructural::getConservedSums ()

Returns values for conservation laws using the current initial conditions.

6.5.4.10 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getDependentReactionIds ()

Returns the list of dependent reactions.

6.5.4.11 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getDependentSpecies ()

Returns the list of dependent species.

6.5.4.12 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getDependentSpeciesNamesList ()

Returns the actual names of the dependent species.

6.5.4.13 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getFullyReorderedStoichiometryMatrix ()

Returns the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.14 void LIB_STRUCTURAL::LibStructural::getFullyReorderedStoichiometryMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the row and column labels for the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.15 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getGammaMatrix ()

Returns Gamma, the conservation law array.

Each row represents a single conservation law where the column indicate the participating molecular species. The number of rows is therefore equal to the number of conservation laws. Columns are ordered according to the rows in the reordered stoichiometry matrix, see LIB_STRUCTURAL::LibStructural::getReorderedSpeciesId and LIB_STRUCTURAL::LibStructural::getReorderedStoichiometryMatrix.

6.5.4.16 void LIB_STRUCTURAL::LibStructural::getGammaMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the row and column labels for Gamma, the conservation law array.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.17 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getIndependentReactionIds ()

Returns the list of independent reactions.

6.5.4.18 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getIndependentSpecies ()

Returns the list of independent species.

6.5.4.19 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getIndependentSpeciesNamesList ()

Returns the actual names of the independent species.

6.5.4.20 std::vector< std::pair <std::string, double> > LIB_STRUCTURAL::LibStructural::getInitialConditions ()

Returns Initial Conditions used in the model.

6.5.4.21 static LibStructural* LIB_STRUCTURAL::LibStructural::getInstance () [static]

static method to get an instance of LibStructural (allows use as singleton)

6.5.4.22 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getK0Matrix ()

Returns the K0 Matrix.

K0 is defined such that K0 = -(NIC)^-1 NDC, or equivalently, [NDC NIC][I K0]' = 0 where [NDC NIC] = Nr

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.23 void LIB_STRUCTURAL::LibStructural::getK0MatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the K0 Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.24 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getKMatrix ()

Returns the K matrix (right nullspace of Nr).

The K matrix has the structure, [I K0]'

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.25 void LIB_STRUCTURAL::LibStructural::getKMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the K matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.26 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getL0Matrix ()

Returns the L0 Matrix.

L0 is defined such that L0 Nr = N0. L0 forms part of the link matrix, L. N0 is the set of linear dependent rows from the lower portion of the reordered stoichiometry matrix.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.27 void LIB_STRUCTURAL::LibStructural::getL0MatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the L0 Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.28 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getLinkMatrix ()

Returns L, the Link Matrix, left nullspace (aka nullspace of the transpose Nr).

L will have the structure, [I L0]', such that L Nr = N

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.29 void LIB_STRUCTURAL::LibStructural::getLinkMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the row and column labels for the Link Matrix, L.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.30 std::string LIB_STRUCTURAL::LibStructural::getModelName ()

Returns the name of the model.

Returns the name of the model if SBML model has Name-tag, otherwise it returns the SBML id. If only a stoichiometry matrix was loaded 'untitled' will be returned.

6.5.4.31 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getN0Matrix ()

Returns the N0 Matrix.

The N0 matrix is the set of linearly dependent rows of N where L0 Nr = N0.

6.5.4.32 void LIB_STRUCTURAL::LibStructural::getN0MatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the N0 Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.33 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getNDCMatrix ()

Returns the NDC Matrix (the set of linearly dependent columns of Nr).

6.5.4.34 void LIB_STRUCTURAL::LibStructural::getNDCMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the NDC Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.35 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getNICMatrix ()

Returns the NIC Matrix (the set of linearly independent columns of Nr).

6.5.4.36 void LIB_STRUCTURAL::LibStructural::getNICMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the NIC Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.37 double LIB_STRUCTURAL::LibStructural::getNmatrixSparsity ()

Returns the number of nonzero values in Stoichiometry matrix.

6.5.4.38 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getNrMatrix ()

Returns the Nr Matrix.

The rows of the Nr matrix will be linearly independent.

6.5.4.39 void LIB_STRUCTURAL::LibStructural::getNrMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the Nr Matrix row and column labels.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

6.5.4.40 int LIB_STRUCTURAL::LibStructural::getNumDepReactions ()

Returns the number of dependent reactions.

6.5.4.41 int LIB_STRUCTURAL::LibStructural::getNumDepSpecies ()

Returns the number of dependent species.

6.5.4.42 int LIB_STRUCTURAL::LibStructural::getNumIndReactions ()

Returns the number of independent reactions.

6.5.4.43 int LIB_STRUCTURAL::LibStructural::getNumIndSpecies ()

Returns the number of independent species.

6.5.4.44 int LIB_STRUCTURAL::LibStructural::getNumReactions ()

Returns the total number of reactions.

6.5.4.45 int LIB_STRUCTURAL::LibStructural::getNumSpecies ()

Returns the total number of species.

6.5.4.46 int LIB_STRUCTURAL::LibStructural::getRank ()

Returns rank of stoichiometry matrix.

6.5.4.47 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getReactions ()

Returns the list of Reactions.

6.5.4.48 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getReactionsNamesList ()

Returns actual names of the Reactions.

6.5.4.49 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getReorderedReactions ()

Returns the reordered list of reactions.

6.5.4.50 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getReorderedSpecies ()

Returns the reordered list of molecular species. (choosing the SBML Id if possible).

6.5.4.51 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getReorderedSpeciesNamesList ()

Returns the reordered list of molecular species. (choosing the SBML Name if possible).

6.5.4.52 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getReorderedStoichiometryMatrix ()

Returns the reordered stoichiometry matrix (row reordered stoichiometry matrix, columns are not reordered!).

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.53 void LIB_STRUCTURAL::LibStructural::getReorderedStoichiometryMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix).

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.54 std::vector< std::string > LIB_STRUCTURAL::LibStructural::getSpecies ()

Returns the unordered list of species Ids.

6.5.4.55 DoubleMatrix* LIB_STRUCTURAL::LibStructural::getStoichiometryMatrix ()

Returns the original, unaltered stoichiometry matrix.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.56 void LIB_STRUCTURAL::LibStructural::getStoichiometryMatrixLabels (std::vector< std::string > & *oRows*, std::vector< std::string > & *oCols*)

Returns the row and column labels for the original and unaltered stoichiometry matrix.

Parameters:

oRows a string vector that will be overwritten to hold the row labels

oCols a string vector that will be overwritten to hold the column labels.

Examples:

examples/cpp/printmatrices.cpp.

6.5.4.57 std::string LIB_STRUCTURAL::LibStructural::getTestDetails ()

Return Return Details about validation tests.

Examples:

examples/cpp/loadsbmlfromfile.cpp, examples/cpp/loadstoichiometry.cpp, and examples/cpp/printmatrices.cpp.

6.5.4.58 double LIB_STRUCTURAL::LibStructural::getTolerance () [inline]

Returns the currently used tolerance.

This function returns the tolerance currently used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered zero and will be neglected.

6.5.4.59 void LIB_STRUCTURAL::LibStructural::loadReactionNames (std::vector< std::string > & *reactionNames*)

Load reaction names.

This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the reaction names (ids).

Remarks:

This method should only be called after LibStructural_loadStoichiometryMatrix

Parameters:

reactionNames a vector of reaction names (ids)

Examples:

examples/cpp/loadstoichiometry.cpp.

6.5.4.60 std::string LIB_STRUCTURAL::LibStructural::loadSBML (std::string *sSBML*)

Load a SBML model.

Parameters:

sSBML the SBML string to load

Returns:

information about the loaded model

6.5.4.61 std::string LIB_STRUCTURAL::LibStructural::loadSBMLFromFile (std::string *sFileName*)

Load a SBML model from the specified file.

Parameters:

sFileName a file name to a SBML file to load

Returns:

information about the loaded model

Examples:

examples/cpp/loadsbmlfromfile.cpp, and examples/cpp/printmatrices.cpp.

6.5.4.62 std::string LIB_STRUCTURAL::LibStructural::loadSBMLwithTests (std::string *sSBML*)

Load an SBML model into the library and carry out tests using the internal test suite.

Parameters:

sSBML the SBML file to load

Returns:

information about the loaded model and results of the internal test suite

6.5.4.63 void LIB_STRUCTURAL::LibStructural::loadSpecies (std::vector< std::string > & *speciesNames*, std::vector< double > & *speciesValues*)

Load species names and initial values.

This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the species names (ids). It is also possible to provide an initial condition for each of the species. This will be used when calculating the conserved sums.

Remarks:

This method should only be called after LibStructural_loadStoichiometryMatrix

Parameters:

speciesNames a vector of species names (ids) to load

speciesValues a vector of initial concentrations

Examples:

examples/cpp/loadstoichiometry.cpp.

6.5.4.64 void LIB_STRUCTURAL::LibStructural::loadStoichiometryMatrix (DoubleMatrix & *oMatrix*)

Load a new stoichiometry matrix.

Loads the stoichiometry matrix into the library. To analyze the stoichiometry call one of the following:

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

Remarks:

if matrix labels are needed it is recommended to call LIB_STRUCTURAL::LibStructural::loadSpecies and LIB_STRUCTURAL::LibStructural::loadReactionNames after a call to this method.

Parameters:

oMatrix the stoichiometry matrix to load

Examples:

examples/cpp/loadstoichiometry.cpp.

6.5.4.65 void LIB_STRUCTURAL::LibStructural::setTolerance (double *dTolerance*)

Set user specified tolerance.

This function sets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

Parameters:

dTolerance Sets the tolerance used by the library to determine a value close to zero

6.5.4.66 std::vector< std::string > LIB_STRUCTURAL::LibStructural::validateStructuralMatrices ()

Validates structural matrices.

