AlignmentCalculator

1.0.0

Generated by Doxygen 1.8.9.1

Wed May 25 2016 13:45:49

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

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 CONSTANTS Namespace Reference

Variables

- const double HBAR = 1.0

    hbar in atomic units
- const double EMASS = 1.0

    mass of an electron in atomic units
- const double ECHARGE = 1.0

    charge of an electron in atomic units
- const double VACPERM = (1.0/(4.0∗M_PI))

    Vacuum permitivity in atomic units.
- const double LEN = 0.0529177

    nanometers in an atomic unit of length
- const double VEL = 2.18e8

    1 atomic unit of velocity in cm/s
- const double EN = 27.21

    1 atomic unit of energy in eV
- const double TIME = 2.42e-17

    1 atomic unit of time in s
- const double AUperFS = 41.34137333656137

    atomic units of time in 1 fs
- const double FREQ = 4.13e16

    1 atomic unit of frequency in Hz
- const double ELECFIELD = 5.14e9

    1 atomic unit of electric field amplitude in V/cm^2
- const double LASERINTEN = 3.51e16

    1 atomic unit of intensity in W/cm^2 (0.5∗vacuum_permitivity∗speed_of_light∗EField^2)
- const double C = 2.998e10/VEL

    speed of light in atomic units
- const double BOLTZ = 3.1669e-6

    Boltzmann constant in atomic units (energy) per Kelvin.

5.1.1 Variable Documentation

5.1.1.1 const double CONSTANTS::AUperFS = 41.34137333656137

atomic units of time in 1 fs

5.1.1.2 const double CONSTANTS::BOLTZ = 3.1669e-6

Boltzmann constant in atomic units (energy) per Kelvin.

5.1.1.3 const double CONSTANTS::C = 2.998e10/VEL

speed of light in atomic units

5.1.1.4 const double CONSTANTS::ECHARGE = 1.0

charge of an electron in atomic units

5.1.1.5 const double CONSTANTS::ELECFIELD = 5.14e9

1 atomic unit of electric field amplitude in V/cm^2

5.1.1.6 const double CONSTANTS::EMASS = 1.0

mass of an electron in atomic units

5.1.1.7 const double CONSTANTS::EN = 27.21

1 atomic unit of energy in eV

5.1.1.8 const double CONSTANTS::FREQ = 4.13e16

1 atomic unit of frequency in Hz

5.1.1.9 const double CONSTANTS::HBAR = 1.0

hbar in atomic units

5.1.1.10 const double CONSTANTS::LASERINTEN = 3.51e16

1 atomic unit of intensity in W/cm^2 (0.5∗vacuum_permitivity∗speed_of_light∗EField^2)

5.1.1.11 const double CONSTANTS::LEN = 0.0529177

nanometers in an atomic unit of length

5.1.1.12 const double CONSTANTS::TIME = 2.42e-17

1 atomic unit of time in s

5.1.1.13 const double CONSTANTS::VACPERM = (1.0/(4.0∗M_PI))

Vacuum permitivity in atomic units.

5.1.1.14 const double CONSTANTS::VEL = 2.18e8

1 atomic unit of velocity in cm/s

Chapter 6

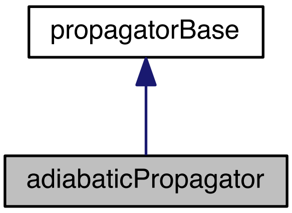
Class Documentation

6.1 adiabaticPropagator Class Reference

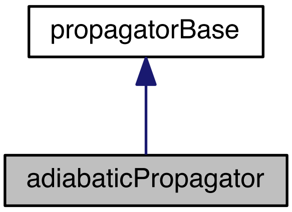
Adiabatic Calculation Propagator.

`#include <adiabaticPropagator.hpp>`

Inheritance diagram for adiabaticPropagator:



Collaboration diagram for adiabaticPropagator:



Public Member Functions

- adiabaticPropagator (inputParameters &IP)

  Constructor.
- void initializeOutputs (inputParameters &IP)

  output initializer
- void step ()

  calculation stepper
- void run ()

  run calculation
- void printOutputs ()

  print outputs

Public Attributes

- double intensity_

  Initial intensity and variable for holding the current field strength.
- double dItn_

  intensity increment
- double itn_final_

  Final intensity.
- std::string output_file_name_

  Output filename for observables.
- std::ofstream out_file_

  Output file stream.
- std::function< double(double)> intensity_stepper_

  Function to step intensity (addition and multiplication are currently supported)
- std::shared_ptr< arrays > eigenenergies_

  Storage for all eigenvalues at a particular intensity.

6.1.1 Detailed Description

Adiabatic Calculation Propagator.

Class for managing data and outputs for aligning molecules in an adiabatic field (constant intensity)

6.1.2 Constructor & Destructor Documentation

6.1.2.1 adiabaticPropagator::adiabaticPropagator (**inputParameters &** *IP*)

Constructor.

Constructor function requiring input parameters to initialize molecule data and outputs

Parameters

IP	input parameters

6.1.3 Member Function Documentation

6.1.3.1 void adiabaticPropagator::initializeOutputs (**inputParameters &** *IP*) `[virtual]`

output initializer

Creates output streams and matrix representations of all observables

Parameters

IP	input parameters

Implements propagatorBase.

6.1.3.2 void adiabaticPropagator::printOutputs ( ) [virtual]

print outputs

Calculates all observables and prints them to the output file

Implements propagatorBase.

6.1.3.3 void adiabaticPropagator::run ( )

run calculation

Run all intensities

6.1.3.4 void adiabaticPropagator::step ( )

calculation stepper

Evaluate the alignment for the current value of the intensity

6.1.4 Member Data Documentation

6.1.4.1 double adiabaticPropagator::dltn_

intensity increment

6.1.4.2 std::shared_ptr<arrays> adiabaticPropagator::eigenenergies_

Storage for all eigenvalues at a particular intensity.

6.1.4.3 double adiabaticPropagator::intensity_

Initial intensity and variable for holding the current field strength.

6.1.4.4 std::function<double(double)> adiabaticPropagator::intensity_stepper_

Function to step intensity (addition and multiplication are currently supported)

6.1.4.5 double adiabaticPropagator::itn_final_

Final intensity.

6.1.4.6 std::ofstream adiabaticPropagator::out_file_

Output file stream.

6.1.4.7 std::string adiabaticPropagator::output_file_name_

Output filename for observables.

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

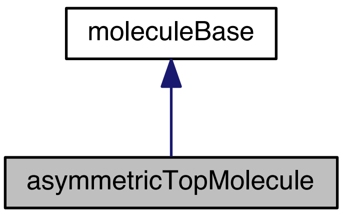
- src/adiabaticPropagator.hpp
- src/adiabaticPropagator.cpp

6.2 asymmetricTopMolecule Class Reference

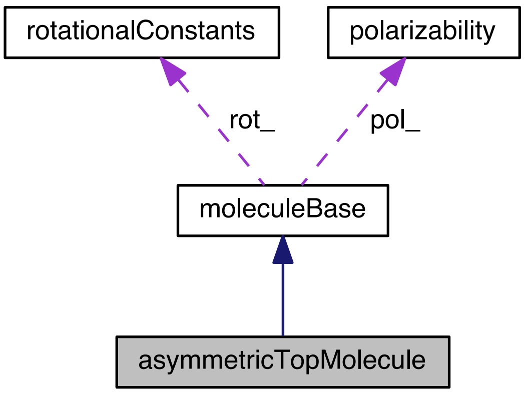
Asymmetric Top Molecules.

`#include <molecules.hpp>`

Inheritance diagram for asymmetricTopMolecule:



Collaboration diagram for asymmetricTopMolecule:



Public Member Functions

- asymmetricTopMolecule (inputParameters &IP)

- void constructTransformationMatrices (std::shared_ptr< matrices >)

  Create U and invU for basis set transformations.
- std::shared_ptr< basisSubsets > createBasisSets (int JMAX)

  create basis sets
- std::shared_ptr< matrices > createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates field-free rigid rotor Hamiltonians for the basis subsets provided.
- std::shared_ptr< arrays > initializePopulations (std::shared_ptr< basisSubsets >, std::shared_ptr< matrices >, double)

  Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.
- std::shared_ptr< matrices > initializeDensities (std::shared_ptr< arrays >)

  Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.
- std::shared_ptr< matrices > createInteractionHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Public Attributes

- double Xe_
- double Ye_
- double Ze_

  Placeholders to track coordinate system.

6.2.1 Detailed Description

Asymmetric Top Molecules.

Class derived from molecule base for asymmetric top molecules

Parameters

IP	Input parameters

6.2.2 Constructor & Destructor Documentation

6.2.2.1 asymmetricTopMolecule::asymmetricTopMolecule (inputParameters & *IP*)

6.2.3 Member Function Documentation

6.2.3.1 void asymmetricTopMolecule::constructTransformationMatrices (std::shared_ptr< matrices > *offDiagHamiltonians*)

Create U and invU for basis set transformations.

6.2.3.2 std::shared_ptr< basisSubsets > asymmetricTopMolecule::createBasisSets (int *JMAX*) [virtual]

create basis sets

Creates the basis subsets for the molecule based on selection rules

Parameters

JMAX	maximum value for J

Returns

pointer to basis subset array

Implements moleculeBase.

6.2.3.3 std::shared_ptr< matrices > asymmetricTopMolecule::createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > *sets*) [virtual]

Creates field-free rigid rotor Hamiltonians for the basis subsets provided.

Implements moleculeBase.

6.2.3.4 std::shared_ptr< matrices > asymmetricTopMolecule::createInteractionHamiltonians (std::shared_ptr< basisSubsets > *sets*) [virtual]

Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Implements moleculeBase.

6.2.3.5 std::shared_ptr< matrices > asymmetricTopMolecule::initializeDensities (std::shared_ptr< arrays >) [virtual]

Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.

Implements moleculeBase.

6.2.3.6 std::shared_ptr< arrays > asymmetricTopMolecule::initializePopulations (std::shared_ptr< basisSubsets > , std::shared_ptr< matrices > , double) [virtual]

Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.

< Avoids divide by zero error

Spin degeneracy statistics can be included here if need be

Implements moleculeBase.

6.2.4 Member Data Documentation

6.2.4.1 double asymmetricTopMolecule::Xe_

6.2.4.2 double asymmetricTopMolecule::Ye_

6.2.4.3 double asymmetricTopMolecule::Ze_

Placeholders to track coordinate system.

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

- src/molecules.hpp
- src/molecules.cpp

6.3 basis Struct Reference

Storage for the basis function |JKM> quantum numbers.

`#include <molecules.hpp>`

Public Member Functions

- basis (int, int, int)

Public Attributes

- int J

  Angular momentum quantum number.
- int K

  Projection of angular momentum onto body-fixed z axis.
- int M

  Projection of angular momentum onto space-fixed z axis.

6.3.1 Detailed Description

Storage for the basis function |JKM> quantum numbers.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 basis::basis (int *j*, int *k*, int *m*)

6.3.3 Member Data Documentation

6.3.3.1 int basis::J

Angular momentum quantum number.

6.3.3.2 int basis::K

Projection of angular momentum onto body-fixed z axis.

6.3.3.3 int basis::M

Projection of angular momentum onto space-fixed z axis.

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

- src/molecules.hpp
- src/molecules.cpp

6.4 inputParameters Class Reference

```
#include <inputs.hpp>
```

Public Attributes

Adiabatic Inputs

Input Parameters needed for adiabatic calculations.

- double initial_intensity_

  Initial intensity.
- double final_intensity_

  Final intensity.
- double intensity_increment_

  Intensity step value.
- bool add_increment_

Adds step rather than multiplies if true.
- bool output_density_

  Outputs probability density of ground state in theta and phi (chi = 0 by default)
- double chi_

  change chi to number other than zero
- bool output_eigenvectors_

  print eigenvector information (first 10)
- int n_eigenvalues_

Nonadiabatic Inputs

Input Parameters needed for nonadiabatic calculations.

- std::vector< pulse > pulses_

  List of laser pulses, all assumed to have same polarization.
- int n_outputs_

  Total number of data points to collect.
- double max_time_

  Time of nonadiabatic simulation.
- double atol_

  CVODE Absolute tolerance.
- double rtol_

  CVODE Relative tolerance.

General Inputs

Input Parameters needed for all calculations.

- JOBTYPE jobtype_

  Specifies nonadiabatic or adiabatic calculation.
- std::string filename_

  JSON file containing all input parameters.
- std::string molecule_name_

  Tag for output files, default is "Molecule".
- std::string library_file_

  Optional file containing polarizability and rotational constants.
- std::string library_molecule_

  Name of molecule in the library file.
- double rotational_temp_

  Initial rotational temperature.
- std::vector< double > rotational_constants_

  Array of one (linear) or three (asymmetric or symmetric) rotational constants.
- std::vector< double > polarizabilities_

  Three polarizability elements.
- double odd_j_degeneracy_

  Term to modify thermal distribution for bosonic or fermionic nuclei.
- double even_j_degeneracy_

  Same as odd_j_degeneracy.
- int max_j

  Maximum J state to include in calculation.
- bool output_basis_list_

  Outputs list of basis functions.
- bool output_coupling_matrix_

  Outputs couplings.

- bool output_cos2D_

  Outputs projection of cosine squared onto a unit disk.
- bool output_cos3D_

  Outputs cosine squared.
- bool output_cos3DAlt_

  Outputs chi squared expectation value.
- bool output_energy_

  Outputs total energy.
- bool output_J_

  Output expectation value for J quantum number.
- bool output_K_

  Output expectation value for K quantum number.
- bool output_M_

  Output expectation value for M quantum number.
- inputParameters (std::string infile)

  Parse inputs.
- void stripComments_ ()

  Strips all comments.
- void parseAllInputs_ ()

  Read input file into property tree.
- void parseJobType (boost::property_tree::ptree &)

  Gets jobtype.
- void parseMoleculeInfo (boost::property_tree::ptree &)

  Parses molecule information.
- void parseFieldInfo (boost::property_tree::ptree &)

  Parses field information.
- void parseNumericalParams (boost::property_tree::ptree &)

  Parses numerical parameters.
- void parseOutputsInfo (boost::property_tree::ptree &)

6.4.1 Constructor & Destructor Documentation

6.4.1.1 inputParameters::inputParameters (std::string *infile*)

Parse inputs.

Gets all calculation inputs from a json file

Parameters

infile	filename

6.4.2 Member Function Documentation

6.4.2.1 void inputParameters::parseAllInputs_ ( )

Read input file into property tree.

6.4.2.2 void inputParameters::parseFieldInfo (boost::property_tree::ptree & *IP*)

Parses field information.

6.4.2.3 void inputParameters::parseJobType (boost::property_tree::ptree & *IP*)

Gets jobtype.

6.4.2.4 void inputParameters::parseMoleculeInfo (boost::property_tree::ptree & *IP*)

Parses molecule information.