Calling this method will run the internal test suite against the structural matrices those tests include:

- Test 1 : Gamma∗N = 0 (Zero matrix)
- Test 2 : Rank(N) using SVD (5) is same as m0 (5)
- Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
- Test 4 : Rank(NR) using QR (5) is same as m0 (5)
- Test 5 : L0 obtained with QR matches Q21∗inv(Q11)
- Test 6 : N∗K = 0 (Zero matrix)

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

- libstructural.h

Chapter 7

File Documentation

7.1 complex.h File Reference

```
#include <iosfwd>
```

Namespaces

- namespace LIB_LA

Classes

- class LIB_LA::Complex

  LIB_LA::Complex is the complex class used by LIB_LA::LibLA.

Functions

- std::ostream & LIB_LA::operator<< (std::ostream &os, const Complex &complex)

  overload that allows to print a complex number on a std::stream

7.2 libla.h File Reference

All definitions needed for the LibLA (Linear Algebra) library.

```
#include "libutil.h"
#include <vector>
#include "matrix.h"
#include "complex.h"
```

Namespaces

- namespace LIB_LA

Classes

- class LIB_LA::LibLA

  The LIB_LA::LibLA class represents the entry point to a variety of useful functionality operating on double and complex matrices.

- class LIB_LA::LU_Result

  LUResult is intended to hold the return values of the Clapack LU decomposition methods.

Functions

- LIB_EXTERN void LibLA_freeMatrix (void **matrix, int numRows)

  Frees a matrix previously allocated by this library.

- LIB_EXTERN void LibLA_freeVector (void *vector)

  Frees a vector previously allocated by this library.

- LIB_EXTERN int LibLA_fullyPivotedGaussJordan (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols, int **oRowPivots, int *nRowLength, int **oColPivots, int *nColLength)

  This method calculates the fully pivoted Gauss Jordan form of the given matrix.

- LIB_EXTERN int LibLA_gaussJordan (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols, int **oPivots, int *nLength)

  This method calculates the Gauss Jordan or row echelon form of the given matrix.

- LIB_EXTERN int LibLA_getEigenValues (double **inMatrix, int numRows, int numCols, double **outReal, double **outImag, int *outLength)

  Calculates the eigen-values of a square real matrix.

- LIB_EXTERN int LibLA_getLU (double **inMatrix, int numRows, int numCols, double ***outL, int *outLRows, int *outLCols, double ***outU, int *outURows, int *outUCols, int ***outP, int *outPRows, int *outPCols, int *info)

  This function computes the LU factorization of the given real M-by-N matrix A.

- LIB_EXTERN int LibLA_getLUwithFullPivoting (double **inMatrix, int numRows, int numCols, double ***outL, int *outLRows, int *outLCols, double ***outU, int *outURows, int *outUCols, int ***outP, int *outPRows, int *outPCols, int ***outQ, int *outQRows, int *outQCols, int *info)

  This function computes the LU factorization of the given real N-by-N matrix A using complete pivoting (with row and column interchanges).

- LIB_EXTERN int LibLA_getQR (double **inMatrix, int numRows, int numCols, double ***outQ, int *outQRows, int *outQCols, double ***outR, int *outRRows, int *outRCols)

  This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

- LIB_EXTERN int LibLA_getQRWithPivot (double **inMatrix, int numRows, int numCols, double ***outQ, int *outQRows, int *outQCols, double ***outR, int *outRRows, int *outRCols, double ***outP, int *outPRows, int *outPCols)

  This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

- LIB_EXTERN int LibLA_getRank (double **inMatrix, int numRows, int numCols)

  This method computes the rank of the given matrix.

- LIB_EXTERN int LibLA_getSingularValsBySVD (double **inMatrix, int numRows, int numCols, double **outSingularVals, int *outLength)

  This method performs the Singular Value Decomposition of the given real matrix, returning only the singular values.

- LIB_EXTERN double LibLA_getTolerance ()

  Returns the currently used tolerance.

- LIB_EXTERN int LibLA_inverse (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols)

  This function calculates the inverse of a square real matrix.

- LIB_EXTERN int LibLA_leftNullspace (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols)

  This function calculates the left null space of a given real matrix.

- LIB_EXTERN int LibLA_rightNullspace (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols)

  This function calculates the right null space of a given real matrix.

- LIB_EXTERN int LibLA_scaledLeftNullspace (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols)

  This function calculates the scaled left null space of a given real matrix.

- LIB_EXTERN int LibLA_scaledRightNullspace (double **inMatrix, int numRows, int numCols, double ***outMatrix, int *outRows, int *outCols)

  This function calculates the scaled right null space of a given real matrix.

- LIB_EXTERN void LibLA_setTolerance (const double value)

  Set user specified tolerance.

- LIB_EXTERN int LibLA_ZgetEigenValues (double **inMatrixReal, double **inMatrixImag, int numRows, int numCols, double **outReal, double **outImag, int *outLength)

Calculates the eigen-values of a square complex matrix.

- LIB_EXTERN int LibLA_Zinverse (double **inMatrixReal, double **inMatrixImag, int numRows, int numCols, double ***outMatrixReal, double **outMatrixImag, int *outRows, int *outCols)

  This function calculates the inverse of a square complex matrix.

Variables

- BEGIN_C_DECLS
- END_C_DECLS

7.2.1 Detailed Description

All definitions needed for the LibLA (Linear Algebra) library.

The current scope of the library encompasses matrix factorizations (QR and LU factorization) as well as commonly needed matrix operations, such as calculating the inverse of a matrix, computing eigen values and singular values as well as the null space of a matrix (both left and right null space) along with a method for the computation of the row echelon or Gauss Jordan form of a matrix.

Author:

Frank T. Bergmann (fbergman@u.washington.edu)
Herbert M. Sauro
Ravishankar Rao Vallabhajosyula (developed a previous version of the sructural analysis code)

7.2.2 Function Documentation

7.2.2.1 LIB_EXTERN void LibLA_freeMatrix (void ** *matrix*, int *numRows*)

Frees a matrix previously allocated by this library.

7.2.2.2 LIB_EXTERN void LibLA_freeVector (void * *vector*)

Frees a vector previously allocated by this library.

7.2.2.3 LIB_EXTERN int LibLA_fullyPivotedGaussJordan (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*, int ** *oRowPivots*, int * *nRowLength*, int ** *oColPivots*, int * *nColLength*)

This method calculates the fully pivoted Gauss Jordan form of the given matrix.

Fully pivoted means, that rows as well as column swaps will be used. These permutations are captured in the integer vectors rowPivots and colPivots.

If no permutations have occurred those vectors will be in ascending form [0, 1, 2, 3]; However if say row one and three would be swapped this vector would look like: [0, 3, 2, 1];

7.2.2.4 LIB_EXTERN int LibLA_gaussJordan (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*, int ** *oPivots*, int * *nLength*)

This method calculates the Gauss Jordan or row echelon form of the given matrix.

Only row swaps are used. These permutations will be returned in the 'pivots' vector.

If no permutations have occurred this vector will be in ascending form [0, 1, 2, 3]; However if say row one and three would be swapped this vector would look like: [0, 3, 2, 1];

7.2.2.5 LIB_EXTERN int LibLA_getEigenValues (double ** *inMatrix*, int *numRows*, int *numCols*, double ** *outReal*, double ** *outImag*, int * *outLength*)

Calculates the eigen-values of a square real matrix.

This function calculates the complex eigenvalues of the given real matrix. The complex vector of eigenvalues will be returned in two real vectors, one for the real and one for the imaginary part.

Parameters:

inMatrix real matrix to calculate the eigen-values for.

numRows the number of rows of the input matrix

numCols the number of columns of the input matrix

outReal pointer to the output array for the eigenvalues (real part)

outImag pointer to the output array for the eigenvalues (imaginary part)

outLength the number of eigenvalues written to outReal and outImag

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred (for example because the caller supplied a non-square matrix)

Remarks:

free outReal and outImag using LibLA_freeVector

7.2.2.6 LIB_EXTERN int LibLA_getLU (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outL*, int * *outLRows*, int * *outLCols*, double *** *outU*, int * *outURows*, int * *outUCols*, int *** *outP*, int * *outPRows*, int * *outPCols*, int * *info*)

This function computes the LU factorization of the given real M-by-N matrix A.

using partial pivoting with row interchanges. This procedure is carried out by the LAPACK method dgetrf.

A is factorized into:

A = P * L * U

Here P is the row permutation matrix.

7.2.2.7 LIB_EXTERN int LibLA_getLUwithFullPivoting (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outL*, int * *outLRows*, int * *outLCols*, double *** *outU*, int * *outURows*, int * *outUCols*, int *** *outP*, int * *outPRows*, int * *outPCols*, int *** *outQ*, int * *outQRows*, int * *outQCols*, int * *info*)

This function computes the LU factorization of the given real N-by-N matrix A using complete pivoting (with row and column interchanges).

This procedure is carried out by the LAPACK method dgetc2.

A is factorized into:

A = P * L * U * Q

Here P and Q are permutation matrices for the rows and columns respectively.

Remarks:

This function supports only square matrices (N-by-N), choose LibLA_getQRWithPivot for a stable method operating on N-by-M matrices.

7.2.2.8 LIB_EXTERN int LibLA_getQR (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outQ*, int * *outQRows*, int * *outQCols*, double *** *outR*, int * *outRRows*, int * *outRCols*)

This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

The LAPACK method dgeqp3 is used followed by an orthonormalization of Q through the use of DORGQR. The factorized form is:

A = Q * R

Parameters:

inMatrix real matrix to factorize

numRows number of rows of the matrix

numCols number of columns of the matrix

outQ pointer to a real matrix where Q will be written

outQRows number of rows of the Q matrix

outQCols number of columns of the Q matrix

outR pointer to a real matrix where R will be written

outRRows number of rows of the R matrix

outRCols number of columns of the R matrix

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Remarks:

free outQ and outR using LibLA_freeMatrix

7.2.2.9 LIB_EXTERN int LibLA_getQRWithPivot (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outQ*, int * *outQRows*, int * *outQCols*, double *** *outR*, int * *outRRows*, int * *outRCols*, double *** *outP*, int * *outPRows*, int * *outPCols*)

This function computes the QR factorization of the given real M-by-N matrix A with column pivoting.

The LAPACK method dgeqp3 is used followed by an orthonormalization of Q through the use of DORGQR. The factorized form is:

$$A = Q * R$$

this method also returns the column pivots used in the P matrix.

Parameters:

inMatrix real matrix to factorize

numRows number of rows of the matrix

numCols number of columns of the matrix

outQ pointer to a real matrix where Q will be written

outQRows number of rows of the Q matrix

outQCols number of columns of the Q matrix

outR pointer to a real matrix where R will be written

outRRows number of rows of the R matrix

outRCols number of columns of the R matrix

outP pointer to a real matrix where P will be written

outPRows number of rows of the P matrix

outPCols number of columns of the P matrix

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Remarks:

free outP, outQ and outR using LibLA_freeMatrix

7.2.2.10 LIB_EXTERN int LibLA_getRank (double ** *inMatrix*, int *numRows*, int *numCols*)

This method computes the rank of the given matrix.

The singular values of the matrix are calculated and the rank is determined by the number of non-zero values.

Note that zero here is defined as any value whose absolute value is bigger than the set tolerance (see LibLA_setTolerance)

7.2.2.11 LIB_EXTERN int LibLA_getSingularValsBySVD (double ** *inMatrix*, int *numRows*, int *numCols*, double ** *outSingularVals*, int * *outLength*)

This method performs the Singular Value Decomposition of the given real matrix, returning only the singular values.

This procedure is carried out by the LAPACK method dgesdd.

Parameters:

inMatrix real matrix

numRows number of rows of the matrix

numCols number of columns of the matrix

outSingularVals pointer to the double array where the singular values will be stored

outLength number of singular values

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Remarks:

free outSingularVals using LibLA_freeVector

7.2.2.12 LIB_EXTERN double LibLA_getTolerance ()

Returns the currently used tolerance.

This function returns the tolerance currently used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered zero and will be neglected.

7.2.2.13 LIB_EXTERN int LibLA_inverse (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*)

This function calculates the inverse of a square real matrix.

This procedure is carried out by the LAPACK methods dgetrf and dgetri. This means that the matrix will be factorized using LU decomposition first, followed by the calculation of the inverse based on:

inv(A)*L = inv(U) for inv(A).

7.2.2.14 LIB_EXTERN int LibLA_leftNullspace (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*)

This function calculates the left null space of a given real matrix.

That is:

null(A)*A = 0

Remarks:

This function is equivalent to returning the right null space of the transposed matrix. See LibLA_rightNullspace

7.2.2.15 LIB_EXTERN int LibLA_rightNullspace (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*)

This function calculates the right null space of a given real matrix.

That is:

A∗null(A) = 0

In order to calculate the (right) null space, we first calculate the full singular value decomposition (employing dgesdd) of the matrix:

[U,S,V] = svd(A');

then calculate the rank:

r = rank(A)

and finally return the last columns of the U matrix (r+1...n) as the null space matrix.

7.2.2.16 LIB_EXTERN int LibLA_scaledLeftNullspace (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*)

This function calculates the scaled left null space of a given real matrix.

This function is equivalent to calling LibLA_leftNullspace however the resulting matrix will be scaled (employing Gauss Jordan factorization) to yield whole numbered entries wherever possible.

7.2.2.17 LIB_EXTERN int LibLA_scaledRightNullspace (double ** *inMatrix*, int *numRows*, int *numCols*, double *** *outMatrix*, int * *outRows*, int * *outCols*)

This function calculates the scaled right null space of a given real matrix.

This function is equivalent to calling LibLA_rightNullspace however the resulting matrix will be scaled (employing Gauss Jordan factorization) to yield whole numbered entries wherever possible.

7.2.2.18 LIB_EXTERN void LibLA_setTolerance (const double *value*)

Set user specified tolerance.

This function sets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

Parameters:

value Sets the tolerance used by the library to determine a value close to zero

7.2.2.19 LIB_EXTERN int LibLA_ZgetEigenValues (double ** *inMatrixReal*, double ** *inMatrixImag*, int *numRows*, int *numCols*, double ** *outReal*, double ** *outImag*, int * *outLength*)

Calculates the eigen-values of a square complex matrix.

This function calculates the complex eigenvalues of the given complex matrix. The input matrix should be broken up into two matrices representing the real and imaginary parts respectively. The complex vector of eigenvalues will be returned in two real vectors, one for the real and one for the imaginary part.

Parameters:

inMatrixReal real part of the complex matrix to calculate the eigen-values for.

inMatrixImag imaginary part of the complex matrix to calculate the eigen-values for

numRows the number of rows of the input matrix

numCols the number of columns of the input matrix

outReal pointer to the output array for the eigenvalues (real part)

outImag pointer to the output array for the eigenvalues (imaginary part)

outLength the number of eigenvalues written to outReal and outImag

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred (for example non square matrix)

Remarks:

free outReal and outImag using LibLA_freeVector

7.2.2.20 LIB_EXTERN int LibLA_Zinverse (double ** *inMatrixReal*, double ** *inMatrixImag*, int *numRows*, int *numCols*, double *** *outMatrixReal*, double ** *outMatrixImag*, int * *outRows*, int * *outCols*)

This function calculates the inverse of a square complex matrix.

This procedure is carried out by the LAPACK methods: zgetrf and zgetri. This means that the matrix will be factorized using LU decomposition first, followed by the calculation of the inverse based on:

inv(A)*L = inv(U) for inv(A).

7.2.3 Variable Documentation

7.2.3.1 BEGIN_C_DECLS

7.2.3.2 END_C_DECLS

7.3 libstructural.h File Reference

All definitions needed for the Structural Analysis Library.

```
#include "libutil.h"
#include <vector>
#include <string>
#include <map>
#include "libla.h"
#include "matrix.h"
#include "complex.h"
```

Namespaces

- namespace LIB_STRUCTURAL

Classes

- class LIB_STRUCTURAL::LibStructural

  Entrypoint for the C++ API of the Structural Analysis Library.

Functions

- LIB_EXTERN int LibStructural_analyzeWithFullyPivotedLU (char **outMessage, int *nLength)

  Uses fully pivoted LU decomposition for structural analysis.

- LIB_EXTERN int LibStructural_analyzeWithFullyPivotedLUwithTests (char **outMessage, int *nLength)

  Uses fully pivoted LU decomposition for structural analysis.

- LIB_EXTERN int LibStructural_analyzeWithLU (char **outMessage, int *nLength)

  Uses LU Decomposition for structural analysis.

- LIB_EXTERN int LibStructural_analyzeWithLUandRunTests (char **outMessage, int *nLength)

  Uses LU Decomposition for structural analysis.

- LIB_EXTERN int LibStructural_analyzeWithQR (char **outMessage, int *nLength)

  Uses QR factorization for structural analysis.

- LIB_EXTERN void LibStructural_freeMatrix (void **matrix, int numRows)

  Frees a matrix previously allocated by this library.

- LIB_EXTERN void LibStructural_freeVector (void *vector)

  Frees a vector previously allocated by this library.

- LIB_EXTERN int LibStructural_getColumnReorderedNrMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the Nr Matrix repartitioned into NIC (independent columns) and NDC (dependent columns).

- LIB_EXTERN int LibStructural_getColumnReorderedNrMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the Nr Matrix row and column labels (repartitioned into NIC and NDC).

- LIB_EXTERN int LibStructural_getConservedLaws (char ***outArray, int *outLength)

  Returns algebraic expressions for the conservation laws.

- LIB_EXTERN int LibStructural_getConservedSums (double **outArray, int *outLength)

  Returns values for conservation laws using the current initial conditions.

- LIB_EXTERN int LibStructural_getDependentReactionIds (char ***outArray, int *outLength)

  Returns the list of dependent reaction Ids.

- LIB_EXTERN int LibStructural_getDependentSpeciesIds (char ***outArray, int *outLength)

  Returns the list of dependent species Ids.

- LIB_EXTERN int LibStructural_getFullyReorderedStoichiometryMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

- LIB_EXTERN int LibStructural_getFullyReorderedStoichiometryMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the row and column labels for the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

- LIB_EXTERN int LibStructural_getGammaMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns Gamma, the conservation law array.

- LIB_EXTERN int LibStructural_getGammaMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the row and column labels for Gamma, the conservation law array.

- LIB_EXTERN int LibStructural_getIndependentReactionIds (char ***outArray, int *outLength)

  Returns the list of independent reaction ids.

- LIB_EXTERN int LibStructural_getIndependentSpeciesIds (char ***outArray, int *outLength)

  Returns the list of independent species ids.

- LIB_EXTERN int LibStructural_getInitialConditions (char ***outVariableNames, double **outValues, int *outLength)

  Returns the initial conditions used in the model.

- LIB_EXTERN int LibStructural_getK0Matrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the K0 Matrix.

- LIB_EXTERN int LibStructural_getK0MatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the K0 Matrix row and column labels.

- LIB_EXTERN int LibStructural_getKMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the K matrix (right nullspace of Nr) The K matrix has the structure, [I K0]'.

- LIB_EXTERN int LibStructural_getKMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the K matrix row and column labels.

- LIB_EXTERN int LibStructural_getL0Matrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the L0 Matrix.

- LIB_EXTERN int LibStructural_getL0MatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the L0 Matrix row and column labels.

- LIB_EXTERN int LibStructural_getLinkMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns L, the Link Matrix, left nullspace (aka nullspace of the transpose Nr).

- LIB_EXTERN int LibStructural_getLinkMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the row and column labels for the Link Matrix, L.

- LIB_EXTERN int LibStructural_getModelName (char **outMessage, int *nLength)

  Returns the name of the model.

- LIB_EXTERN int LibStructural_getN0Matrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the N0 Matrix.

- LIB_EXTERN int LibStructural_getN0MatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the N0 Matrix row and column labels.

- LIB_EXTERN int LibStructural_getNDCMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the NDC Matrix (the set of linearly dependent columns of Nr).