6.4.2.5 void inputParameters::parseNumericalParams (boost::property_tree::ptree & *IP*)

Parses numerical parameters.

6.4.2.6 void inputParameters::parseOutputsInfo (boost::property_tree::ptree & *IP*)

Parses output requests

6.4.2.7 void inputParameters::stripComments_ ( )

Strips all comments.

Removes all comments in input file that begin with '//'

6.4.3 Member Data Documentation

6.4.3.1 bool inputParameters::add_increment_

Adds step rather than multiplies if true.

6.4.3.2 double inputParameters::atol_

CVODE Absolute tolerance.

6.4.3.3 double inputParameters::chi_

change chi to number other than zero

6.4.3.4 double inputParameters::even_j_degeneracy_

Same as odd_j_degeneracy.

6.4.3.5 std::string inputParameters::filename_

JSON file containing all input parameters.

6.4.3.6 double inputParameters::final_intensity_

Final intensity.

6.4.3.7 double inputParameters::initial_intensity_

Initial intensity.

6.4.3.8 double inputParameters::intensity_increment_

Intensity step value.

6.4.3.9 JOBTYPE inputParameters::jobtype_

Specifies nonadiabatic or adiabatic calculation.

6.4.3.10 std::string inputParameters::library_file_

Optional file containing polarizability and rotational constants.

6.4.3.11 std::string inputParameters::library_molecule_

Name of molecule in the library file.

6.4.3.12 int inputParameters::max_j

Maximum J state to include in calculation.

6.4.3.13 double inputParameters::max_time_

Time of nonadiabatic simulation.

6.4.3.14 std::string inputParameters::molecule_name_

Tag for output files, default is "Molecule".

6.4.3.15 int inputParameters::n_eigenvalues_

number of eigenvalues to output

6.4.3.16 int inputParameters::n_outputs_

Total number of data points to collect.

6.4.3.17 double inputParameters::odd_j_degeneracy_

Term to modify thermal distribution for bosonic or fermionic nuclei.

6.4.3.18 bool inputParameters::output_basis_list_

Outputs list of basis functions.

6.4.3.19 bool inputParameters::output_cos2D_

Outputs projection of cosine squared onto a unit disk.

6.4.3.20 bool inputParameters::output_cos3D_

Outputs cosine squared.

6.4.3.21 bool inputParameters::output_cos3DAlt_

Outputs chi squared expectation value.

6.4.3.22 bool inputParameters::output_coupling_matrix_

Outputs couplings.

6.4.3.23 bool inputParameters::output_density_

Outputs probability density of ground state in theta and phi (chi = 0 by default)

6.4.3.24 bool inputParameters::output_eigenvectors_

print eigenvector information (first 10)

6.4.3.25 bool inputParameters::output_energy_

Outputs total energy.

6.4.3.26 bool inputParameters::output_J_

Output expectation value for J quantum number.

6.4.3.27 bool inputParameters::output_K_

Output expectation value for K quantum number.

6.4.3.28 bool inputParameters::output_M_

Output expectation value for M quantum number.

6.4.3.29 std::vector< double > inputParameters::polarizabilities_

Three polarizability elements.

6.4.3.30 std::vector< pulse > inputParameters::pulses_

List of laser pulses, all assumed to have same polarization.

6.4.3.31 std::vector<double> inputParameters::rotational_constants_

Array of one (linear) or three (asymmetric or symmetric) rotational constants.

6.4.3.32 double inputParameters::rotational_temp_

Initial rotational temperature.

6.4.3.33 double inputParameters::rtol_

CVODE Relative tolerance.

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

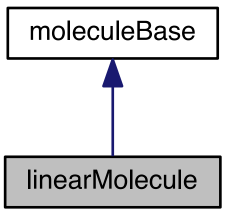
- src/inputs.hpp
- src/inputs.cpp

6.5 linearMolecule Class Reference

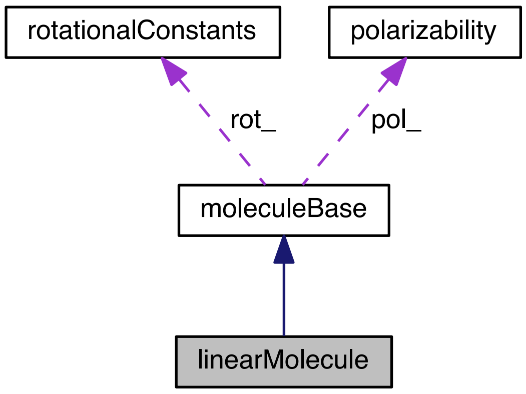
Linear Molecules.

`#include <molecules.hpp>`

Inheritance diagram for linearMolecule:



Collaboration diagram for linearMolecule:



Public Member Functions

- linearMolecule (inputParameters &IP)
- std::shared_ptr< basisSubsets > createBasisSets (int JMAX)

  create basis sets
- std::shared_ptr< matrices > createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates field-free rigid rotor Hamiltonians for the basis subsets provided.
- std::shared_ptr< arrays > initializePopulations (std::shared_ptr< basisSubsets >, std::shared_ptr< matrices >, double)

  Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.
- std::shared_ptr< matrices > initializeDensities (std::shared_ptr< arrays >)

  Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.
- std::shared_ptr< matrices > createInteractionHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Additional Inherited Members

6.5.1 Detailed Description

Linear Molecules.

Class derived from molecule base for linear molecules

Parameters

IP	Input parameters

6.5.2 Constructor & Destructor Documentation

6.5.2.1 linearMolecule::linearMolecule (inputParameters & *IP*)

6.5.3 Member Function Documentation

6.5.3.1 std::shared_ptr< **basisSubsets** > linearMolecule::createBasisSets (int *JMAX*) `[virtual]`

create basis sets

Creates the basis subsets for the molecule based on selection rules

Parameters

JMAX	maximum value for J

Returns

pointer to basis subset array

Implements moleculeBase.

6.5.3.2 std::shared_ptr< **matrices** > linearMolecule::createFieldFreeHamiltonians (std::shared_ptr< **basisSubsets** > *sets*) `[virtual]`

Creates field-free rigid rotor Hamiltonians for the basis subsets provided.

Implements moleculeBase.

6.5.3.3 std::shared_ptr< **matrices** > linearMolecule::createInteractionHamiltonians (std::shared_ptr< **basisSubsets** > *sets*) `[virtual]`

Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Implements moleculeBase.

6.5.3.4 std::shared_ptr< **matrices** > linearMolecule::initializeDensities (std::shared_ptr< **arrays** >) `[virtual]`

Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.

Implements moleculeBase.

6.5.3.5 std::shared_ptr< **arrays** > linearMolecule::initializePopulations (std::shared_ptr< **basisSubsets** > , std::shared_ptr< **matrices** > , double) `[virtual]`

Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.

< Avoids divide by zero error

Implements moleculeBase.

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

- src/molecules.hpp
- src/molecules.cpp

6.6 matrixBase< T > Class Template Reference

```
#include <matrix.hpp>
```

Public Member Functions

- matrixBase (const int nr, const int nc)

  Constructor.
- matrixBase (const matrixBase &o)

  Copy Constructor.
- matrixBase (matrixBase &&o)

  Move Constructor.
- ∼matrixBase ()

  Destructor.
- size_t size () const

  Returns the size of the matrix.
- T ∗ data ()

  Returns pointer to the beginning of the data array.
- const T ∗ data () const
- void zero ()

  Fill entire matrix with zeroes.
- size_t nr () const

  Return number of rows.
- size_t nc () const

  Return number of columns.
- void random ()

  Fill matrix with randomly generated numbers.
- T & element (const int row, const int col)

  Get or set array element at (row,col) position.
- const T & element (const int row, const int col) const

  Const version of previous function.
- T & operator() (const int row, const int col)

  Operator definition calling on the element function.
- const T & operator() (const int row, const int col) const

  Const version of previous function.
- void makeIdentity ()

  Set Diagonal elements to unity.
- T trace ()

  Compute the trace.
- void scale (const T a)
- void printMem () const

  Print memory usage for all matrices.
- template< class U >
  std::shared_ptr< U > getSub_impl (int r, int c, int nr, int nc) const

  Extract a portion of the matrix starting at element(r,c), get (nr x nc matrix)
- template< typename U >
  void setSub (int r, int c, U o)

  Place matrix o at position (r,c)

Protected Attributes

- size_t nrows
- size_t ncols

  Number of rows and columns in matrix.
- std::unique_ptr< T[ ]> vals

  Data containment array.

Static Protected Attributes

- static unsigned int memSize = 0

  Total memory allocated to all matrix objects.

Friends

- template<typename U >
  std::ostream & operator<< (std::ostream &out, const matrixBase< U > &o)

  Overload of the << operator to print out a portion of the matrix.

6.6.1 Constructor & Destructor Documentation

6.6.1.1 template<typename T> matrixBase< T >::matrixBase (const int *nr*, const int *nc*) [inline]

Constructor.

6.6.1.2 template<typename T> matrixBase< T >::matrixBase (const matrixBase< T > & *o*) [inline]

Copy Constructor.

6.6.1.3 template<typename T> matrixBase< T >::matrixBase (matrixBase< T > && *o*) [inline]

Move Constructor.

6.6.1.4 template<typename T> matrixBase< T >::∼matrixBase ( ) [inline]

Destructor.

6.6.2 Member Function Documentation

6.6.2.1 template<typename T> T∗ matrixBase< T >::data ( ) [inline]

Returns pointer to the beginning of the data array.

6.6.2.2 template<typename T> const T∗ matrixBase< T >::data ( ) const [inline]

6.6.2.3 template<typename T> T& matrixBase< T >::element (const int *row*, const int *col*) [inline]

Get or set array element at (row,col) position.

6.6.2.4 template<typename T> const T& matrixBase< T >::element (const int *row*, const int *col*) const [inline]

Const version of previous function.

6.6.2.5 template<typename T> template<class U > std::shared_ptr<U> matrixBase< T >::getSub_impl (int *r*, int *c*, int *nr*, int *nc*) const [inline]

Extract a portion of the matrix starting at element(r,c), get (nr x nc matrix)

6.6.2.6 template<typename T> void matrixBase< T >::makeIdentity ( ) [inline]

Set Diagonal elements to unity.

6.6.2.7 template<typename T> size_t matrixBase< T >::nc ( ) const [inline]

Return number of columns.

6.6.2.8 template<typename T> size_t matrixBase< T >::nr ( ) const [inline]

Return number of rows.

6.6.2.9 template<typename T> T& matrixBase< T >::operator() (const int *row*, const int *col*) [inline]

Operator definition calling on the element function.

6.6.2.10 template<typename T> const T& matrixBase< T >::operator() (const int *row*, const int *col*) const [inline]

Const version of previous function.

6.6.2.11 template<typename T> void matrixBase< T >::printMem ( ) const [inline]

Print memory usage for all matrices.

6.6.2.12 template<typename T> void matrixBase< T >::random ( ) [inline]

Fill matrix with randomly generated numbers.

6.6.2.13 template<typename T> void matrixBase< T >::scale (const T *a*) [inline]

6.6.2.14 template<typename T> template<typename U > void matrixBase< T >::setSub (int *r*, int *c*, U *o*) [inline]

Place matrix o at position (r,c)

6.6.2.15 template<typename T> size_t matrixBase< T >::size ( ) const [inline]

Returns the size of the matrix.

6.6.2.16 template<typename T> T matrixBase< T >::trace ( ) [inline]

Compute the trace.

6.6.2.17 template<typename T> void matrixBase< T >::zero ( ) [inline]

Fill entire matrix with zeroes.

6.6.3 Friends And Related Function Documentation

6.6.3.1 template<typename T> template<typename U > std::ostream& operator<< (std::ostream & *out*, const matrixBase< U > & *o*) `[friend]`

Overload of the << operator to print out a portion of the matrix.

6.6.4 Member Data Documentation

6.6.4.1 template<typename T> unsigned int **matrixBase**< T >::memSize = 0 `[static], [protected]`

Total memory allocated to all matrix objects.

6.6.4.2 template<typename T> size_t **matrixBase**< T >::ncols `[protected]`

Number of rows and columns in matrix.

6.6.4.3 template<typename T> size_t **matrixBase**< T >::nrows `[protected]`

6.6.4.4 template<typename T> std::unique_ptr<T[ ]> **matrixBase**< T >::vals `[protected]`

Data containment array.

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

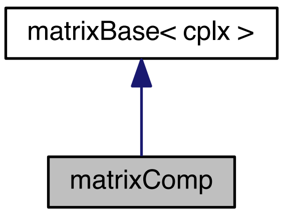
- src/matrix.hpp

6.7 matrixComp Class Reference

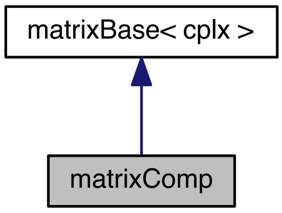
Matrix of complex numbers.

`#include <matrix.hpp>`

Inheritance diagram for matrixComp:



Collaboration diagram for matrixComp:



Public Member Functions

- matrixComp (const int nr, const int nc)

  Empty Matrix constructor.
- matrixComp (const matrixComp &)

  copy constructor
- matrixComp (matrixComp &&)

  move constructor
- matrixComp & operator= (const matrixComp &)
- matrixComp operator∗ (const matrixComp &) const
- matrixComp & operator∗= (const matrixComp &)
- matrixComp operator+ (const matrixComp &) const
- matrixComp & operator+= (const matrixComp &)
- matrixComp operator- (const matrixComp &) const
- matrixComp & operator-= (const matrixComp &)
- matrixComp operator| (const matrixComp &) const
- void getEigvals (double ∗eigVals)

  Calculate eigenvalues without eigenvectors.
- std::shared_ptr< matrixComp > transpose () const

  A^T
- matrixComp operator∗ (const cplx &) const

  Multiplication operator overload.
- matrixComp operator/ (const cplx &) const

  Division operator overload.
- matrixComp & operator∗= (const cplx &)

  in-place Multiplication operator overload
- matrixComp & operator/= (const cplx &)

  in-place Division operator overload
- std::shared_ptr< matrixComp > getSub (int ii, int jj, int kk, int ll) const

  Submatrix.
- void diagonalize (double ∗eigVals)

  Full Diagonalization with dsyev.
- void invert ()

  Calculate the inverse of the matrix in place.

Friends

- matrixComp matrixReal::operator∗ (const matrixComp &o) const

  Enables real∗complex matrix multiplication.

Additional Inherited Members

6.7.1 Detailed Description

Matrix of complex numbers.

Matrix base class using std::complex<double>, added mathematical funcitonality

6.7.2 Constructor & Destructor Documentation

6.7.2.1 matrixComp::matrixComp (const int *nr*, const int *nc*)

Empty Matrix constructor.

6.7.2.2 matrixComp::matrixComp (const **matrixComp** & *o*)

copy constructor

6.7.2.3 matrixComp::matrixComp (**matrixComp** && *o*)

move constructor

6.7.3 Member Function Documentation

6.7.3.1 void matrixComp::diagonalize (double ∗ *eigVals*)

Full Diagonalization with dsyev.