- LIB_EXTERN int LibStructural_getNDCMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the NDC Matrix row and column labels.

- LIB_EXTERN int LibStructural_getNICMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the NIC Matrix (the set of linearly independent columns of Nr).

- LIB_EXTERN int LibStructural_getNICMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the NIC Matrix row and column labels.

- LIB_EXTERN double LibStructural_getNmatrixSparsity ()

  Returns the percentage of nonzero values in the stoichiometry matrix.

- LIB_EXTERN int LibStructural_getNrMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the Nr Matrix. The rows of the Nr matrix will be linearly independent.

- LIB_EXTERN int LibStructural_getNrMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the Nr Matrix row and column labels.

- LIB_EXTERN int LibStructural_getNumConservedSums ()

  Returns the number of conservation laws.

- LIB_EXTERN int LibStructural_getNumDepReactions ()

  Returns the number of dependent reactions.

- LIB_EXTERN int LibStructural_getNumDepSpecies ()

  Returns the number of dependent species.

- LIB_EXTERN int LibStructural_getNumIndReactions ()

  Returns the number of independent reactions.

- LIB_EXTERN int LibStructural_getNumIndSpecies ()

  Returns the number of independent species.

- LIB_EXTERN int LibStructural_getNumReactions ()

  Returns the total number of reactions.

- LIB_EXTERN int LibStructural_getNumSpecies ()

  Returns the total number of species.

- LIB_EXTERN int LibStructural_getRank ()

  Returns the rank of the stoichiometry matrix.

- LIB_EXTERN int LibStructural_getReactionIds (char ***outArray, int *outLength)

  Returns the list of unordered Reactions. Returns the original list of reactions in the same order as when it was loaded.

- LIB_EXTERN int LibStructural_getReorderedReactionIds (char ***outArray, int *outLength)

  Returns the reordered list of reactions Ids.

- LIB_EXTERN int LibStructural_getReorderedSpeciesIds (char ***outArray, int *outLength)

  Returns the reordered list of molecular species.

- LIB_EXTERN int LibStructural_getReorderedStoichiometryMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the reordered stoichiometry matrix (row reordered stoichiometry matrix, columns are not re-ordered!).

- LIB_EXTERN int LibStructural_getReorderedStoichiometryMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix).

- LIB_EXTERN int LibStructural_getSpeciesIds (char ***outArray, int *outLength)

  Returns the unordered list of species Ids.

- LIB_EXTERN int LibStructural_getStoichiometryMatrix (double ***outMatrix, int *outRows, int *outCols)

  Returns the original, unaltered stoichiometry matrix.

- LIB_EXTERN int LibStructural_getStoichiometryMatrixLabels (char ***outRowLabels, int *outRowCount, char ***outColLabels, int *outColCount)

  Returns the row and column labels for the original and unaltered stoichiometry matrix.

- LIB_EXTERN int LibStructural_getTestDetails (char **outMessage, int *nLength)

  Return Details about validation tests.

- LIB_EXTERN double LibStructural_getTolerance ()

  Get user specified tolerance.

- LIB_EXTERN int LibStructural_loadReactionNames (const char **reactionNames, const int nLength)

  Load reaction names.

- LIB_EXTERN int LibStructural_loadSBML (const char *sSBML, char **outMessage, int *nLength)

  Load a SBML model.

- LIB_EXTERN int LibStructural_loadSBMLFromFile (const char *sFileName, char **outMessage, int *nLength)

  Load a SBML model from the specified file.

- LIB_EXTERN int LibStructural_loadSBMLwithTests (const char *sSBML, char **outMessage, int *nLength)

  Load an SBML model into the library and carry out tests using the internal test suite.

- LIB_EXTERN int LibStructural_loadSpecies (const char **speciesNames, const double *speciesValues, const int nLength)

  Load species names and initial values.

- LIB_EXTERN int LibStructural_loadStoichiometryMatrix (const double **oMatrix, const int nRows, const int nCols)

  Load a new stoichiometry matrix.

- LIB_EXTERN void LibStructural_setTolerance (const double dTolerance)

  Set user specified tolerance.

- LIB_EXTERN int LibStructural_validateStructuralMatrices (int **outResults, int *outLength)

  Validates structural matrices.

Variables

- BEGIN_C_DECLS
- END_C_DECLS

7.3.1 Detailed Description

All definitions needed for the Structural Analysis Library.

The structural analysis of stoichiometric networks is an important step in a number of computational methods in systems biology. The structure of a network based on the stoichiometry matrix is divided into two areas, structural constraints imposed by moiety conservation and constraints imposed by flux distributions at steady state. The former constraints have important applications in numerical methods for simulation and the analysis of control, while the later constraints have important applications in flux balance analysis. The LibStructural API provides a wide variety of methods that permit access to the constraint information in the stoichiometry matrix.

Stoichiometric Constraints

Moiety constraints concern the conservation of molecular subgroups in stoichiometric networks. Their existence results in dependencies among the model differential equations and the emergence of additional model parameters in the form of moiety mass totals. In the API we provide robust methods for extracting the constraint information and include specific methods to obtain for example the number of moiety cycles, the number of independent and dependent species and all the pertinent matrices such as the link matrix, reduced stoichiometry matrix etc. In addition to moiety constraints the library also provides robust methods for determining the flux constraints in a model. These include the dependent and independent flux, and the K matrix (and corresponding terms) that relates the two.

All matrices provided by the API are fully labeled with reaction and species labels. The API can accept models either directly from standard SBML or by specifying the stoichiometry matrix. In the case of SBML the species and reaction labels are obtained directly from the SBML otherwise they are entered manually.

Further and more detailed information on this work can be found in Reder (1988), Sauro and Ingalls (2004), Vallabhajosyula et al. (2005).

Author:

Frank T. Bergmann (fbergman@u.washington.edu)
Herbert M. Sauro
Ravishankar Rao Vallabhajosyula (developed a previous version of the sructural analysis code)

7.3.2 Function Documentation

7.3.2.1 LIB_EXTERN int LibStructural_analyzeWithFullyPivotedLU (char ** *outMessage*, int * *nLength*)

Uses fully pivoted LU decomposition for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

Remarks:

Unlike the other methods, this method handles only square stoichiometry matrices. This method was only included for backward compatibility use LibStructural_analyzeWithQR

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand. See LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML or LibStructural_loadSBMLFromFile

7.3.2.2 LIB_EXTERN int LibStructural_analyzeWithFullyPivotedLUwithTests (char ** *outMessage*, int * *nLength*)

Uses fully pivoted LU decomposition for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

This method additionally performs the integrated test suite and returns those results.

Remarks:

Unlike the other methods, this method handles only square stoichiometry matrices. For non-square matrices use a method like LibStructural_analyzeWithQR.

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand see LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML or LibStructural_loadSBMLFromFile

7.3.2.3 LIB_EXTERN int LibStructural_analyzeWithLU (char ** *outMessage*, int * *nLength*)

Uses LU Decomposition for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand see LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML or LibStructural_loadSBMLFromFile

7.3.2.4 LIB_EXTERN int LibStructural_analyzeWithLUandRunTests (char ** *outMessage*, int * *nLength*)

Uses LU Decomposition for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

This method additionally performs the integrated test suite and returns those results.

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand see LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML or LibStructural_loadSBMLFromFile

7.3.2.5 LIB_EXTERN int LibStructural_analyzeWithQR (char ** *outMessage*, int * *nLength*)

Uses QR factorization for structural analysis.

This method performs the actual analysis of the stoichiometry matrix (loaded either via LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML. Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

- LibStructural_analyzeWithQR,
- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

Remarks:

This is the prefered method for structural analysis.

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand see LibStructural_loadStoichiometryMatrix or LibStructural_loadSBML or LibStructural_loadSBMLFromFile

Examples:

examples/c/loadlabelledstoichiometry.c, and examples/c/loadstoichiometry.c.

7.3.2.6 LIB_EXTERN void LibStructural_freeMatrix (void ** *matrix*, int *numRows*)

Frees a matrix previously allocated by this library.

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.7 LIB_EXTERN void LibStructural_freeVector (void * *vector*)

Frees a vector previously allocated by this library.

Examples:

examples/c/loadlabelledstoichiometry.c, examples/c/loadsbmlfromfile.c, and examples/c/loadstoichiometry.c.

7.3.2.8 LIB_EXTERN int LibStructural_getColumnReorderedNrMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the Nr Matrix repartitioned into NIC (independent columns) and NDC (dependent columns).

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.9 LIB_EXTERN int LibStructural_getColumnReorderedNrMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the Nr Matrix row and column labels (repartitioned into NIC and NDC).

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.10 LIB_EXTERN int LibStructural_getConservedLaws (char *** *outArray*, int * *outLength*)

Returns algebraic expressions for the conservation laws.

Parameters:

outArray pointer to string array that will be allocated and filled

outLength the number of conservation laws

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.11 LIB_EXTERN int LibStructural_getConservedSums (double ** *outArray*, int * *outLength*)

Returns values for conservation laws using the current initial conditions.

Parameters:

outArray will be allocated and filled with a double vector of all conserved sums

outLength is the number of conserved sums

Remarks:

free outArray using LibStructural_freeVector

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.12 LIB_EXTERN int LibStructural_getDependentReactionIds (char *** *outArray*, int * *outLength*)

Returns the list of dependent reaction Ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the dependent reaction Ids

outLength the number of dependent reactions

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.13 LIB_EXTERN int LibStructural_getDependentSpeciesIds (char *** *outArray*, int * *outLength*)

Returns the list of dependent species Ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the dependent species Ids

outLength the number of dependent species

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.14 LIB_EXTERN int LibStructural_getFullyReorderedStoichiometryMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.15 LIB_EXTERN int LibStructural_getFullyReorderedStoichiometryMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the row and column labels for the fully reordered stoichiometry matrix (row and column reordered stoichiometry matrix).

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.16 LIB_EXTERN int LibStructural_getGammaMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns Gamma, the conservation law array.