6.7.3.2 void matrixComp::getEigvals (double ∗ *eigVals*)

Calculate eigenvalues without eigenvectors.

6.7.3.3 std::shared_ptr<**matrixComp**> matrixComp::getSub (int *ii*, int *jj*, int *kk*, int *ll*) const [inline]

Submatrix.

Return matrix object containing a portion of the original

Parameters

ii	upper left row coordinate
jj	upper left column coordinate
kk	number of rows to copy
ll	number of columns to copy

Returns

new matrix

6.7.3.4 void matrixComp::invert ( )

Calculate the inverse of the matrix in place.

6.7.3.5 **matrixComp** matrixComp::operator$*$ (const **matrixComp** & *o*) const

6.7.3.6 **matrixComp** matrixComp::operator$*$ (const **cplx** & *a*) const

Multiplication operator overload.

6.7.3.7 **matrixComp** & matrixComp::operator$*$= (const **matrixComp** & *o*)

6.7.3.8 **matrixComp** & matrixComp::operator$*$= (const **cplx** & *a*)

in-place Multiplication operator overload

6.7.3.9 **matrixComp** matrixComp::operator+ (const **matrixComp** & *o*) const

6.7.3.10 **matrixComp** & matrixComp::operator+= (const **matrixComp** & *o*)

6.7.3.11 **matrixComp** matrixComp::operator- (const **matrixComp** & *o*) const

6.7.3.12 **matrixComp** & matrixComp::operator-= (const **matrixComp** & *o*)

6.7.3.13 **matrixComp** matrixComp::operator/ (const **cplx** & *a*) const

Division operator overload.

6.7.3.14 **matrixComp** & matrixComp::operator/= (const **cplx** & *a*)

in-place Division operator overload

6.7.3.15 **matrixComp** & matrixComp::operator= (const **matrixComp** & *o*)

6.7.3.16 **matrixComp** matrixComp::operator$|$ (const **matrixComp** & *o*) const

6.7.3.17 std::shared_ptr$<$ **matrixComp** $>$ matrixComp::transpose ( ) const

A^T

6.7.4 Friends And Related Function Documentation

6.7.4.1 **matrixComp** matrixReal::operator$*$ (const **matrixComp** & *o*) const [friend]

Enables real$*$complex matrix multiplication.

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

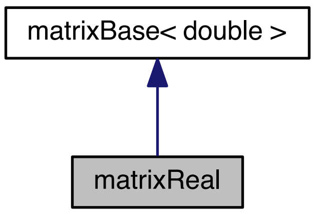
- src/matrix.hpp
- src/matrix.cpp

6.8 matrixReal Class Reference

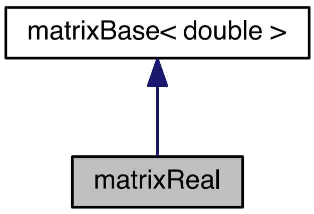
Matrix of real numbers.

```
#include <matrix.hpp>
```

Inheritance diagram for matrixReal:



Collaboration diagram for matrixReal:



Public Member Functions

- matrixReal (const int nr, const int nc)

  Default constructor.
- matrixReal (const matrixReal &)

  Copy constructor.
- matrixReal (matrixReal &&)

  Move constructor.

Matrix-Matrix Operations

Binary Operations accepting two real matricies

- matrixReal & operator= (const matrixReal &)

  $A = B$
- matrixReal operator$*$ (const matrixReal &) const

  $A * B$
- matrixReal & operator$*$= (const matrixReal &)

$A = (A * B)$

- matrixReal operator+ (const matrixReal &) const
  $A + B$
- matrixReal & operator+= (const matrixReal &)
  $A = (A + B)$
- matrixReal operator- (const matrixReal &) const
  $A - B$
- matrixReal & operator-= (const matrixReal &)
  $A = (A - B)$
- matrixReal operator| (const matrixReal &) const
  $A^T * B$
- matrixReal operator^ (const matrixReal &) const
  $A * B^T$

Matrix(Real)-Matrix(Complex) Operations

Binary Operations accepting two real matricies

- matrixComp operator∗ (const matrixComp &) const

Scalar-Matrix Operations

Binary Operations accepting a matrix and a constant, only rhs operators at the moment

- matrixReal operator∗ (const double &) const
  cA
- matrixReal operator/ (const double &) const
  $\frac{1}{c}A$
- matrixReal & operator∗= (const double &)
  $A = cA$
- matrixReal & operator/= (const double &)
  $A = \frac{1}{c}A$

BLAS and LAPACK

Other routines that require BLAS and LAPACK libraries to function, assumes symmetric matrices

- void diagonalize (double ∗eigVals)
  Full Diagonalization with dsyev.
- void diagonalize_alt (double ∗eigVals)
  Alternate diagonalization wuth dsyevd.
- void diagonalize (double ∗eigVals, bool getLowEigVal, int keepNum, double abstol)
  Partial diagonaliation returning lowest several eigenvectors, uses dsyevr.
- std::shared_ptr< matrixReal > transpose () const
  A^T
- std::tuple< std::shared_ptr< matrixReal >, std::shared_ptr< matrixReal > > svd (std::vector< double > &)
  Single Value Decomposition for non square matrices.

Other Operations

- double dot_product (const matrixReal &o) const
  $c = A \cdot B$
- double norm () const
  $|A| = \sqrt{A \cdot A}$
- double rms () const
  $\frac{|A|}{\sqrt{N}}$
- double variance () const
  $\frac{|A|^2}{N}$
- double operator% (const matrixReal &o) const
  Dot product for vectors with one column.

- matrixReal kron (matrixReal &o) const

  $A \otimes A$, very slow
- std::shared_ptr< matrixReal > getSub (int ii, int jj, int kk, int ll) const

  Returns pointer to sub matrix.
- void ax_plus_y (const double a, matrixReal &o)

  $cA + B$
- void invert ()

  Calculate the inverse of the matrix in-place.

Additional Inherited Members

6.8.1 Detailed Description

Matrix of real numbers.

Matrix base class using doubles, added mathematical funcitonality

6.8.2 Constructor & Destructor Documentation

6.8.2.1 matrixReal::matrixReal (const int *nr,* const int *nc*)

Default constructor.

6.8.2.2 matrixReal::matrixReal (const **matrixReal** & *o*)

Copy constructor.

6.8.2.3 matrixReal::matrixReal (**matrixReal** && *o*)

Move constructor.

6.8.3 Member Function Documentation

6.8.3.1 void matrixReal::ax_plus_y (const double *a,* **matrixReal** & *o*)

$cA + B$

6.8.3.2 void matrixReal::diagonalize (double $*$ *eigVals*)

Full Diagonalization with dsyev.

6.8.3.3 void matrixReal::diagonalize (double $*$ *eigVals,* bool *getLowEigVal,* int *keepNum,* double *abstol*)

Partial diagonaliation returning lowest several eigenvectors, uses dsyevr.

6.8.3.4 void matrixReal::diagonalize_alt (double $*$ *eigVals*)

Alternate diagonalization wuth dsyevd.

6.8.3.5 double matrixReal::dot_product (const **matrixReal** & *o*) const

$c = A \cdot B$

6.8.3.6 std::shared_ptr<matrixReal> matrixReal::getSub (int *ii*, int *jj*, int *kk*, int *ll*) const [inline]

Returns pointer to sub matrix.

6.8.3.7 void matrixReal::invert ( )

Calculate the inverse of the matrix in-place.

6.8.3.8 matrixReal matrixReal::kron (matrixReal & *o*) const

$A \otimes A$, very slow

6.8.3.9 double matrixReal::norm ( ) const

$|A| = \sqrt{A \cdot A}$

6.8.3.10 double matrixReal::operator% (const matrixReal & *o*) const

Dot product for vectors with one column.

6.8.3.11 matrixReal matrixReal::operator* (const matrixReal & *o*) const

$A * B$

6.8.3.12 matrixComp matrixReal::operator* (const matrixComp & *o*) const

$A * B$

6.8.3.13 matrixReal matrixReal::operator* (const double & *a*) const

cA

6.8.3.14 matrixReal & matrixReal::operator*= (const matrixReal & *o*)

$A = (A * B)$

6.8.3.15 matrixReal & matrixReal::operator*= (const double & *a*)

$A = cA$

6.8.3.16 matrixReal matrixReal::operator+ (const matrixReal & *o*) const

$A + B$

6.8.3.17 matrixReal & matrixReal::operator+= (const matrixReal & *o*)

$A = (A + B)$

6.8.3.18 matrixReal matrixReal::operator- (const matrixReal & *o*) const

$A - B$

6.8.3.19 matrixReal & matrixReal::operator-= (const matrixReal & *o*)

$A = (A - B)$

6.8.3.20 matrixReal matrixReal::operator/ (const double & *a*) const

$\frac{1}{c}A$

6.8.3.21 matrixReal & matrixReal::operator/= (const double & *a*)

$A = \frac{1}{c}A$

6.8.3.22 matrixReal & matrixReal::operator= (const matrixReal & *o*)

$A = B$

6.8.3.23 matrixReal matrixReal::operator^ (const matrixReal & *o*) const

$A * B^T$

6.8.3.24 matrixReal matrixReal::operator| (const matrixReal & *o*) const

$A^T * B$

6.8.3.25 double matrixReal::rms ( ) const

$\frac{|A|}{\sqrt{N}}$

6.8.3.26 tuple< shared_ptr< matrixReal >, shared_ptr< matrixReal > > matrixReal::svd (std::vector< double > &)

Single Value Decomposition for non square matrices.

6.8.3.27 std::shared_ptr< matrixReal > matrixReal::transpose ( ) const

A^T

6.8.3.28 double matrixReal::variance ( ) const

$\frac{|A|^2}{N}$

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

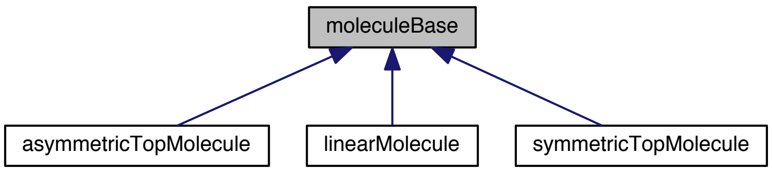
- src/matrix.hpp
- src/matrix.cpp

6.9 moleculeBase Class Reference

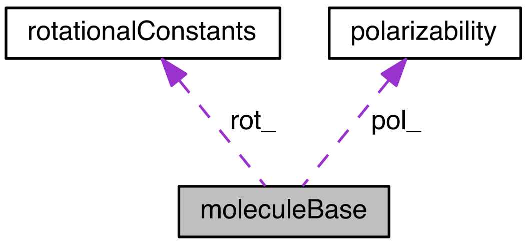
Molecule base class.

`#include <molecules.hpp>`

Inheritance diagram for moleculeBase:



Collaboration diagram for moleculeBase:



Public Member Functions

- moleculeBase ()

  Empty constructor.
- moleculeBase (inputParameters &)

  Constructor based on input parameters.
- virtual std::shared_ptr< basisSubsets > createBasisSets (int JMAX)=0

  create basis sets
- virtual std::shared_ptr< matrices > createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > sets)=0

  Creates field-free rigid rotor Hamiltonians for the basis subsets provided.
- virtual std::shared_ptr< arrays > initializePopulations (std::shared_ptr< basisSubsets >, std::shared_ptr< matrices >, double)=0

  Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.
- virtual std::shared_ptr< matrices > initializeDensities (std::shared_ptr< arrays >)=0

  Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.
- virtual std::shared_ptr< matrices > createInteractionHamiltonians (std::shared_ptr< basisSubsets > sets)=0

  Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Public Attributes

- double even_j_degen_

  partition function factor to account for nuclear spin degeneracy of even J states in linear molecules
- double odd_j_degen_

  partition function factor to account for nuclear spin degeneracy of odd J states in linear molecules
- double partition_function_

  Rotational partition function.
- polarizability pol_

  Polarizability components.
- rotationalConstants rot_

  Rotational constants.
- std::shared_ptr< matrices > Us_
- std::shared_ptr< matrices > invUs_

  Transformation matrices used if field-free Hamiltonian is not diagonal.

6.9.1 Detailed Description

Molecule base class.

Class for storing and generating molecule data common to all rigid molecule symmetries

6.9.2 Constructor & Destructor Documentation

6.9.2.1 moleculeBase::moleculeBase ( )

Empty constructor.

6.9.2.2 moleculeBase::moleculeBase (**inputParameters** & *IP*)

Constructor based on input parameters.

6.9.3 Member Function Documentation

6.9.3.1 virtual std::shared_ptr<**basisSubsets**> moleculeBase::createBasisSets (int *JMAX*) `[pure virtual]`

create basis sets

Creates the basis subsets for the molecule based on selection rules

Parameters

JMAX	maximum value for J

Returns

pointer to basis subset array

Implemented in asymmetricTopMolecule, symmetricTopMolecule, and linearMolecule.

6.9.3.2 virtual std::shared_ptr<**matrices**> moleculeBase::createFieldFreeHamiltonians (std::shared_ptr< **basisSubsets** > *sets*) `[pure virtual]`

Creates field-free rigid rotor Hamiltonians for the basis subsets provided.

Implemented in asymmetricTopMolecule, symmetricTopMolecule, and linearMolecule.

6.9.3.3 virtual std::shared_ptr<**matrices**> moleculeBase::createInteractionHamiltonians (std::shared_ptr< **basisSubsets** > *sets*) [pure virtual]

Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Implemented in asymmetricTopMolecule, symmetricTopMolecule, and linearMolecule.

6.9.3.4 virtual std::shared_ptr<**matrices**> moleculeBase::initializeDensities (std::shared_ptr< **arrays** >) [pure virtual]

Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.

Implemented in asymmetricTopMolecule, symmetricTopMolecule, and linearMolecule.

6.9.3.5 virtual std::shared_ptr<**arrays**> moleculeBase::initializePopulations (std::shared_ptr< **basisSubsets** > , std::shared_ptr< **matrices** > , double) [pure virtual]

Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.

Implemented in asymmetricTopMolecule, symmetricTopMolecule, and linearMolecule.

6.9.4 Member Data Documentation

6.9.4.1 double moleculeBase::even_j_degen_

partition function factor to account for nuclear spin degeneracy of even J states in linear molecules

6.9.4.2 std::shared_ptr<**matrices**> moleculeBase::invUs_

Transformation matrices used if field-free Hamiltonian is not diagonal.

6.9.4.3 double moleculeBase::odd_j_degen_

partition function factor to account for nuclear spin degeneracy of odd J states in linear molecules

6.9.4.4 double moleculeBase::partition_function_

Rotational partition function.

6.9.4.5 **polarizability** moleculeBase::pol_

Polarizability components.

6.9.4.6 **rotationalConstants** moleculeBase::rot_

Rotational constants.

6.9.4.7 std::shared_ptr<**matrices**> moleculeBase::Us_

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

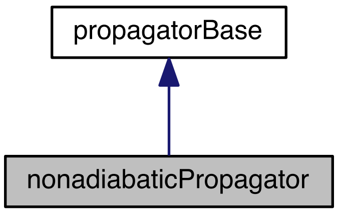
- src/molecules.hpp
- src/molecules.cpp

6.10 nonadiabaticPropagator Class Reference

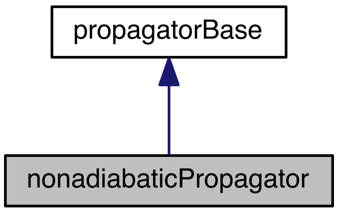
Nonadiabatic Calculation Propagator.

```
#include <nonadiabaticPropagator.hpp>
```

Inheritance diagram for nonadiabaticPropagator:



Collaboration diagram for nonadiabaticPropagator:



Public Member Functions

- nonadiabaticPropagator (inputParameters &IP)

  Constructor.
- void initializeCVODE (inputParameters &IP)

  Initialize all differential equation solvers.
- void initializeOutputs (inputParameters &IP)

  Initialize output streams.
- void step ()

  Step density matrices by time step.
- void run ()

  Run full simulation.

- void printOutputs ()

  Print all observables to output file.

Static Public Member Functions

- static int evalRHS (realtype t, N_Vector y, N_Vector ydot, void *user_data)

  Evaluate the right hand side function (Liouville von Neumann equation)

Public Attributes

- bool firstRun_

  Flag for propagator to identify if the calculation has run previously.
- int noutputs_

  Number of output times.
- int index_flag_

  Flag for passing information to the cvode propagator.
- double t0_

  Initial time.
- double dt_

  Time Step.
- double time_

  Current Time.
- double tFinal_

  Final Time.
- double atol_
- double rtol_

  absolute and relative error tolerances
- std::string output_file_name_

  Name of the output file.
- std::ofstream out_file_

  Output file stream.
- std::vector< pulse > pulses_

  vector containing all pulse objects (for calculating pulse trains)
- std::vector< N_Vector > atols_
- std::vector< N_Vector > ys_

  CVode tolerance and RHS vector storage.
- std::shared_ptr< matrices > scratch_matrices_
- std::shared_ptr< matrices > scratch_ydot_

  Useful scratch space to avoid reallocation of memory.
- std::vector< void * > cvode_managers_

  CVode objects.

6.10.1 Detailed Description

Nonadiabatic Calculation Propagator.

Class for managing data and outputs for aligning molecules in a nonadiabatic field (laser pulse)

6.10.2 Constructor & Destructor Documentation

6.10.2.1 nonadiabaticPropagator::nonadiabaticPropagator (inputParameters & *IP*)

Constructor.

6.10.3 Member Function Documentation

6.10.3.1 int nonadiabaticPropagator::evalRHS (realtype *t,* N_Vector *y,* N_Vector *ydot,* void $*$ *user_data*) `[static]`

Evaluate the right hand side function (Liouville von Neumann equation)

6.10.3.2 void nonadiabaticPropagator::initializeCVODE (inputParameters & *IP*)

Initialize all differential equation solvers.

6.10.3.3 void nonadiabaticPropagator::initializeOutputs (inputParameters & *IP*) `[virtual]`

Initialize output streams.

Implements propagatorBase.

6.10.3.4 void nonadiabaticPropagator::printOutputs ( ) `[virtual]`

Print all observables to output file.

Implements propagatorBase.

6.10.3.5 void nonadiabaticPropagator::run ( )

Run full simulation.

6.10.3.6 void nonadiabaticPropagator::step ( )

Step density matrices by time step.

Step using the boost diff eq libraries

Step using CVODE libraries

6.10.4 Member Data Documentation

6.10.4.1 double nonadiabaticPropagator::atol_

6.10.4.2 std::vector<N_Vector> nonadiabaticPropagator::atols_

6.10.4.3 std::vector<void$*$> nonadiabaticPropagator::cvode_managers_

CVode objects.

6.10.4.4 double nonadiabaticPropagator::dt_

Time Step.

6.10.4.5 bool nonadiabaticPropagator::firstRun_

Flag for propagator to identify if the calculation has run previously.

6.10.4.6 int nonadiabaticPropagator::index_flag_

Flag for passing information to the cvode propagator.

6.10.4.7 int nonadiabaticPropagator::noutputs_

Number of output times.

6.10.4.8 std::ofstream nonadiabaticPropagator::out_file_

Output file stream.

6.10.4.9 std::string nonadiabaticPropagator::output_file_name_

Name of the output file.

6.10.4.10 std::vector<pulse> nonadiabaticPropagator::pulses_

vector containing all pulse objects (for calculating pulse trains)

6.10.4.11 double nonadiabaticPropagator::rtol_

absolute and relative error tolerances

6.10.4.12 std::shared_ptr<matrices> nonadiabaticPropagator::scratch_matrices_

6.10.4.13 std::shared_ptr<matrices> nonadiabaticPropagator::scratch_ydot_

Useful scratch space to avoid reallocation of memory.

6.10.4.14 double nonadiabaticPropagator::t0_

Initial time.

6.10.4.15 double nonadiabaticPropagator::tFinal_

Final Time.

6.10.4.16 double nonadiabaticPropagator::time_

Current Time.

6.10.4.17 std::vector< N_Vector > nonadiabaticPropagator::ys_

CVode tolerance and RHS vector storage.

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

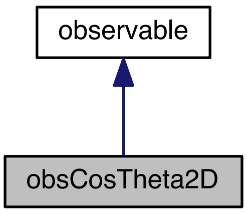
- src/nonadiabaticPropagator.hpp
- src/nonadiabaticPropagator.cpp

6.11 obsCosTheta2D Class Reference

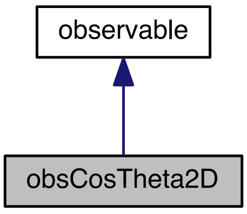
$\langle\cos^2\theta_{2D}\rangle$

```
#include <outputs.hpp>
```

Inheritance diagram for obsCosTheta2D:



Collaboration diagram for obsCosTheta2D:



Public Member Functions

- obsCosTheta2D (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)
- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

    Initialize the observable matrix.

Additional Inherited Members

6.11.1 Detailed Description

$\langle \cos^2 \theta_{2D} \rangle$

6.11.2 Constructor & Destructor Documentation

6.11.2.1 obsCosTheta2D::obsCosTheta2D (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

6.11.3 Member Function Documentation

6.11.3.1 void obsCosTheta2D::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) `[virtual]`

Initialize the observable matrix.

Implements observable.

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

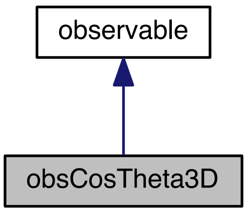
- src/outputs.hpp
- src/outputs.cpp

6.12 obsCosTheta3D Class Reference

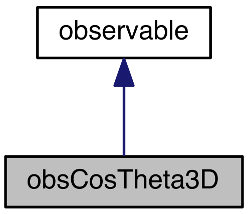
$\langle \cos^2 \theta_{3D} \rangle = \langle \cos^2(\hat{z} \cdot \hat{Z}) \rangle$

`#include <outputs.hpp>`

Inheritance diagram for obsCosTheta3D:



Collaboration diagram for obsCosTheta3D:



Public Member Functions

- obsCosTheta3D (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)
- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

  Initialize the observable matrix.

Additional Inherited Members

6.12.1 Detailed Description

$\langle\cos^2\theta_{3D}\rangle = \langle\cos^2(\hat{z}\cdot\hat{Z})\rangle$

6.12.2 Constructor & Destructor Documentation

6.12.2.1 obsCosTheta3D::obsCosTheta3D (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

6.12.3 Member Function Documentation

6.12.3.1 void obsCosTheta3D::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) [virtual]

Initialize the observable matrix.

Implements observable.

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

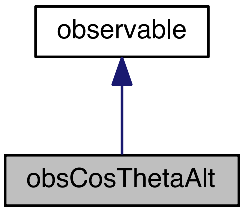
- src/outputs.hpp
- src/outputs.cpp

6.13 obsCosThetaAlt Class Reference

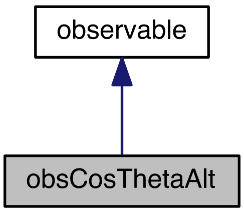
$\langle\cos^2\theta'_{3D}\rangle = \langle\cos^2(\hat{x}\cdot\hat{Z})\rangle$

```
#include <outputs.hpp>
```

Inheritance diagram for obsCosThetaAlt:



Collaboration diagram for obsCosThetaAlt:



Public Member Functions

- obsCosThetaAlt (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)
- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

  Initialize the observable matrix.

Additional Inherited Members

6.13.1 Detailed Description

$\langle\cos^2\theta'_{3D}\rangle = \langle\cos^2(\hat{x}\cdot\hat{Z})\rangle$

6.13.2 Constructor & Destructor Documentation

6.13.2.1 obsCosThetaAlt::obsCosThetaAlt (std::shared_ptr< basisSubsets > *basisSets*, std::shared_ptr< matrices > *fieldFreeHamiltonians*)

6.13.3 Member Function Documentation

6.13.3.1 void obsCosThetaAlt::initialize_ (std::shared_ptr< basisSubsets > *basisSets*, std::shared_ptr< matrices > *fieldFreeHamiltonians*) [virtual]

Initialize the observable matrix.

Implements observable.

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

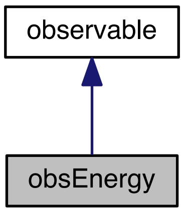
- src/outputs.hpp
- src/outputs.cpp

6.14 obsEnergy Class Reference

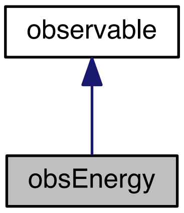
$\langle \hat{H} \rangle$

`#include <outputs.hpp>`

Inheritance diagram for obsEnergy:



Collaboration diagram for obsEnergy:



Public Member Functions

- obsEnergy (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

  Initialize the observable matrix.

Additional Inherited Members

6.14.1 Detailed Description

$\langle \hat{H} \rangle$

6.14.2 Constructor & Destructor Documentation

6.14.2.1 obsEnergy::obsEnergy (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

6.14.3 Member Function Documentation

6.14.3.1 void obsEnergy::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) [virtual]

Initialize the observable matrix.

Implements observable.

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

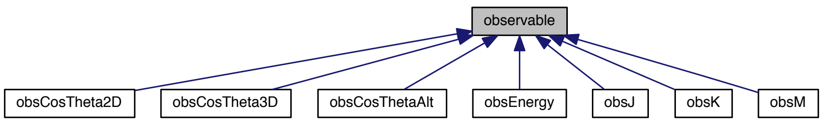
- src/outputs.hpp
- src/outputs.cpp

6.15 observable Class Reference

Base class for observables that are obtained from a density matrix as Tr(O∗rho). Other outputs are obtained from other members of the propagator class, such as the wavefunction density or the list of basis states used in the calculation.

`#include <outputs.hpp>`

Inheritance diagram for observable:



Public Member Functions

- observable (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

  Constructor.
- virtual void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)=0

  Initialize the observable matrix.
- virtual double density_evaluate_ (std::shared_ptr< matrices > densities_)

  Calculate expectation value given a density matrix.
- virtual double wvfxn_evaluate_ (std::shared_ptr< matrices > densities_, std::shared_ptr< arrays > populations_)

  Calculate expectation value given a set of wavefunctions stored in a matrix.

Public Attributes

- std::string id_tag_

  Unique identifier tag for printing to first line of output file.
- std::shared_ptr< matrices > operator_matrix_

  Matrix representation of observable.

6.15.1 Detailed Description

Base class for observables that are obtained from a density matrix as Tr(O∗rho). Other outputs are obtained from other members of the propagator class, such as the wavefunction density or the list of basis states used in the calculation.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 observable::observable (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

Constructor.

6.15.3 Member Function Documentation

6.15.3.1 double observable::density_evaluate_ (std::shared_ptr< **matrices** > *densities_*) `[virtual]`

Calculate expectation value given a density matrix.

6.15.3.2 virtual void observable::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) `[pure virtual]`

Initialize the observable matrix.

Implemented in obsM, obsK, obsJ, obsCosThetaAlt, obsEnergy, obsCosTheta2D, and obsCosTheta3D.

6.15.3.3 double observable::wvfxn_evaluate_ (std::shared_ptr< **matrices** > *densities_,* std::shared_ptr< **arrays** > *populations_*) `[virtual]`

Calculate expectation value given a set of wavefunctions stored in a matrix.

6.15.4 Member Data Documentation

6.15.4.1 std::string observable::id_tag_

Unique identifier tag for printing to first line of output file.

6.15.4.2 std::shared_ptr<**matrices**> observable::operator_matrix_

Matrix representation of observable.

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

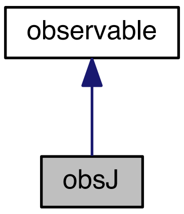
- src/outputs.hpp
- src/outputs.cpp

6.16 obsJ Class Reference

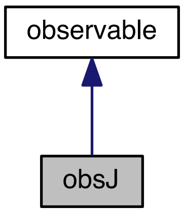
$\langle J^2 \rangle$

```
#include <outputs.hpp>
```

Inheritance diagram for obsJ:



Collaboration diagram for obsJ:



Public Member Functions

- obsJ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFreeHamiltonians)
- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree↩ Hamiltonians)

    Initialize the observable matrix.

Additional Inherited Members

6.16.1 Detailed Description

$\langle J^2 \rangle$

6.16.2 Constructor & Destructor Documentation

6.16.2.1 obsJ::obsJ (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

6.16.3 Member Function Documentation

6.16.3.1 void obsJ::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets*, std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) `[virtual]`

Initialize the observable matrix.

Implements observable.

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

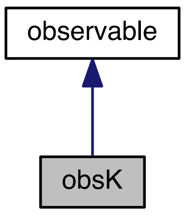
- src/outputs.hpp
- src/outputs.cpp

6.17 obsK Class Reference

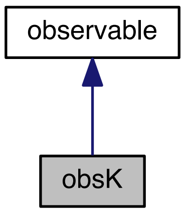
$\langle K^2 \rangle$

`#include <outputs.hpp>`

Inheritance diagram for obsK:



Collaboration diagram for obsK:



Public Member Functions

- obsK (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFreeHamiltonians)
- void initialize_ (std::shared_ptr< basisSubsets > basisSets, std::shared_ptr< matrices > fieldFree←Hamiltonians)

  Initialize the observable matrix.