Each row represents a single conservation law where the column indicate the participating molecular species. The number of rows is therefore equal to the number of conservation laws. Columns are ordered according to the rows in the reordered stoichiometry matrix, see LibStructural_getReorderedSpeciesId and LibStructural_getReorderedStoichiometryMatrix.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.17 LIB_EXTERN int LibStructural_getGammaMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the row and column labels for Gamma, the conservation law array.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.18 LIB_EXTERN int LibStructural_getIndependentReactionIds (char *** *outArray*, int * *outLength*)

Returns the list of independent reaction ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the independent reaction Ids

outLength the number of independent reaction

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.19 LIB_EXTERN int LibStructural_getIndependentSpeciesIds (char *** *outArray*, int * *outLength*)

Returns the list of independent species ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the independent species Ids

outLength the number of independent species

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.20 LIB_EXTERN int LibStructural_getInitialConditions (char *** *outVariableNames*, double ** *outValues*, int * *outLength*)

Returns the initial conditions used in the model.

Parameters:

outVariableNames a string vector of all species Ids

outValues a double vector of corresponding initial conditions

outLength number of elements in outVariableNames and outValues (number of species)

7.3.2.21 LIB_EXTERN int LibStructural_getK0Matrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the K0 Matrix.

K0 is defined such that K0 = -(NIC)^-1 NDC, or equivalently, [NDC NIC][I K0]' = 0 where [NDC NIC] = Nr

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.22 LIB_EXTERN int LibStructural_getK0MatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the K0 Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.23 LIB_EXTERN int LibStructural_getKMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the K matrix (right nullspace of Nr) The K matrix has the structure, [I K0]'.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.24 LIB_EXTERN int LibStructural_getKMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the K matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.25 LIB_EXTERN int LibStructural_getL0Matrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the L0 Matrix.

L0 is defined such that L0 Nr = N0. L0 forms part of the link matrix, L. N0 is the set of linear dependent rows from the lower portion of the reordered stoichiometry matrix.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods have been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.26 LIB_EXTERN int LibStructural_getL0MatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the L0 Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.27 LIB_EXTERN int LibStructural_getLinkMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns L, the Link Matrix, left nullspace (aka nullspace of the transpose Nr).

L will have the structure, [I L0]', such that L Nr = N

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.28 LIB_EXTERN int LibStructural_getLinkMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the row and column labels for the Link Matrix, L.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.29 LIB_EXTERN int LibStructural_getModelName (char ** *outMessage*, int * *nLength*)

Returns the name of the model.

Returns the name of the model if SBML model has Name-tag, otherwise it returns the SBML id. If only a stoichiometry matrix was loaded 'untitled' will be returned.

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Remarks:

free outMessage using LibStructural_freeVector

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.30 LIB_EXTERN int LibStructural_getN0Matrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the N0 Matrix.

The N0 matrix is the set of linearly dependent rows of N where L0 Nr = N0.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.31 LIB_EXTERN int LibStructural_getN0MatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the N0 Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.32 LIB_EXTERN int LibStructural_getNDCMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the NDC Matrix (the set of linearly dependent columns of Nr).

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.33 LIB_EXTERN int LibStructural_getNDCMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the NDC Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.34 LIB_EXTERN int LibStructural_getNICMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the NIC Matrix (the set of linearly independent columns of Nr).

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.35 LIB_EXTERN int LibStructural_getNICMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the NIC Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.36 LIB_EXTERN double LibStructural_getNmatrixSparsity ()

Returns the percentage of nonzero values in the stoichiometry matrix.

7.3.2.37 LIB_EXTERN int LibStructural_getNrMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the Nr Matrix. The rows of the Nr matrix will be linearly independent.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.38 LIB_EXTERN int LibStructural_getNrMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the Nr Matrix row and column labels.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.39 LIB_EXTERN int LibStructural_getNumConservedSums ()

Returns the number of conservation laws.

Returns:

the number of conservation laws

7.3.2.40 LIB_EXTERN int LibStructural_getNumDepReactions ()

Returns the number of dependent reactions.

7.3.2.41 LIB_EXTERN int LibStructural_getNumDepSpecies ()

Returns the number of dependent species.

7.3.2.42 LIB_EXTERN int LibStructural_getNumIndReactions ()

Returns the number of independent reactions.

7.3.2.43 LIB_EXTERN int LibStructural_getNumIndSpecies ()

Returns the number of independent species.

7.3.2.44 LIB_EXTERN int LibStructural_getNumReactions ()

Returns the total number of reactions.

7.3.2.45 LIB_EXTERN int LibStructural_getNumSpecies ()

Returns the total number of species.

7.3.2.46 LIB_EXTERN int LibStructural_getRank ()

Returns the rank of the stoichiometry matrix.

7.3.2.47 LIB_EXTERN int LibStructural_getReactionIds (char *** *outArray*, int * *outLength*)

Returns the list of unordered Reactions. Returns the original list of reactions in the same order as when it was loaded.

Parameters:

outArray pointer to string array that will be allocated and filled with the reaction Ids

outLength the number of reactions

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.48 LIB_EXTERN int LibStructural_getReorderedReactionIds (char *** *outArray*, int * *outLength*)

Returns the reordered list of reactions Ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the reordered reaction Ids

outLength the number of species

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.49 LIB_EXTERN int LibStructural_getReorderedSpeciesIds (char *** *outArray*, int * *outLength*)

Returns the reordered list of molecular species.

Parameters:

outArray pointer to string array that will be allocated and filled with the species Ids

outLength the number of species

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

7.3.2.50 LIB_EXTERN int LibStructural_getReorderedStoichiometryMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the reordered stoichiometry matrix (row reordered stoichiometry matrix, columns are not reordered!).

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.51 LIB_EXTERN int LibStructural_getReorderedStoichiometryMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix).

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.52 LIB_EXTERN int LibStructural_getSpeciesIds (char *** *outArray*, int * *outLength*)

Returns the unordered list of species Ids.

Parameters:

outArray pointer to string array that will be allocated and filled with the species Ids

outLength the number of species

Remarks:

free outArray using LibStructural_freeMatrix with the outLength parameter

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.2.53 LIB_EXTERN int LibStructural_getStoichiometryMatrix (double *** *outMatrix*, int * *outRows*, int * *outCols*)

Returns the original, unaltered stoichiometry matrix.

Parameters:

outMatrix a pointer to a double array that holds the output

outRows will be overwritten with the number of rows

outCols will be overwritten with the number of columns.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the returned matrix call LibStructural_freeMatrix with the outMatrix and outRows as parameter.

7.3.2.54 LIB_EXTERN int LibStructural_getStoichiometryMatrixLabels (char *** *outRowLabels*, int * *outRowCount*, char *** *outColLabels*, int * *outColCount*)

Returns the row and column labels for the original and unaltered stoichiometry matrix.

Parameters:

outRowLabels a pointer to a string array where the row labels will be allocated and written.

outRowCount after the call this variable will hold the number of row labels returned.

outColLabels a pointer to a string array where the column labels will be allocated and written.

outColCount after the call this variable will hold the number of column labels returned.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Remarks:

To free the string arrays (outRowLabels and outColLabels) call LibStructural_freeMatrix with the string array and its corresponding length (outRowCount or outColCount)

7.3.2.55 LIB_EXTERN int LibStructural_getTestDetails (char ** *outMessage*, int * *nLength*)

Return Details about validation tests.

Parameters:

outMessage a pointer to a string where status information of the analysis will be returned.

nLength the length of the message.

Remarks:

free outMessage using LibStructural_freeVector

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

Examples:

examples/c/loadlabelledstoichiometry.c, examples/c/loadsbmlfromfile.c, and examples/c/loadstoichiometry.c.

7.3.2.56 LIB_EXTERN double LibStructural_getTolerance ()

Get user specified tolerance.

This function gets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

Returns:

the tolerance used by the library to determine a value close to zero

7.3.2.57 LIB_EXTERN int LibStructural_loadReactionNames (const char ** *reactionNames*, const int *nLength*)

Load reaction names.

This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the reaction names (ids).

Parameters:

reactionNames an array of strings of reaction names with length nLength

nLength number of elements in reactionNames

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Remarks:

This method should only be called after LibStructural_loadStoichiometryMatrix

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.58 LIB_EXTERN int LibStructural_loadSBML (const char * *sSBML*, char ** *outMessage*, int * *nLength*)

Load a SBML model.

Parameters:

sSBML the SBML string to load into the library

outMessage a pointer to a string that the library can use to provide information about the loaded SBML

nLength is the length of the above message

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred (invalid SBML)

7.3.2.59 LIB_EXTERN int LibStructural_loadSBMLFromFile (const char * *sFileName*, char ** *outMessage*, int * *nLength*)

Load a SBML model from the specified file.

Parameters:

sFileName the full path to the SBML file to be loaded.

outMessage a pointer to a string that the library can use to provide information about the loaded SBML

nLength is the length of the above message

Remarks:

To avoid unintentional errors be sure to pass in the full path to the SBML file.

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred (invalid SBML, file not readable ...).

Examples:

examples/c/loadsbmlfromfile.c.

7.3.2.60 LIB_EXTERN int LibStructural_loadSBMLwithTests (const char * *sSBML*, char ** *outMessage*, int * *nLength*)

Load an SBML model into the library and carry out tests using the internal test suite.

Parameters:

sSBML the SBML string to load into the library

outMessage a pointer to a string that contains information about the loaded model as well as the test results of the internal test suite.

nLength is the length of the above message

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred (invalid SBML)

7.3.2.61 LIB_EXTERN int LibStructural_loadSpecies (const char ** *speciesNames*, const double * *speciesValues*, const int *nLength*)

Load species names and initial values.

This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the species names (ids). It is also possible to provide an initial condition for each of the species. This will be used when calculating the conserved sums.

Parameters:

speciesNames an array of strings of species names with length nLength

speciesValues an array of real numbers of species concentrations corresponding to the speciesName with the same index

nLength number of elements in speciesNames and speciesValues

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Remarks:

This method should only be called after LibStructural_loadStoichiometryMatrix

Examples:

examples/c/loadlabelledstoichiometry.c.