Additional Inherited Members

6.17.1 Detailed Description

$\langle K^2 \rangle$

6.17.2 Constructor & Destructor Documentation

6.17.2.1 obsK::obsK (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*)

6.17.3 Member Function Documentation

6.17.3.1 void obsK::initialize_ (std::shared_ptr< **basisSubsets** > *basisSets,* std::shared_ptr< **matrices** > *fieldFreeHamiltonians*) `[virtual]`

Initialize the observable matrix.

Implements observable.

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

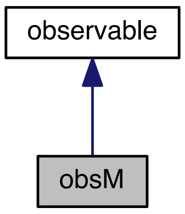
- src/outputs.hpp
- src/outputs.cpp

6.18 obsM Class Reference

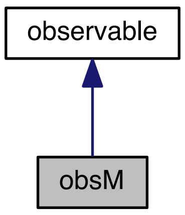
$\langle M \rangle$

`#include <outputs.hpp>`

Inheritance diagram for obsM:



Collaboration diagram for obsM:



Public Member Functions

- obsM (std::shared_ptr$<$ basisSubsets $>$ basisSets, std::shared_ptr$<$ matrices $>$ fieldFreeHamiltonians)
- void initialize_ (std::shared_ptr$<$ basisSubsets $>$ basisSets, std::shared_ptr$<$ matrices $>$ fieldFree↩ Hamiltonians)

    Initialize the observable matrix.

Additional Inherited Members

6.18.1 Detailed Description

$\langle M \rangle$

6.18.2 Constructor & Destructor Documentation

6.18.2.1 obsM::obsM (std::shared_ptr$<$ **basisSubsets** $>$ *basisSets,* std::shared_ptr$<$ **matrices** $>$ *fieldFreeHamiltonians*)

6.18.3 Member Function Documentation

6.18.3.1 void obsM::initialize_ (std::shared_ptr< basisSubsets > *basisSets*, std::shared_ptr< matrices > *fieldFreeHamiltonians*) [virtual]

Initialize the observable matrix.

Implements observable.

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

- src/outputs.hpp
- src/outputs.cpp

6.19 polarizability Struct Reference

Container class for diagonal polarizability components.

```
#include <molecules.hpp>
```

Public Attributes

- double aXX_
- double aYY_
- double aZZ_

6.19.1 Detailed Description

Container class for diagonal polarizability components.

6.19.2 Member Data Documentation

6.19.2.1 double polarizability::aXX_

6.19.2.2 double polarizability::aYY_

6.19.2.3 double polarizability::aZZ_

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

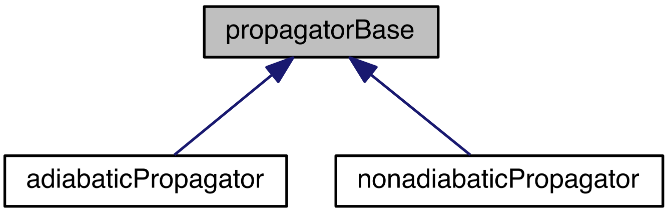
- src/molecules.hpp

6.20 propagatorBase Class Reference

Base class for adiabatic and nonadiabatic calculations.

```
#include <propagatorBase.hpp>
```

Inheritance diagram for propagatorBase:



Public Member Functions

- propagatorBase ()

    Default constructor, no real functionality.
- propagatorBase (inputParameters &IP)

    Recommended constructor.
- void outputBasisStats ()

    Outputs basis information to file.
- MOLSYM determineSymmetry (inputParameters &IP)
- void initialize_ ()
- virtual void initializeOutputs (inputParameters &IP)=0
- virtual void printOutputs ()=0
- virtual void transformObservables ()
- void removeSmallPopulations ()

Public Attributes

- double partition_function_

    Partition function for the initial state.
- double temperature_

    Initial temperature.
- MOLSYM symmetry_

    Symmetry of the molecule determining the coordinate system.
- std::shared_ptr< moleculeBase > molecule_

    molecule object
- std::shared_ptr< basisSubsets > basisSets_

    |JKM> states arranged in symmetry coupled subsets
- std::shared_ptr< matrices > fieldFreeHamiltonians_

    Rigid rotor Hamiltonian matrices.
- std::shared_ptr< matrices > intHamiltonians_

    Interaction Hamiltonian prefactors (i.e. not including field strength)
- std::shared_ptr< matrices > densities_

    Density matrix storage space.
- std::shared_ptr< arrays > populations_

    Boltzmann population for corresponding to the basis states.
- std::vector< std::shared_ptr< observable > > observables_

    Observables to be calculated during propagation.

6.20.1 Detailed Description

Base class for adiabatic and nonadiabatic calculations.

Class for managing setup of calculations and outputting data containing functionality universal to all jobtypes

6.20.2 Constructor & Destructor Documentation

6.20.2.1 propagatorBase::propagatorBase ( )

Default constructor, no real functionality.

6.20.2.2 propagatorBase::propagatorBase (**inputParameters** & *IP*)

Recommended constructor.

6.20.3 Member Function Documentation

6.20.3.1 **MOLSYM** propagatorBase::determineSymmetry (**inputParameters** & *IP*)

6.20.3.2 void propagatorBase::initialize_ ( )

6.20.3.3 virtual void propagatorBase::initializeOutputs (**inputParameters** & *IP*) `[pure virtual]`

Implemented in nonadiabaticPropagator, and adiabaticPropagator.

6.20.3.4 void propagatorBase::outputBasisStats ( )

Outputs basis information to file.

Prints file at end of propagation including a list of all basis states, energies, and thermal populations

6.20.3.5 virtual void propagatorBase::printOutputs ( ) `[pure virtual]`

Implemented in adiabaticPropagator, and nonadiabaticPropagator.

6.20.3.6 void propagatorBase::removeSmallPopulations ( )

6.20.3.7 void propagatorBase::transformObservables ( ) `[virtual]`

6.20.4 Member Data Documentation

6.20.4.1 std::shared_ptr< **basisSubsets** > propagatorBase::basisSets_

|JKM> states arranged in symmetry coupled subsets

6.20.4.2 std::shared_ptr< **matrices** > propagatorBase::densities_

Density matrix storage space.

6.20.4.3 std::shared_ptr<matrices> propagatorBase::fieldFreeHamiltonians_

Rigid rotor Hamiltonian matrices.

6.20.4.4 std::shared_ptr<matrices> propagatorBase::intHamiltonians_

Interaction Hamiltonian prefactors (i.e. not including field strength)

6.20.4.5 std::shared_ptr<moleculeBase> propagatorBase::molecule_

molecule object

6.20.4.6 std::vector<std::shared_ptr<observable> > propagatorBase::observables_

Observables to be calculated during propagation.

6.20.4.7 double propagatorBase::partition_function_

Partition function for the initial state.

6.20.4.8 std::shared_ptr<arrays> propagatorBase::populations_

Boltzmann population for corresponding to the basis states.

6.20.4.9 MOLSYM propagatorBase::symmetry_

Symmetry of the molecule determining the coordinate system.

6.20.4.10 double propagatorBase::temperature_

Initial temperature.

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

- src/propagatorBase.hpp
- src/propagatorBase.cpp

6.21 pulse Class Reference

```
#include <pulses.hpp>
```

Public Member Functions

- pulse (double, double, double)
- double evaluate (double)

Public Attributes

- double peakIntensity_
- double sigma_
- double t0_

6.21.1 Constructor & Destructor Documentation

6.21.1.1 pulse::pulse (double *I*, double *s*, double *t*)

6.21.2 Member Function Documentation

6.21.2.1 double pulse::evaluate (double *t*)

6.21.3 Member Data Documentation

6.21.3.1 double pulse::peakIntensity_

6.21.3.2 double pulse::sigma_

6.21.3.3 double pulse::t0_

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

- src/pulses.hpp
- src/pulses.cpp

6.22 rotationalConstants Struct Reference

Container for rotational constants.

`#include <molecules.hpp>`

Public Attributes

- double Ae_
- double Be_
- double Ce_

6.22.1 Detailed Description

Container for rotational constants.

6.22.2 Member Data Documentation

6.22.2.1 double rotationalConstants::Ae_

6.22.2.2 double rotationalConstants::Be_

6.22.2.3 double rotationalConstants::Ce_

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

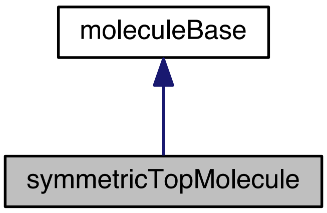
- src/molecules.hpp

6.23 symmetricTopMolecule Class Reference

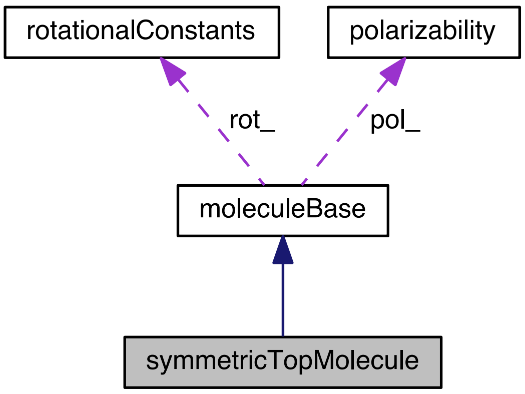
Symmetric Top Molecules.

`#include <molecules.hpp>`

Inheritance diagram for symmetricTopMolecule:



Collaboration diagram for symmetricTopMolecule:



Public Member Functions

- symmetricTopMolecule (inputParameters &IP)
- std::shared_ptr< basisSubsets > createBasisSets (int JMAX)

  create basis sets
- std::shared_ptr< matrices > createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates field-free rigid rotor Hamiltonians for the basis subsets provided.
- std::shared_ptr< arrays > initializePopulations (std::shared_ptr< basisSubsets >, std::shared_ptr< matrices >, double)

  Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.
- std::shared_ptr< matrices > initializeDensities (std::shared_ptr< arrays >)

Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.

- std::shared_ptr< matrices > createInteractionHamiltonians (std::shared_ptr< basisSubsets > sets)

  Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Public Attributes

- MOLSYM symmetry_

  Used to differentiate oblate from prolate tops.

6.23.1 Detailed Description

Symmetric Top Molecules.

Class derived from molecule base for symmetric top molecules

Parameters

IP	Input parameters

6.23.2 Constructor & Destructor Documentation

6.23.2.1 symmetricTopMolecule::symmetricTopMolecule (inputParameters & *IP*)

6.23.3 Member Function Documentation

6.23.3.1 std::shared_ptr< basisSubsets > symmetricTopMolecule::createBasisSets (int *JMAX*) `[virtual]`

create basis sets

Creates the basis subsets for the molecule based on selection rules

Parameters

JMAX	maximum value for J

Returns

pointer to basis subset array

Implements moleculeBase.

6.23.3.2 std::shared_ptr< matrices > symmetricTopMolecule::createFieldFreeHamiltonians (std::shared_ptr< basisSubsets > *sets*) `[virtual]`

Creates field-free rigid rotor Hamiltonians for the basis subsets provided.

Implements moleculeBase.

6.23.3.3 std::shared_ptr< matrices > symmetricTopMolecule::createInteractionHamiltonians (std::shared_ptr< basisSubsets > *sets*) `[virtual]`

Creates the interaction Hamiltonian prefactors (i.e. all terms except the field intensity)

Implements moleculeBase.

6.23.3.4 std::shared_ptr< **matrices** > symmetricTopMolecule::initializeDensities (std::shared_ptr< **arrays** >) [virtual]

Creates a set of matrices to be used as density matrix storage or as scratch space with population data placed on the diagonals.

Implements moleculeBase.

6.23.3.5 std::shared_ptr< **arrays** > symmetricTopMolecule::initializePopulations (std::shared_ptr< **basisSubsets** >, std::shared_ptr< **matrices** >, double) [virtual]

Calculates thermal populations and partition function based on the field free Hamiltonian and basis set information.

< Avoids divide by zero error

Spin degeneracy statistics can be included here if need be

Implements moleculeBase.

6.23.4 Member Data Documentation

6.23.4.1 MOLSYM symmetricTopMolecule::symmetry_

Used to differentiate oblate from prolate tops.

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

- src/molecules.hpp
- src/molecules.cpp

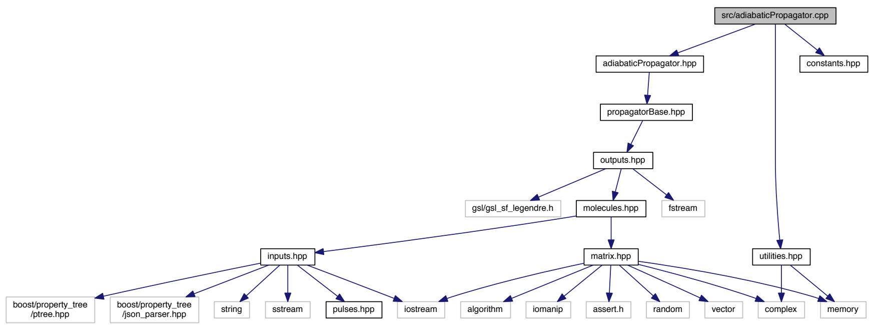
Chapter 7

File Documentation

7.1 src/adiabaticPropagator.cpp File Reference

```
#include "adiabaticPropagator.hpp"
#include "utilities.hpp"
#include "constants.hpp"
```

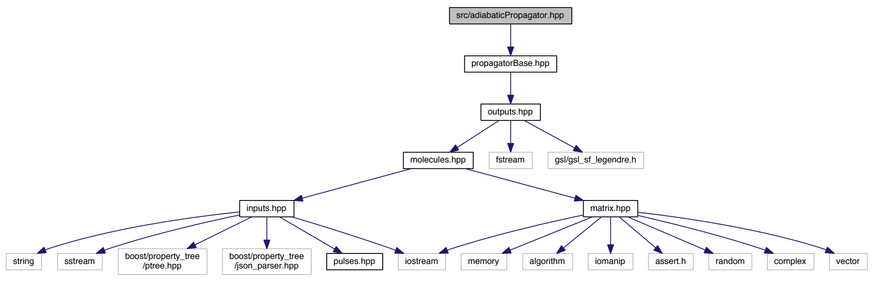
Include dependency graph for adiabaticPropagator.cpp:



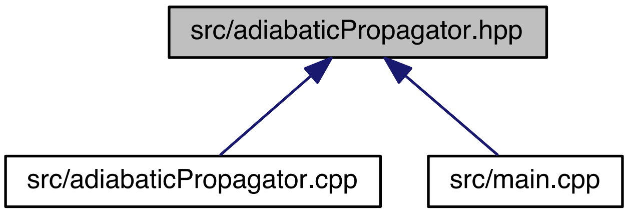
7.2 src/adiabaticPropagator.hpp File Reference

```
#include "propagatorBase.hpp"
```

Include dependency graph for adiabaticPropagator.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class adiabaticPropagator

    Adiabatic Calculation Propagator.

7.2.1 Detailed Description

Author

J. Szekely

7.3 src/alignmentcalculator_config.h File Reference

Macros

- #define HAVE_BOOST
- #define HAVE_CXX11 1
- #define HAVE_INTTYPES_H 1
- #define HAVE_LIBBOOST_FILESYSTEM_MT 1
- #define HAVE_LIBBOOST_SYSTEM_MT 1
- #define HAVE_LIBGSL 1
- #define HAVE_LIBGSLCBLAS 1
- #define HAVE_LIBSUNDIALS_CVODE 1
- #define HAVE_LIBSUNDIALS_NVECSERIAL 1
- #define HAVE_MEMORY_H 1
- #define HAVE_MKL_H 1
- #define HAVE_STDINT_H 1
- #define HAVE_STDLIB_H 1
- #define HAVE_STRINGS_H 1
- #define HAVE_STRING_H 1
- #define HAVE_SYS_STAT_H 1
- #define HAVE_SYS_TYPES_H 1
- #define HAVE_UNISTD_H 1
- #define PACKAGE "alignmentcalculator"
- #define PACKAGE_BUGREPORT "jeszekely@gmail.com"
- #define PACKAGE_NAME "AlignmentCalculator"
- #define PACKAGE_STRING "AlignmentCalculator 1.0"

- #define PACKAGE_TARNAME "alignmentcalculator"
- #define PACKAGE_URL ""
- #define PACKAGE_VERSION "1.0"
- #define STDC_HEADERS 1
- #define VERSION "1.0"

7.3.1 Macro Definition Documentation

7.3.1.1 #define HAVE_BOOST

7.3.1.2 #define HAVE_CXX11 1

7.3.1.3 #define HAVE_INTTYPES_H 1

7.3.1.4 #define HAVE_LIBBOOST_FILESYSTEM_MT 1

7.3.1.5 #define HAVE_LIBBOOST_SYSTEM_MT 1

7.3.1.6 #define HAVE_LIBGSL 1

7.3.1.7 #define HAVE_LIBGSLCBLAS 1

7.3.1.8 #define HAVE_LIBSUNDIALS_CVODE 1

7.3.1.9 #define HAVE_LIBSUNDIALS_NVECSERIAL 1

7.3.1.10 #define HAVE_MEMORY_H 1

7.3.1.11 #define HAVE_MKL_H 1

7.3.1.12 #define HAVE_STDINT_H 1

7.3.1.13 #define HAVE_STDLIB_H 1

7.3.1.14 #define HAVE_STRING_H 1

7.3.1.15 #define HAVE_STRINGS_H 1

7.3.1.16 #define HAVE_SYS_STAT_H 1

7.3.1.17 #define HAVE_SYS_TYPES_H 1

7.3.1.18 #define HAVE_UNISTD_H 1

7.3.1.19 #define PACKAGE "alignmentcalculator"

7.3.1.20 #define PACKAGE_BUGREPORT "jeszekely@gmail.com"

7.3.1.21 #define PACKAGE_NAME "AlignmentCalculator"

7.3.1.22 #define PACKAGE_STRING "AlignmentCalculator 1.0"

7.3.1.23 #define PACKAGE_TARNAME "alignmentcalculator"

7.3.1.24 #define PACKAGE_URL ""

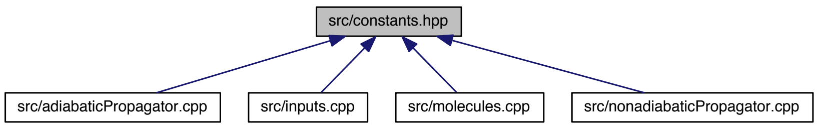
7.3.1.25 #define PACKAGE_VERSION "1.0"

7.3.1.26 #define STDC_HEADERS 1

7.3.1.27 #define VERSION "1.0"

7.4 src/constants.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- CONSTANTS

Variables

- const double CONSTANTS::HBAR = 1.0

    hbar in atomic units
- const double CONSTANTS::EMASS = 1.0

    mass of an electron in atomic units
- const double CONSTANTS::ECHARGE = 1.0

    charge of an electron in atomic units
- const double CONSTANTS::VACPERM = (1.0/(4.0∗M_PI))

    Vacuum permitivity in atomic units.
- const double CONSTANTS::LEN = 0.0529177

    nanometers in an atomic unit of length
- const double CONSTANTS::VEL = 2.18e8

    1 atomic unit of velocity in cm/s
- const double CONSTANTS::EN = 27.21

    1 atomic unit of energy in eV
- const double CONSTANTS::TIME = 2.42e-17

    1 atomic unit of time in s
- const double CONSTANTS::AUperFS = 41.34137333656137

    atomic units of time in 1 fs
- const double CONSTANTS::FREQ = 4.13e16

    1 atomic unit of frequency in Hz
- const double CONSTANTS::ELECFIELD = 5.14e9

    1 atomic unit of electric field amplitude in V/cm^2
- const double CONSTANTS::LASERINTEN = 3.51e16

    1 atomic unit of intensity in W/cm^2 (0.5∗vacuum_permitivity∗speed_of_light∗EField^2)
- const double CONSTANTS::C = 2.998e10/VEL

    speed of light in atomic units
- const double CONSTANTS::BOLTZ = 3.1669e-6

    Boltzmann constant in atomic units (energy) per Kelvin.

7.4.1 Detailed Description

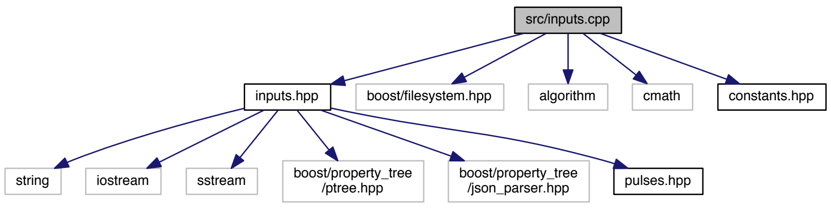
Author

J. Szekely

7.5 src/inputs.cpp File Reference

```
#include "inputs.hpp"
#include <boost/filesystem.hpp>
#include <algorithm>
#include <cmath>
#include "constants.hpp"
```

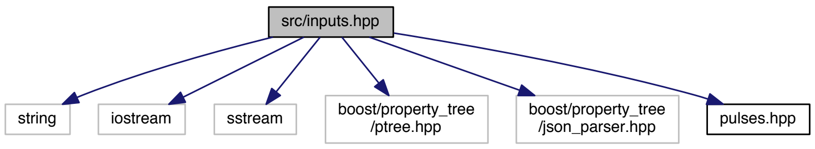
Include dependency graph for inputs.cpp:



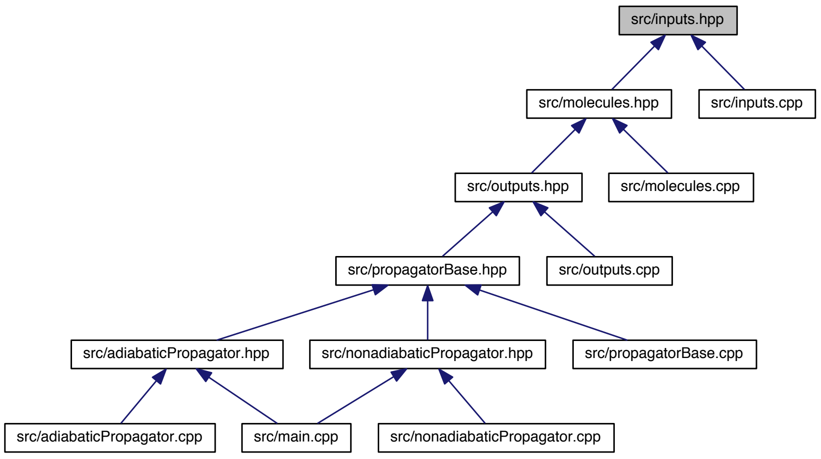
7.6 src/inputs.hpp File Reference

```
#include <string>
#include <iostream>
#include <sstream>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>
#include "pulses.hpp"
```

Include dependency graph for inputs.hpp:


src/inputs.hpp
string
iostream
sstream
boost/property_tree /ptree.hpp
boost/property_tree /json_parser.hpp
pulses.hpp

This graph shows which files directly or indirectly include this file:



Classes

- class inputParameters

Enumerations

- enum JOBTYPE { JOBTYPE::ADIABATIC, JOBTYPE::NONADIABATIC }

  Jobtypes.
- enum MOLSYM { MOLSYM::LINEAR, MOLSYM::SYMMETRIC_OBLATE, MOLSYM::SYMMETRIC_PRO←
LATE, MOLSYM::ASYMMETRIC }

  Symmetry Class.

Functions

- template< typename T >
std::vector< T > as_vector (boost::property_tree::ptree const &pt, boost::property_tree::ptree::key_type const &key)

  boost json to vector

7.6.1 Detailed Description

Author

J. Szekely

7.6.2 Enumeration Type Documentation

7.6.2.1 enum JOBTYPE [strong]

Jobtypes.

Specifies adiabatic or nonadiabatic calculation

Enumerator

ADIABATIC

NONADIABATIC

7.6.2.2 enum MOLSYM `[strong]`

Symmetry Class.

Determines the symmetry influencing the basis and elements of the Hamiltonian

Enumerator

LINEAR

SYMMETRIC_OBLATE

SYMMETRIC_PROLATE

ASYMMETRIC

7.6.3 Function Documentation

7.6.3.1 template<typename T > std::vector<T> as_vector (boost::property_tree::ptree const & *pt*, boost::property_tree::ptree::key_type const & *key*)

boost json to vector

helper function to convert boost json object into std::vector

Parameters

pt	property tree
key	property tree key

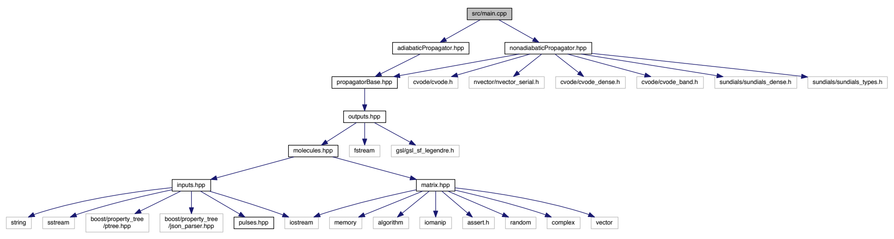
Returns

vector

7.7 src/main.cpp File Reference

```
#include "nonadiabaticPropagator.hpp"
#include "adiabaticPropagator.hpp"
```

Include dependency graph for main.cpp:



Functions

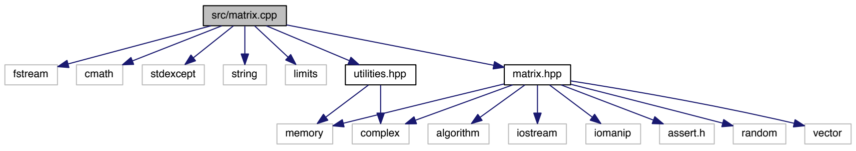
- int main (int argc, char const ∗argv[ ])

7.7.1 Function Documentation

7.7.1.1 int main (int *argc*, char const ∗ *argv[ ]*)

7.8 src/matrix.cpp File Reference

```
#include <fstream>
#include <cmath>
#include <stdexcept>
#include <string>
#include <limits>
#include "matrix.hpp"
#include "utilities.hpp"
```
Include dependency graph for matrix.cpp:



Functions

- void printMatrix (matrixComp &o, string filename, double ∗x, double ∗y)
- void printMatrix (matrixReal &o, string filename, double ∗x, double ∗y)

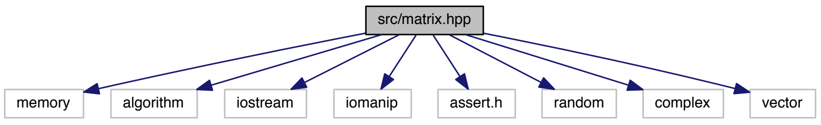
7.8.1 Function Documentation

7.8.1.1 void printMatrix (matrixComp & *o*, string *filename*, double ∗ *x*, double ∗ *y*)

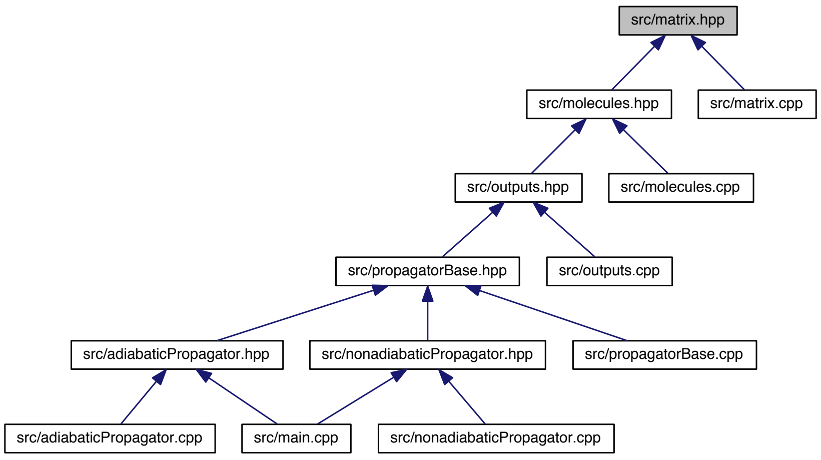
7.8.1.2 void printMatrix (matrixReal & *o*, string *filename*, double ∗ *x*, double ∗ *y*)

7.9 src/matrix.hpp File Reference

```
#include <memory>
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <assert.h>
#include <random>
#include <complex>
#include <vector>
```
Include dependency graph for matrix.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class matrixBase< T >
- class matrixReal

  Matrix of real numbers.
- class matrixComp

  Matrix of complex numbers.

Typedefs

- typedef std::complex< double > cplx

Functions

- template<typename T >
  std::ostream & operator<< (std::ostream &out, const matrixBase< T > &o)

  Overload the << operator to print a matrix.
- void printMatrix (matrixComp &o, std::string filename, double ∗x=nullptr, double ∗y=nullptr)

  Print entire matrix to file with optional x and y coordinates included.
- void printMatrix (matrixReal &o, std::string filename, double ∗x=nullptr, double ∗y=nullptr)

  Print entire matrix to file with optional x and y coordinates included.

7.9.1 Detailed Description

Author

J. Szekely

7.9.2 Typedef Documentation

7.9.2.1 typedef std::complex<double> **cplx**

7.9.3 Function Documentation

7.9.3.1 template<typename T > std::ostream& operator<< (std::ostream & *out,* const **matrixBase**< T > & *o*)

Overload the << operator to print a matrix.