7.3.2.62 LIB_EXTERN int LibStructural_loadStoichiometryMatrix (const double ** *oMatrix*, const int *nRows*, const int *nCols*)

Load a new stoichiometry matrix.

Loads the stoichiometry matrix into the library. To analyze the stoichiometry call one of the following:

- LibStructural_analyzeWithQR,

- LibStructural_analyzeWithLU,
- LibStructural_analyzeWithLUandRunTests,
- LibStructural_analyzeWithFullyPivotedLU or
- LibStructural_analyzeWithFullyPivotedLUwithTests

Remarks:

if matrix labels are needed it is recommended to call LibStructural_loadSpecies and LibStructural_loadReactionNames after a call to this function.

Parameters:

oMatrix a pointer to a double** matrix

nRows the number of rows of the matrix

nCols the number of columns of the matrix

Returns:

The return value will be zero (0) when successful, and negative (-1) in case an error occurred

Examples:

examples/c/loadlabelledstoichiometry.c, and examples/c/loadstoichiometry.c.

7.3.2.63 LIB_EXTERN void LibStructural_setTolerance (const double *dTolerance*)

Set user specified tolerance.

This function sets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

Parameters:

dTolerance Sets the tolerance used by the library to determine a value close to zero

7.3.2.64 LIB_EXTERN int LibStructural_validateStructuralMatrices (int ** *outResults*, int * *outLength*)

Validates structural matrices.

Calling this method will run the internal test suite against the structural matrices those tests include:

- Test 1 : Gamma*N = 0 (Zero matrix)
- Test 2 : Rank(N) using SVD (5) is same as m0 (5)
- Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
- Test 4 : Rank(NR) using QR (5) is same as m0 (5)
- Test 5 : L0 obtained with QR matches Q21*inv(Q11)
- Test 6 : N*K = 0 (Zero matrix)

Parameters:

outResults an integer vector, each element represents the result for one of the above tests (the 0th element representing the test result for test1), if the test passed the value is 1 and 0 otherwise.

outLength number of tests

Remarks:

free outResults using LibStructural_freeVector

Returns:

The return value will be zero (0) when successful, and negative (-1) in case no stoichiometry matrix was loaded beforehand or none of the analysis methods has been called yet.

7.3.3 Variable Documentation

7.3.3.1 BEGIN_C_DECLS

7.3.3.2 END_C_DECLS

7.4 libutil.h File Reference

Defines

- #define BEGIN_C_DECLS extern "C" {
- #define END_C_DECLS }
- #define LIB_EXTERN

7.4.1 Define Documentation

7.4.1.1 #define BEGIN_C_DECLS extern "C" {

7.4.1.2 #define END_C_DECLS }

7.4.1.3 #define LIB_EXTERN

7.5 matrix.h File Reference

```
#include <iosfwd>
```

Namespaces

- namespace LIB_LA

Classes

- class LIB_LA::Matrix< T >

  LIB_LA::Matrix is the matrix class used by LIB_LA::LibLA and LIB_STRUTURAL::LibStructural.

Typedefs

- typedef Matrix< Complex > LIB_LA::ComplexMatrix

  defines a complex matrix (hides the templates in signatures)

- typedef Matrix< double > LIB_LA::DoubleMatrix

  defines a real matrix (hides the templates in signatures)

- typedef Matrix< int > LIB_LA::IntMatrix

  defines a integer matrix (hides the templates in signatures)

Chapter 8

Example Documentation

8.1 examples/c/loadlabelledstoichiometry.c

This is an example of how to load a labeled stoichiometry matrix and read test results. The example also shows how to print the reordered stoichiometry matrix as well as the Gamma matrix.

```
#include <stdio.h>           // for printf
#include <stdlib.h>          // for malloc
#include <string.h>          // for memset
#include <libstructural.h>   // the structural analysis library

// construct simple stoichiometry matrix with labelled species and reactions
void GetMatrixFromSomeWhere(double** *oMatrix, int *nRows, int *nCols,
         char** *speciesNames, double* *initialConcentrations,
         char** *reactionNames);

// gets the reordered stoichiometry matrix from the library
void PrintReorderedStoichiometryMatrix();

// gets the gamma matrix from the librar
void PrintGammaMatrix();


int main (int argc, char** argv)
{
        int      i,j;
        int      nRows;
        int      nCols;
        double** oMatrix;
        char*    sMessage;
        char**   speciesNames;
        char**   reactionNames;
        double*  initialConcentrations;
        int      nLength;


        // get matrix to analyze from another part of the code as well
        // as species and reaction names
        GetMatrixFromSomeWhere(&oMatrix, &nRows, &nCols,
               &speciesNames, &initialConcentrations, &reactionNames);


        // load matrix into the structural analysis library
        LibStructural_loadStoichiometryMatrix (oMatrix, nRows, nCols);
        // load species names and initial concentrations
        LibStructural_loadSpecies(speciesNames, initialConcentrations, nRows);
```

```
        // load reaction names
        LibStructural_loadReactionNames(reactionNames, nCols);

        // analyze the stoichiometry matrix using the QR method
        LibStructural_analyzeWithQR( &sMessage, &nLength);

        // print model overview
        printf("%s", sMessage);

        // free the memory used by the message
        LibStructural_freeVector(sMessage);

        // obtain and print the test results
        LibStructural_getTestDetails( &sMessage, &nLength );
        printf("%s", sMessage);

        // finally free the memory used by the message
        LibStructural_freeVector(sMessage);


        // Print Reordered Stoichiometry Matrix
        PrintReorderedStoichiometryMatrix();

        // Print Gamma Matrix
        PrintGammaMatrix();


        // and free the memory used to hold the stoichiometry matrix
        for (i = 0; i < nRows; i++)
                free(oMatrix[i]);
        free(oMatrix);

        // free species names
        for (i = 0; i < nRows; i++)
                free(speciesNames[i]);
        free(speciesNames);

        // free reaction names
        for (i = 0; i < nCols; i++)
                free(reactionNames[i]);
        free(reactionNames);

        return 0;

}

void PrintReorderedStoichiometryMatrix()
{
        int      i,j;
        double** reorderedStoichiometryMatrix;
        int      reorderedNumRows;
        int      reorderedNumCols;
        char**   reorderedCols;
        char**   reorderedRows;

        printf("\nReordered Stoichiometry Matrix");

        // get Reordered Stoichiometry Matrix
        LibStructural_getReorderedStoichiometryMatrix
                (&reorderedStoichiometryMatrix, &reorderedNumRows, &reorderedNumCols);
        LibStructural_getReorderedStoichiometryMatrixLabels
                (&reorderedRows, &reorderedNumRows, &reorderedCols, &reorderedNumCols);

        // print Reordered stoichiometry matrix:
        printf("\n\t");
        for (i = 0; i < reorderedNumCols; i++)
                printf("%s\t", reorderedCols[i]);
```

```
        printf("\n");

        for (i = 0; i < reorderedNumRows; i++)
        {
                printf("%s\t", reorderedRows[i]);
                for (j = 0; j < reorderedNumCols; j++)
                        printf ("%2.1lf\t", reorderedStoichiometryMatrix[i][j]);
                printf("\n");
        }

        // free reordered stoichiometry matrix and labels
        LibStructural_freeMatrix((void**)reorderedStoichiometryMatrix, reorderedNumRows);
        LibStructural_freeMatrix((void**)reorderedCols, reorderedNumCols);
        LibStructural_freeMatrix((void**)reorderedRows, reorderedNumRows);

        printf("\n");
}

void PrintGammaMatrix()
{
        int      i,j;
        double** gammaMatrix;
        int      gammaNumRows;
        int      gammaNumCols;
        char**   gammaCols;
        char**   gammaRows;


        printf("\nGamma Matrix");

        // get Gamma Matrix and labels
        LibStructural_getGammaMatrix
                (&gammaMatrix, &gammaNumRows, &gammaNumCols);
        LibStructural_getGammaMatrixLabels
                (&gammaRows, &gammaNumRows, &gammaCols, &gammaNumCols);

        // print gamma matrix:
        printf("\n\t");
        for (i = 0; i < gammaNumCols; i++)
                printf("%s\t", gammaCols[i]);
        printf("\n");

        for (i = 0; i < gammaNumRows; i++)
        {
                printf("%s\t", gammaRows[i]);
                for (j = 0; j < gammaNumCols; j++)
                        printf ("%2.1lf\t", gammaMatrix[i][j]);
                printf("\n");
        }

        // free gamma stoichiometry matrix and labels
        LibStructural_freeMatrix((void**)gammaMatrix, gammaNumRows);
        LibStructural_freeMatrix((void**)gammaCols, gammaNumCols);
        LibStructural_freeMatrix((void**)gammaRows, gammaNumRows);

        printf("\n");

}


void GetMatrixFromSomeWhere(double** *oMatrix, int *nRows, int *nCols,
         char** *speciesNames, double* *initialConcentrations,
         char** *reactionNames)
{
        int numCols, numRows, i;
        numRows = 4; numCols = 3;
```

```
        // initialize memory needed for the stoichiometry matrix
        *oMatrix = (double**)malloc(sizeof(double*)*numRows);
        memset(*oMatrix, 0, sizeof(double*)*numRows);

        for (i = 0; i < numRows; i ++)
        {
                (*oMatrix)[i] = (double*)malloc(sizeof(double)*numCols);
                memset((*oMatrix)[i], 0, sizeof(double)*numCols);
        }

        // initialize memory needed for speciesNames
        (*speciesNames) = (char**)malloc(sizeof(char*)*numRows);
        memset(*speciesNames, 0, sizeof(char*)*numRows);

        *initialConcentrations = (double*)malloc(sizeof(double)*numRows);
        memset(*initialConcentrations, 0, sizeof(double)*numRows);

        // initialize memory needed for reactionNames
        (*reactionNames) = (char**)malloc(sizeof(char*)*numCols);
        memset(*reactionNames, 0, sizeof(char*)*numCols);

        // set non zero entries of the stoichiometry matrix
        (*oMatrix)[0][1] = -1.0;        (*oMatrix)[0][2] =  1.0;      // ES
        (*oMatrix)[1][0] =  1.0;        (*oMatrix)[1][2] = -1.0;      // S2
        (*oMatrix)[2][0] = -1.0;        (*oMatrix)[2][1] =  1.0;      // S1
        (*oMatrix)[3][1] =  1.0;        (*oMatrix)[3][2] = -1.0;      // E

        // set species names
        (*speciesNames)[0] = strdup("S2");  (*speciesNames)[1] = strdup("ES");
        (*speciesNames)[2] = strdup("S1");  (*speciesNames)[3] = strdup("E");

        // set reaction names
        (*reactionNames)[0] = strdup("J1"); (*reactionNames)[1] = strdup("J2");
        (*reactionNames)[2] = strdup("J3");

        // be sure to return number of rows and columns
        *nRows = numRows;
        *nCols = numCols;
}



//The above returns the following output:
//
//--------------------------------------------------------------------------------
//--------------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//--------------------------------------------------------------------------------
//--------------------------------------------------------------------------------
//Size of Stochiometric Matrix: 4 x 3 (Rank is  2)
//Nonzero entries in Stochiometric Matrix: 8  (66.6667% full)
//
//Independent Species (2) :
//S2, ES
//
//Dependent Species (2) :
//S1, E
//
//L0 : There are 2 dependencies. L0 is a 2x2 matrix.
//
//Conserved Entities
//1:  + S2 + ES + S1
//2:  + S2 + E
//--------------------------------------------------------------------------------
//--------------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
```

```
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//--------------------------------------------------------------------------------
//--------------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (2) is same as m0 (2)
//Passed Test 3 : Rank(NR) using SVD (2) is same as m0 (2)
//Passed Test 4 : Rank(NR) using QR (2) is same as m0 (2)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
//
//Reordered Stoichiometry Matrix
//J1      J2      J3
//S2      0.0     -1.0    1.0
//ES      1.0     0.0     -1.0
//S1      -1.0    1.0     0.0
//E       0.0     1.0     -1.0
//
//
//Gamma Matrix
//S2      ES      S1      E
//0       1.0     1.0     1.0     0.0
//1       1.0     -0.0    0.0     1.0
//
```