7.9.3.2 void printMatrix (**matrixComp** & *o,* std::string *filename,* double ∗ *x* = `nullptr`, double ∗ *y* = `nullptr`)

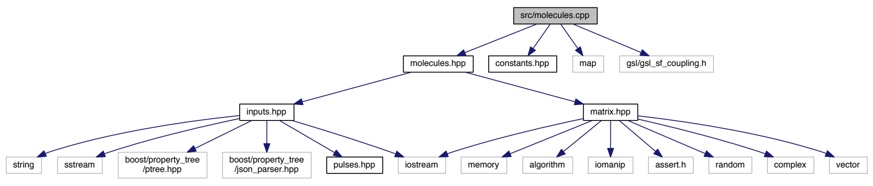
Print entire matrix to file with optional x and y coordinates included.

7.9.3.3 void printMatrix (**matrixReal** & *o,* std::string *filename,* double ∗ *x* = `nullptr`, double ∗ *y* = `nullptr`)

Print entire matrix to file with optional x and y coordinates included.

7.10 src/molecules.cpp File Reference

```
#include "molecules.hpp"
#include "constants.hpp"
#include <map>
#include <gsl/gsl_sf_coupling.h>
```
Include dependency graph for molecules.cpp:



Functions

- double FMIME (int J, int K, int M, int Q, int S, int j, int k, int m)

  Field Matter Interaction Matrix Element.

7.10.1 Function Documentation

7.10.1.1 double FMIME (int *J,* int *K,* int *M,* int *Q,* int *S,* int *j,* int *k,* int *m*)

Field Matter Interaction Matrix Element.

Calculates the coupling between two |JKM> states in an off resonance field

Parameters

J	J of State 1
K	K of State 1
M	M of State 1
Q	Interaction Quantum Number
S	Other Interaction Quantum Number
j	J of State 2
k	K of State 2
m	M of State 2

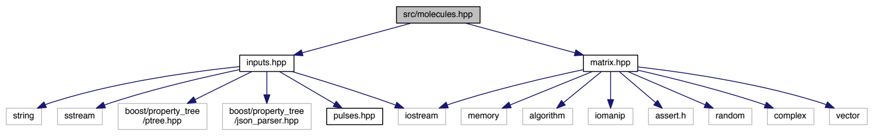
Returns

Coupling Matrix Element

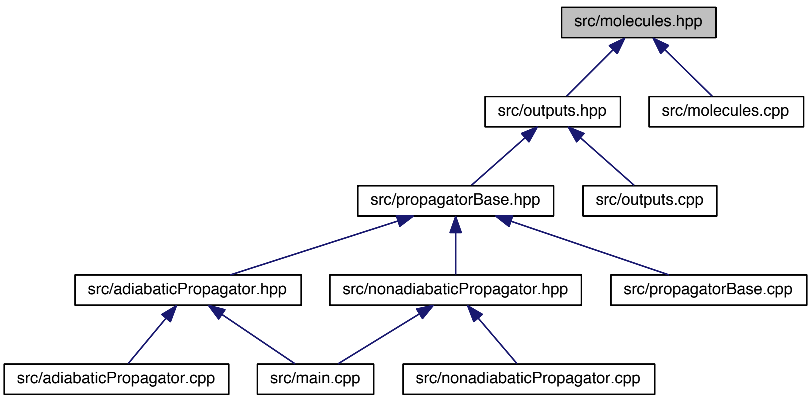
7.11 src/molecules.hpp File Reference

```
#include "inputs.hpp"
#include "matrix.hpp"
```

Include dependency graph for molecules.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct polarizability

  Container class for diagonal polarizability components.
- struct rotationalConstants

  Container for rotational constants.
- struct basis

Storage for the basis function |JKM> quantum numbers.

- class moleculeBase

    Molecule base class.
- class linearMolecule

    Linear Molecules.
- class symmetricTopMolecule

    Symmetric Top Molecules.
- class asymmetricTopMolecule

    Asymmetric Top Molecules.

Typedefs

- typedef std::vector< basis > basisSubset

    vector of basis set objects
- typedef std::vector< std::shared_ptr< basisSubset > > basisSubsets

    vector of pointers to basis subsets
- typedef std::vector< std::shared_ptr< matrixComp > > matrices

    vector of pointers to operator matrices
- typedef std::vector< std::shared_ptr< std::vector< double > > > arrays

    vector of pointers to data arrays

Functions

- double FMIME (int J, int K, int M, int Q, int S, int j, int k, int m)

    Field Matter Interaction Matrix Element.

7.11.1 Detailed Description

Author

J. Szekely

7.11.2 Typedef Documentation

7.11.2.1 typedef std::vector<std::shared_ptr<std::vector<double> > > arrays

vector of pointers to data arrays

7.11.2.2 typedef std::vector<basis> basisSubset

vector of basis set objects

7.11.2.3 typedef std::vector<std::shared_ptr<basisSubset> > basisSubsets

vector of pointers to basis subsets

7.11.2.4 typedef std::vector<std::shared_ptr<matrixComp> > matrices

vector of pointers to operator matrices

7.11.3 Function Documentation

7.11.3.1 double FMIME (int *J*, int *K*, int *M*, int *Q*, int *S*, int *j*, int *k*, int *m*)

Field Matter Interaction Matrix Element.

Calculates the coupling between two |JKM> states in an off resonance field

Parameters

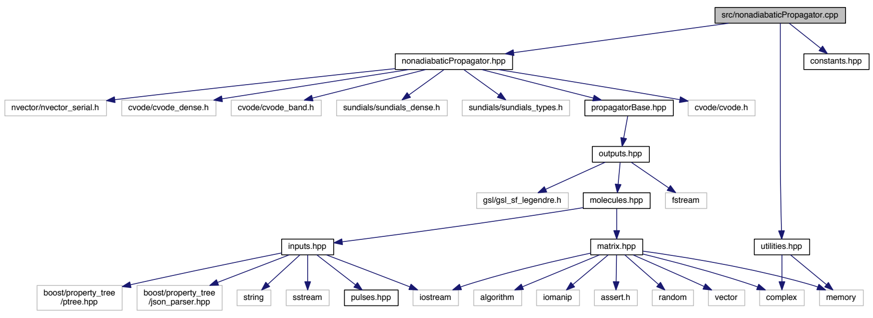
J	J of State 1
K	K of State 1
M	M of State 1
Q	Interaction Quantum Number
S	Other Interaction Quantum Number
j	J of State 2
k	K of State 2
m	M of State 2

Returns

Coupling Matrix Element

7.12 src/nonadiabaticPropagator.cpp File Reference

```
#include "nonadiabaticPropagator.hpp"
#include "utilities.hpp"
#include "constants.hpp"
```
Include dependency graph for nonadiabaticPropagator.cpp:



Macros

- #define Ith(v, i) NV_Ith_S(v,i-1) /∗ Ith numbers components 1..NEQ ∗/

Functions

- int check_flag (void ∗flagvalue, char ∗funcname, int opt)

  Function for checking the proper return of CVode functions.

7.12.1 Macro Definition Documentation

7.12.1.1 #define Ith(*v,* *i*) NV_Ith_S(v,i-1) /∗ Ith numbers components 1..NEQ ∗/

7.12.2 Function Documentation

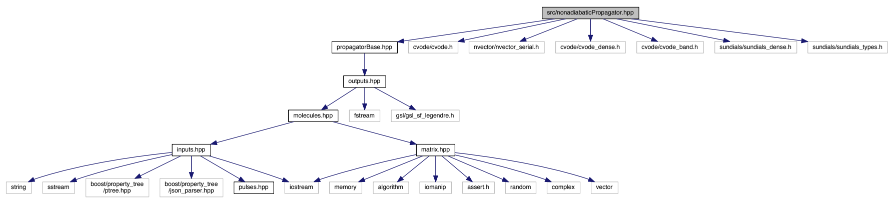
7.12.2.1 int check_flag (void ∗ *flagvalue,* char ∗ *funcname,* int *opt*)

Function for checking the proper return of CVode functions.

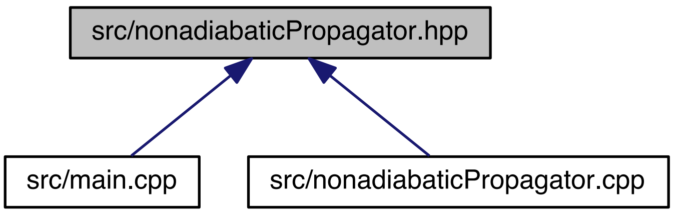
7.13 src/nonadiabaticPropagator.hpp File Reference

```
#include "propagatorBase.hpp"
#include <cvode/cvode.h>
#include <nvector/nvector_serial.h>
#include <cvode/cvode_dense.h>
#include <cvode/cvode_band.h>
#include <sundials/sundials_dense.h>
#include <sundials/sundials_types.h>
```

Include dependency graph for nonadiabaticPropagator.hpp:



This graph shows which files directly or indirectly include this file:


src/nonadiabaticPropagator.hpp
src/main.cpp
src/nonadiabaticPropagator.cpp

Classes

- class nonadiabaticPropagator

  Nonadiabatic Calculation Propagator.

Functions

- int check_flag (void ∗flagvalue, char ∗funcname, int opt)

Function for checking the proper return of CVode functions.

- int row_index (int i, int N)

  row helper function for one-to-one correspondence between matrix coordinate and upper triangular storage scheme

- int column_index (int i, int N)

  column helper function for one-to-one correspondence between matrix coordinate and upper triangular storage scheme

7.13.1 Function Documentation

7.13.1.1 int check_flag (void ∗ *flagvalue,* char ∗ *funcname,* int *opt*)

Function for checking the proper return of CVode functions.

7.13.1.2 int column_index (int *i,* int *N*) `[inline]`

column helper function for one-to-one correspondence between matrix coordinate and upper triangular storage scheme

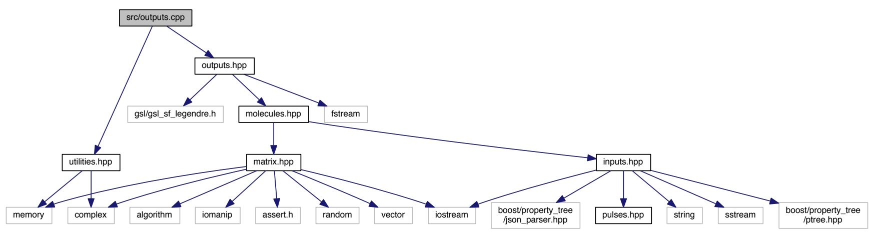
7.13.1.3 int row_index (int *i,* int *N*) `[inline]`

row helper function for one-to-one correspondence between matrix coordinate and upper triangular storage scheme

7.14 src/outputs.cpp File Reference

```
#include "outputs.hpp"
#include "utilities.hpp"
```

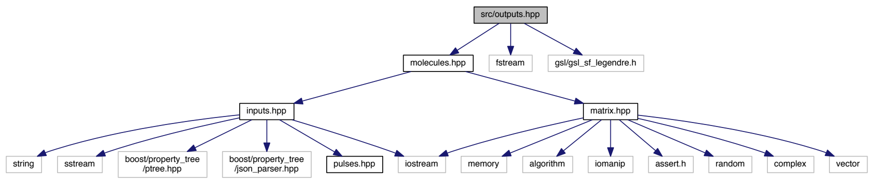
Include dependency graph for outputs.cpp:



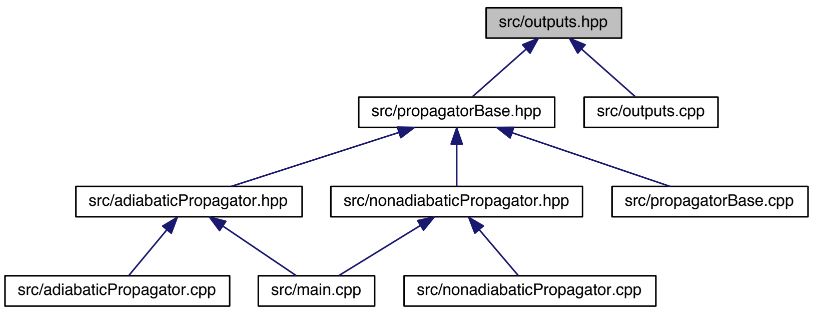
7.15 src/outputs.hpp File Reference

```
#include "molecules.hpp"
#include <fstream>
#include <gsl/gsl_sf_legendre.h>
```

Include dependency graph for outputs.hpp:



This graph shows which files directly or indirectly include this file:


src/outputs.hpp
src/propagatorBase.hpp
src/outputs.cpp
src/adiabaticPropagator.hpp
src/nonadiabaticPropagator.hpp
src/propagatorBase.cpp
src/adiabaticPropagator.cpp
src/main.cpp
src/nonadiabaticPropagator.cpp

Classes

- class observable

  Base class for observables that are obtained from a density matrix as Tr(O∗rho). Other outputs are obtained from other members of the propagator class, such as the wavefunction density or the list of basis states used in the calculation.
- class obsCosTheta3D

  $\langle\cos^2\theta_{3D}\rangle = \langle\cos^2(\hat{z}\cdot\hat{Z})\rangle$
- class obsCosTheta2D

  $\langle\cos^2\theta_{2D}\rangle$
- class obsEnergy

  $\langle\hat{H}\rangle$
- class obsCosThetaAlt

  $\langle\cos^2\theta'_{3D}\rangle = \langle\cos^2(\hat{x}\cdot\hat{Z})\rangle$
- class obsJ

  $\langle J^2\rangle$
- class obsK

  $\langle K^2\rangle$
- class obsM

  $\langle M\rangle$

Functions

- double cos2D (int J, int M, int j, int m)

  cos^2 theta (2D) matrix elements

7.15.1 Detailed Description

Author

J. Szekely

7.15.2 Function Documentation

7.15.2.1 double cos2D (int *J*, int *M*, int *j*, int *m*) `[inline]`

cos^2 theta (2D) matrix elements

Computes the cos^2 2D matrix elements between two spherical harmonic functions

Parameters

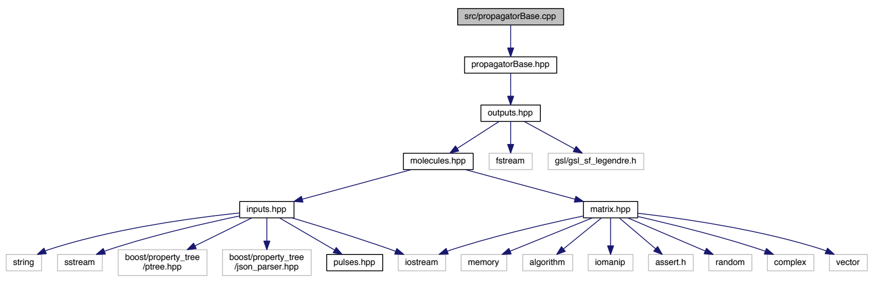
J	angular momentum of state 1
M	angular momentum projection of state 1
j	angular momentum of state 2
m	angular momentum projection of state 2

Returns

overlap

7.16 src/propagatorBase.cpp File Reference

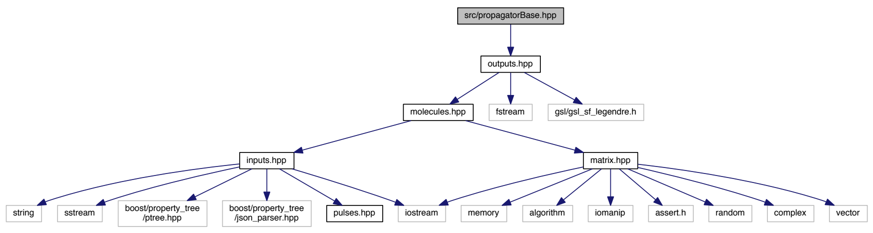
`#include "propagatorBase.hpp"`
Include dependency graph for propagatorBase.cpp:



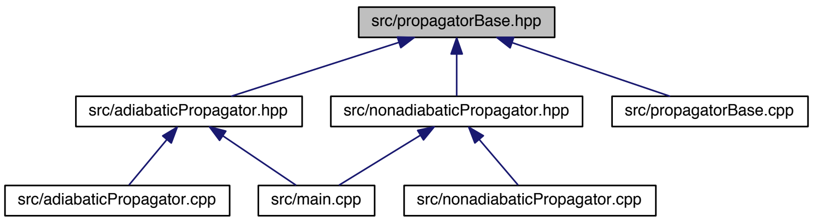
7.17 src/propagatorBase.hpp File Reference

`#include "outputs.hpp"`

Include dependency graph for propagatorBase.hpp:



This graph shows which files directly or indirectly include this file:


src/propagatorBase.hpp
src/adiabaticPropagator.hpp
src/nonadiabaticPropagator.hpp
src/propagatorBase.cpp
src/adiabaticPropagator.cpp
src/main.cpp
src/nonadiabaticPropagator.cpp

Classes

- class propagatorBase

  Base class for adiabatic and nonadiabatic calculations.