8.2 examples/c/loadsbmlfromfile.c

This is an example of how to load a SBML file and print structural analysis test results.

```
#include <stdio.h>           // for printf
#include <stdlib.h>          // for malloc
#include <string.h>          // for memset
#include <libstructural.h>   // the structural analysis library


int main (int argc, char** argv)
{
        int             result;
        char*   message;
        int             length;

        if (argc < 2)
        {
                printf("please provide a filename as argument");
                return -1;
        }

        // load the sbml file and check the argument
        result = LibStructural_loadSBMLFromFile(argv[1], &message, &length);
        if (result != 0)
        {
                printf("the SBML file %s could not be loaded.", argv[1]);
                return -1;
        }

        // print model overview
        printf("%s", message);

        // free the memory used by the message
        LibStructural_freeVector(message);

        // obtain and print the test results
        LibStructural_getTestDetails( &message, &length );
        printf("%s", message);

        // finally free the memory used by the message
        LibStructural_freeVector(message);


        return 0;
}

// Output for model BorisEJB.xml(available with SBW distribution) passed in
//
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
//Size of Stochiometric Matrix: 8 x 10 (Rank is  5)
//Nonzero entries in Stochiometric Matrix: 20  (25% full)
//
//Independent Species (5) :
//MKK_P, MAPK_P, MKKK, MKK, MAPK
//
//Dependent Species (3) :
//MKK_PP, MKKK_P, MAPK_PP
//
//L0 : There are 3 dependencies. L0 is a 3x5 matrix.
//
//Conserved Entities
```

```
//1:  + MKK_P + MKK + MKK_PP
//2:  + MKKK + MKKK_P
//3:  + MAPK_P + MAPK + MAPK_PP
//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (5) is same as m0 (5)
//Passed Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
//Passed Test 4 : Rank(NR) using QR (5) is same as m0 (5)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
```

8.3 examples/c/loadstoichiometry.c

This is an example of how to load a (unlabeled) stoichiometry matrix and read test details.

```
#include <stdio.h>           // for printf
#include <stdlib.h>          // for malloc
#include <string.h>          // for memset
#include <libstructural.h>   // the structural analysis library

// construct simple stoichiometry matrix
void GetMatrixFromSomeWhere(double** *oMatrix, int *nRows, int *nCols)
{
        int numCols, numRows, i;
        numRows = 4; numCols = 3;

        // initialize memory needed
        *oMatrix = (double**)malloc(sizeof(double*)*numRows);
        memset(*oMatrix, 0, sizeof(double*)*numRows);

        for (i = 0; i < numRows; i ++)
        {
                (*oMatrix)[i] = (double*)malloc(sizeof(double)*numCols);
                memset((*oMatrix)[i], 0, sizeof(double)*numCols);
        }

        // set non zero entries of the stoichiometry matrix
        (*oMatrix)[0][1] = -1.0;    (*oMatrix)[0][2] =  1.0;    // ES
        (*oMatrix)[1][0] =  1.0;    (*oMatrix)[1][2] = -1.0;    // S2
        (*oMatrix)[2][0] = -1.0;    (*oMatrix)[2][1] =  1.0;    // S1
        (*oMatrix)[3][1] =  1.0;    (*oMatrix)[3][2] = -1.0;    // E


        // be sure to return number of rows and columns
        *nRows = numRows;
        *nCols = numCols;
}

int main (int argc, char** argv)
{
        int      i;
        int      nRows;
        int      nCols;
        double** oMatrix;
        char*    sMessage;
        int      nLength;


        // get matrix to analyze from another part of the code
        GetMatrixFromSomeWhere(&oMatrix, &nRows, &nCols);

        // load it into the structural analysis library
        LibStructural_loadStoichiometryMatrix (oMatrix, nRows, nCols);

        // analyze the stoichiometry matrix using the QR method
        LibStructural_analyzeWithQR( &sMessage, &nLength);

        // print model overview
        printf("%s", sMessage);

        // free the memory used by the message
        LibStructural_freeVector(sMessage);

        // obtain and print the test results
        LibStructural_getTestDetails( &sMessage, &nLength );
        printf("%s", sMessage);
```

```
        // finally free the memory used by the message
        LibStructural_freeVector(sMessage);

        // and free the memory used to hold the stoichiometry matrix
        for (i = 0; i < nRows; i++)
                free(oMatrix[i]);
        free(oMatrix);

        return 0;

}

//The program above returns the following output:
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Size of Stochiometric Matrix: 4 x 3 (Rank is  2)
//Nonzero entries in Stochiometric Matrix: 8  (66.6667% full)
//
//Independent Species (2) :
//0, 1
//
//Dependent Species (2) :
//2, 3
//
//L0 : There are 2 dependencies. L0 is a 2x2 matrix.
//
//Conserved Entities
//1:  + 0 + 1 + 2
//2:  + 0 + 3
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (2) is same as m0 (2)
//Passed Test 3 : Rank(NR) using SVD (2) is same as m0 (2)
//Passed Test 4 : Rank(NR) using QR (2) is same as m0 (2)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
```

8.4 examples/cpp/loadsbmlfromfile.cpp

This is an example of how to load a SBML file and print structural analysis test results.

```
#include <iostream>
#include <libstructural.h>

using namespace std;
using namespace LIB_STRUCTURAL;


int main(int argc, char*argv[])
{
        if (argc != 2)
        {
                cerr << "Need one argument, full path to SBML file." << endl;
                return -1;
        }

        // get an instance of the library
        LibStructural* instance = LibStructural::getInstance();

        // print out model overview
        cout << instance->loadSBMLFromFile(argv[1]);

        // print test details
        cout << instance->getTestDetails();

        return 0;
}

// Running the program for BorisEJB.xml (available from the SBW
// distribution) yields:
//
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Size of Stochiometric Matrix: 8 x 10 (Rank is  5)
//Nonzero entries in Stochiometric Matrix: 20  (25% full)
//
//Independent Species (5) :
//MKK_P, MAPK_P, MKKK, MKK, MAPK
//
//Dependent Species (3) :
//MKK_PP, MKKK_P, MAPK_PP
//
//L0 : There are 3 dependencies. L0 is a 3x5 matrix.
//
//Conserved Entities
//1:  + MKK_P + MKK + MKK_PP
//2:  + MKKK + MKKK_P
//3:  + MAPK_P + MAPK + MAPK_PP
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
```

```
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (5) is same as m0 (5)
//Passed Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
//Passed Test 4 : Rank(NR) using QR (5) is same as m0 (5)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
```

8.5 examples/cpp/loadstoichiometry.cpp

This is an example of how to load a stoichiometry matrix and read test details.

```
#include <iostream>
#include <libstructural.h>
#include <matrix.h>
#include <vector>
#include <string>

using namespace std;
using namespace LIB_STRUCTURAL;
using namespace LIB_LA;

static double STOICHIOMETRY[4][3] = {{  0.0, -1.0,  1.0},  // ES
                                     {  1.0,  0.0, -1.0},  // S2
                                     { -1.0,  1.0,  0.0},  // S1
                                     {  0.0,  1.0, -1.0}}; // E

void InitializeValues(DoubleMatrix* &oMatrix,
                      vector<string> &speciesNames,
                      vector<double> &initialConcentrations,
                      vector<string> &reactionNames);


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

        vector<string> speciesNames; vector<string> reactionNames;
        vector<double> initialConentrations; DoubleMatrix* oMatrix;

        // some function that produces the stoichiometry matrix
        // and labels
        InitializeValues(oMatrix, speciesNames, initialConentrations, reactionNames);

        // get an instance of the library
        LibStructural* instance = LibStructural::getInstance();

        // load stoichiometry matrix and labels
        instance->loadStoichiometryMatrix(*oMatrix);
        instance->loadSpecies(speciesNames, initialConentrations);
        instance->loadReactionNames(reactionNames);

        // analyze the model and print overview
        cout << instance->analyzeWithQR();

        // print test details
        cout << instance->getTestDetails();

        // delete stoichiometry matrix
        delete oMatrix;

        return 0;
}

void InitializeValues(DoubleMatrix* &oMatrix,
                      vector<string> &speciesNames,
                      vector<double> &initialConcentrations,
                      vector<string> &reactionNames)
{

        // build stoichiometry matrix (this should be easier!!!)
        double *row1 = STOICHIOMETRY[0], *row2 = STOICHIOMETRY[1],
               *row3 = STOICHIOMETRY[2], *row4 = STOICHIOMETRY[3];

        const double* oMatrixRAW[] = {row1, row2, row3, row4};
```

```
        oMatrix = new DoubleMatrix(oMatrixRAW, 4, 3);

        speciesNames.push_back("ES"); speciesNames.push_back("S2");
        speciesNames.push_back("S1"); speciesNames.push_back("E");

        initialConcentrations.push_back(1.0);   initialConcentrations.push_back(1.0);
        initialConcentrations.push_back(1.0);

        reactionNames.push_back("J1"); reactionNames.push_back("J2");
        reactionNames.push_back("J3");
}

// This is the output generated by the code above:
//
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Size of Stochiometric Matrix: 4 x 3 (Rank is  2)
//Nonzero entries in Stochiometric Matrix: 8  (66.6667% full)
//
//Independent Species (2) :
//ES, S2
//
//Dependent Species (2) :
//S1, E
//
//L0 : There are 2 dependencies. L0 is a 2x2 matrix.
//
//Conserved Entities
//1:  + ES + S2 + S1
//2:  + ES + E
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (2) is same as m0 (2)
//Passed Test 3 : Rank(NR) using SVD (2) is same as m0 (2)
//Passed Test 4 : Rank(NR) using QR (2) is same as m0 (2)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
//
//
```

8.6 examples/cpp/printmatrices.cpp

This example demonstrates how to access the matrices calculated by the library from C++

```
#include <iostream>
#include <libstructural.h>
#include <matrix.h>
#include <string>
#include <vector>

using namespace std;
using namespace LIB_STRUCTURAL;
using namespace LIB_LA;

void PrintLabeledMatrix (DoubleMatrix &oMatrix,
                                                vector<string> &rowLabels,
                                                vector<string> &colLabels)
{
        // return if we don't have the right dimensions
        if (oMatrix.numCols() != colLabels.size()
                || oMatrix.numRows() != rowLabels.size())
                return;

        // otherwise print
        vector<string>::iterator it;
        cout << "\t";
        // print column labels
        for (it = colLabels.begin(); it != colLabels.end(); it++)
                cout << *it << "\t";
        cout << endl;

        for (unsigned int i = 0; i < oMatrix.numRows(); i++)
        {
                cout << rowLabels[i] << "\t";
                for (unsigned int j = 0; j < oMatrix.numCols(); j++)
                        cout << oMatrix(i,j) << "\t";
                cout << endl;
        }

}

void PrintFullyReorderedStoichiometry(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getFullyReorderedStoichiometryMatrix();
        vector<string> sRows; vector<string> sCols;
        instance.getFullyReorderedStoichiometryMatrixLabels(sRows, sCols);
        cout << endl << "Fully Reordered Stoichiometry Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}

void PrintReorderedStoichiometry(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getReorderedStoichiometryMatrix();
        vector<string> sRows; vector<string> sCols;
        instance.getReorderedStoichiometryMatrixLabels(sRows, sCols);
        cout << endl << "Reordered Stoichiometry Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}

void PrintStoichiometry(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getStoichiometryMatrix();
        vector<string> sRows; vector<string> sCols;
        instance.getStoichiometryMatrixLabels(sRows, sCols);
        cout << endl << "Stoichiometry Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
```

```
}

void PrintKMatrix(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getKMatrix();
        vector<string> sRows; vector<string> sCols;
        instance.getKMatrixLabels(sRows, sCols);
        cout << endl << "K Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}
void PrintK0Matrix(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getK0Matrix();
        vector<string> sRows; vector<string> sCols;
        instance.getK0MatrixLabels(sRows, sCols);
        cout << endl << "K0 Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}
void PrintLinkMatrix(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getLinkMatrix();
        vector<string> sRows; vector<string> sCols;
        instance.getLinkMatrixLabels(sRows, sCols);
        cout << endl << "L Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}
void PrintL0Matrix(LibStructural &instance)
{
        DoubleMatrix *oMatrix = instance.getL0Matrix();
        vector<string> sRows; vector<string> sCols;
        instance.getL0MatrixLabels(sRows, sCols);
        cout << endl << "L0 Matrix" << endl;
        PrintLabeledMatrix(*oMatrix, sRows, sCols); cout << endl << endl;
}

// print a couple of matrices
void PrintMatrices(LibStructural &instance)
{
        PrintFullyReorderedStoichiometry(instance);
        PrintReorderedStoichiometry(instance);
        PrintStoichiometry(instance);
        PrintKMatrix(instance);
        PrintK0Matrix(instance);
        PrintLinkMatrix(instance);
        PrintL0Matrix(instance);

        // and so on for Gamma Matrix, NDC, NIC,
        // ColumnReorderedNr ...
}

int main(int argc, char*argv[])
{
        if (argc != 2)
        {
                cerr << "Need one argument, full path to SBML file." << endl;
                return -1;
        }

        // get an instance of the library
        LibStructural* instance = LibStructural::getInstance();

        // print out model overview
        cout << instance->loadSBMLFromFile(argv[1]);

        // print test details
        cout << instance->getTestDetails();
```

```
        PrintMatrices(*instance);

        return 0;
}


// Running the program for BorisEJB.xml (available from the SBW
// distribution) yields:
//
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//STRUCTURAL ANALYSIS MODULE : Results
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Size of Stochiometric Matrix: 8 x 10 (Rank is  5)
//Nonzero entries in Stochiometric Matrix: 20  (25% full)
//
//Independent Species (5) :
//MKK_P, MAPK_P, MKKK, MKK, MAPK
//
//Dependent Species (3) :
//MKK_PP, MKKK_P, MAPK_PP
//
//L0 : There are 3 dependencies. L0 is a 3x5 matrix.
//
//Conserved Entities
//1:  + MKK_P + MKK + MKK_PP
//2:  + MKKK + MKKK_P
//3:  + MAPK_P + MAPK + MAPK_PP
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//Developed by the Computational Systems Biology Group at Keck Graduate Institute
//and the Saurolab at the Bioengineering Departmant at  University of Washington.
//Contact : Frank T. Bergmann (fbergman@u.washington.edu) or Herbert M. Sauro.
//
//(previous authors) Ravishankar Rao Vallabhajosyula
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//
//Testing Validity of Conservation Laws.
//
//Passed Test 1 : Gamma*N = 0 (Zero matrix)
//Passed Test 2 : Rank(N) using SVD (5) is same as m0 (5)
//Passed Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
//Passed Test 4 : Rank(NR) using QR (5) is same as m0 (5)
//Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
//Passed Test 6 : N*K = 0 (Zero matrix)
//
//
//Reordered Stoichiometry Matrix
//        J0      J1      J2      J3      J4      J5      J6      J7      J8      J9
//MKK_P   0       0       1       -1      1       -1      0       0       0       0
//MAPK_P  0       0       0       0       0       0       1       -1      1       -1
//MKKK    -1      1       0       0       0       0       0       0       0       0
//MKK     0       0       -1      0       0       1       0       0       0       0
//MAPK    0       0       0       0       0       0       -1      0       0       1
//MKK_PP  0       0       0       1       -1      0       0       0       0       0
//MKKK_P  1       -1      0       0       0       0       0       0       0       0
//MAPK_PP 0       0       0       0       0       0       0       1       -1      0
//
//
//
//Stoichiometry Matrix
//        J0      J1      J2      J3      J4      J5      J6      J7      J8      J9
//MKKK    -1      1       0       0       0       0       0       0       0       0
//MKKK_P  1       -1      0       0       0       0       0       0       0       0
//MKK     0       0       -1      0       0       1       0       0       0       0
```

```
//MKK_P   0       0       1       -1      1       -1      0       0       0       0
//MKK_PP  0       0       0       1       -1      0       0       0       0       0
//MAPK    0       0       0       0       0       0       -1      0       0       1
//MAPK_P  0       0       0       0       0       0       1       -1      1       -1
//MAPK_PP 0       0       0       0       0       0       0       1       -1      0
//
//
//
//K Matrix
//        J1      J5      J9      J8      J4
//J1      1       0       0       0       0
//J5      0       1       0       0       0
//J9      0       0       1       0       0
//J8      0       0       0       1       0
//J4      0       0       0       0       1
//J0      1       0       0       0       0
//J2      0       1       0       0       0
//J6      0       0       1       0       0
//J3      0       0       0       0       1
//J7      0       0       0       1       0
//
//
//
//K0 Matrix
//        J1      J5      J9      J8      J4
//J0      1       0       0       0       0
//J2      0       1       0       0       0
//J6      0       0       1       0       0
//J3      0       0       0       0       1
//J7      0       0       0       1       0
//
//
//
//L Matrix
//        MKK_P   MAPK_P  MKKK    MKK     MAPK
//MKK_P   1       0       0       0       0
//MAPK_P  0       1       0       0       0
//MKKK    0       0       1       0       0
//MKK     0       0       0       1       0
//MAPK    0       0       0       0       1
//MKK_PP  -1      0       0       -1      0
//MKKK_P  0       0       -1      0       0
//MAPK_PP 0       -1      0       0       -1
//
//
//
//L0 Matrix
//        MKK_P   MAPK_P  MKKK    MKK     MAPK
//MKK_PP  -1      0       0       -1      0
//MKKK_P  0       0       -1      0       0
//MAPK_PP 0       -1      0       0       -1
```

Index