7.17.1 Detailed Description

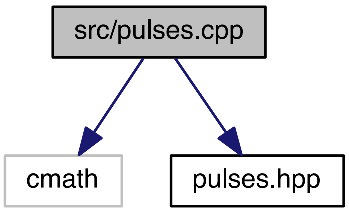
Author

J. Szekely

7.18 src/pulses.cpp File Reference

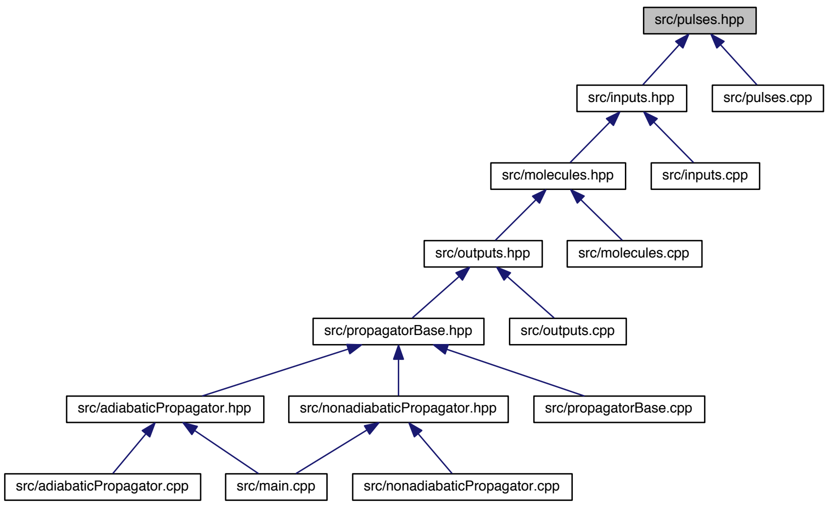
```
#include <cmath>
#include "pulses.hpp"
```

Include dependency graph for pulses.cpp:



7.19 src/pulses.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class pulse

7.19.1 Detailed Description

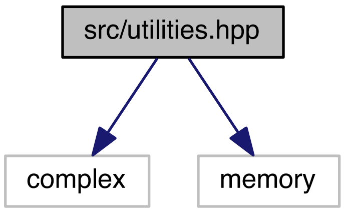
Author

J. Szekely

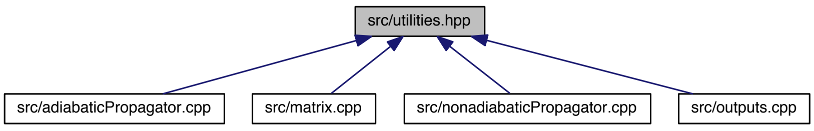
7.20 src/utilities.hpp File Reference

```
#include <complex>
#include <memory>
```

Include dependency graph for utilities.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void dzgemm_ (const char ∗transa, const char ∗transb, const int ∗m, const int ∗n, const int ∗k, const std↵ ::complex< double > ∗alpha, const double ∗a, const int ∗lda, const std::complex< double > ∗b, const int ∗ldb, const std::complex< double > ∗beta, std::complex< double > ∗c, const int ∗ldc)
- void dgemm_ (const char ∗transa, const char ∗transb, const int ∗m, const int ∗n, const int ∗k, const double ∗alpha, const double ∗a, const int ∗lda, const double ∗b, const int ∗ldb, const double ∗beta, double ∗c, const int ∗ldc)
- void dsyev_ (const char ∗, const char ∗, const int ∗, double ∗, const int ∗, double ∗, double ∗, const int ∗, int ∗)
- void dsyevd_ (const char ∗jobz, const char ∗uplo, const int ∗n, double ∗a, const int ∗lda, double ∗w, double ∗work, int ∗lwork, int ∗iwork, int ∗liwork, int ∗info)
- void zheev_ (const char ∗, const char ∗, const int ∗, std::complex< double > ∗, const int ∗, double ∗, std↵ ::complex< double > ∗, const int ∗, double ∗, int ∗)
- double ddot_ (const int ∗, const double ∗, const int ∗, const double ∗, const int ∗)
- void zdotc_ (std::complex< double > ∗, const int ∗, const std::complex< double > ∗, const int ∗, const std::complex< double > ∗, const int ∗)

- void daxpy_ (const int ∗, const double ∗, const double ∗, const int ∗, double ∗, const int ∗)
- void zaxpy_ (const int ∗, const std::complex< double > ∗, const std::complex< double > ∗, const int ∗, const std::complex< double > ∗, const int ∗)
- void zgemm3m_ (const char ∗transa, const char ∗transb, const int ∗m, const int ∗n, const int ∗k, const std↩ ::complex< double > ∗alpha, const std::complex< double > ∗a, const int ∗lda, const std::complex< double > ∗b, const int ∗ldb, const std::complex< double > ∗beta, std::complex< double > ∗c, const int ∗ldc)
- void zhemm_ (const char ∗side, const char ∗uplo, const int ∗m, const int ∗n, const std::complex< double > ∗alpha, const std::complex< double > ∗a, const int ∗lda, const std::complex< double > ∗b, const int ∗ldb, const std::complex< double > ∗beta, std::complex< double > ∗c, const int ∗ldc)
- void zgemv_ (const char ∗, const int ∗, const int ∗, const std::complex< double > ∗, const std::complex< double > ∗, const int ∗, const std::complex< double > ∗, const int ∗, const std::complex< double > ∗, std::complex< double > ∗, const int ∗)
- int izamax_ (const int ∗, const std::complex< double > ∗, const int ∗)
- int izamin_ (const int ∗, const std::complex< double > ∗, const int ∗)
- int idamax_ (const int ∗, const double ∗, const int ∗)
- int idamin_ (const int ∗, const double ∗, const int ∗)
- void mkl_ddnscsr_ (const int ∗, const int ∗, const int ∗, const double ∗, const int ∗, const double ∗, const int ∗, const int ∗, int ∗)
- void mkl_domatcopy_ (const char ∗, const char ∗, const int ∗, const int ∗, const double ∗, const double ∗, const int ∗, double ∗, const int ∗)
- void mkl_zomatcopy_ (const char ∗, const char ∗, const int ∗, const int ∗, const std::complex< double > ∗, const std::complex< double > ∗, const int ∗, std::complex< double > ∗, const int ∗)
- void mkl_dcsrgemv_ (const char ∗, const int ∗, const double ∗, const int ∗, const int ∗, const double ∗, const double ∗)
- void dgetrf_ (const int ∗, const int ∗, double ∗, int ∗, int ∗, int ∗)
- void dgetri_ (const int ∗, double ∗, int ∗, int ∗, double ∗, int ∗, int ∗)
- void dgesv_ (const int ∗n, const int ∗nrhs, double ∗a, const int ∗lda, int ∗ipiv, double ∗b, const int ∗ldb, int ∗info)
- void dswap_ (const int ∗n, double ∗x, const int ∗incx, double ∗y, const int ∗incy)
- void zswap_ (const int ∗n, std::complex< double > ∗x, const int ∗incx, std::complex< double > ∗y, const int ∗incy)
- void zgetrf_ (const int ∗, const int ∗, std::complex< double > ∗, int ∗, int ∗, int ∗)
- void zgetri_ (const int ∗, std::complex< double > ∗, int ∗, int ∗, std::complex< double > ∗, int ∗, int ∗)
- void dgesvd_ (const char ∗, const char ∗, const int ∗, const int ∗, double ∗, const int ∗, double ∗, double ∗, const int ∗, double ∗, const int ∗, double ∗, const int ∗, int ∗)
- void dsyevr_ (const char ∗, const char ∗, const char ∗, const int ∗, double ∗, const int ∗, const double ∗, const double ∗, const int ∗, const int ∗, const double ∗, int ∗, double ∗, double ∗, const int ∗, int ∗, double ∗, int ∗, int ∗, int ∗, int ∗)

7.20.1 Detailed Description

Author

J. Szekely

7.20.2 Function Documentation

7.20.2.1 void daxpy_ (const int ∗ , const double ∗ , const double ∗ , const int ∗ , double ∗ , const int ∗)

7.20.2.2 double ddot_ (const int ∗ , const double ∗ , const int ∗ , const double ∗ , const int ∗)

7.20.2.3 void dgemm_ (const char ∗ *transa,* const char ∗ *transb,* const int ∗ *m,* const int ∗ *n,* const int ∗ *k,* const double ∗ *alpha,* const double ∗ *a,* const int ∗ *lda,* const double ∗ *b,* const int ∗ *ldb,* const double ∗ *beta,* double ∗ *c,* const int ∗ *ldc*)

7.20.2.4 void dgesv_ (const int ∗ *n,* const int ∗ *nrhs,* double ∗ *a,* const int ∗ *lda,* int ∗ *ipiv,* double ∗ *b,* const int ∗ *ldb,* int ∗ *info*)

7.20.2.5 void dgesvd_ (const char ∗ , const char ∗ , const int ∗ , const int ∗ , double ∗ , const int ∗ , double ∗ , double ∗ , const int ∗ , double ∗ , const int ∗ , double ∗ , const int ∗ , int ∗)

7.20.2.6 void dgetrf_ (const int ∗ , const int ∗ , double ∗ , int ∗ , int ∗ , int ∗)

7.20.2.7 void dgetri_ (const int ∗ , double ∗ , int ∗ , int ∗ , double ∗ , int ∗ , int ∗)

7.20.2.8 void dswap_ (const int ∗ *n,* double ∗ *x,* const int ∗ *incx,* double ∗ *y,* const int ∗ *incy*)

7.20.2.9 void dsyev_ (const char ∗ , const char ∗ , const int ∗ , double ∗ , const int ∗ , double ∗ , double ∗ , const int ∗ , int ∗)

7.20.2.10 void dsyevd_ (const char ∗ *jobz,* const char ∗ *uplo,* const int ∗ *n,* double ∗ *a,* const int ∗ *lda,* double ∗ *w,* double ∗ *work,* int ∗ *lwork,* int ∗ *iwork,* int ∗ *liwork,* int ∗ *info*)

7.20.2.11 void dsyevr_ (const char ∗ , const char ∗ , const char ∗ , const int ∗ , double ∗ , const int ∗ , const double ∗ , const double ∗ , const int ∗ , const int ∗ , const double ∗ , int ∗ , double ∗ , double ∗ , const int ∗ , int ∗ , double ∗ , int ∗ , int ∗ , int ∗ , int ∗)

7.20.2.12 void dzgemm_ (const char ∗ *transa,* const char ∗ *transb,* const int ∗ *m,* const int ∗ *n,* const int ∗ *k,* const std::complex< double > ∗ *alpha,* const double ∗ *a,* const int ∗ *lda,* const std::complex< double > ∗ *b,* const int ∗ *ldb,* const std::complex< double > ∗ *beta,* std::complex< double > ∗ *c,* const int ∗ *ldc*)

7.20.2.13 int idamax_ (const int ∗ , const double ∗ , const int ∗)

7.20.2.14 int idamin_ (const int ∗ , const double ∗ , const int ∗)

7.20.2.15 int izamax_ (const int ∗ , const std::complex< double > ∗ , const int ∗)

7.20.2.16 int izamin_ (const int ∗ , const std::complex< double > ∗ , const int ∗)

7.20.2.17 void mkl_dcsrgemv_ (const char ∗ , const int ∗ , const double ∗ , const int ∗ , const int ∗ , const double ∗ , const double ∗)

7.20.2.18 void mkl_ddnscsr_ (const int ∗ , const int ∗ , const int ∗ , const double ∗ , const int ∗ , const double ∗ , const int ∗ , const int ∗ , int ∗)

7.20.2.19 void mkl_domatcopy_ (const char ∗ , const char ∗ , const int ∗ , const int ∗ , const double ∗ , const double ∗ , const int ∗ , double ∗ , const int ∗)

7.20.2.20 void mkl_zomatcopy_ (const char ∗ , const char ∗ , const int ∗ , const int ∗ , const std::complex< double > ∗ , const std::complex< double > ∗ , const int ∗ , std::complex< double > ∗ , const int ∗)

7.20.2.21 void zaxpy_ (const int ∗ , const std::complex< double > ∗ , const std::complex< double > ∗ , const int ∗ , const std::complex< double > ∗ , const int ∗)

7.20.2.22 void zdotc_ (std::complex< double > ∗ , const int ∗ , const std::complex< double > ∗ , const int ∗ , const std::complex< double > ∗ , const int ∗)

7.20.2.23 void zgemm3m_ (const char ∗ *transa,* const char ∗ *transb,* const int ∗ *m,* const int ∗ *n,* const int ∗ *k,* const std::complex< double > ∗ *alpha,* const std::complex< double > ∗ *a,* const int ∗ *lda,* const std::complex< double > ∗ *b,* const int ∗ *ldb,* const std::complex< double > ∗ *beta,* std::complex< double > ∗ *c,* const int ∗ *ldc*)

7.20.2.24 void zgemv_ (const char $*$, const int $*$, const int $*$, const std::complex$<$ double $>$ $*$, const std::complex$<$ double $>$ $*$, const int $*$, const std::complex$<$ double $>$ $*$, const int $*$, const std::complex$<$ double $>$ $*$, std::complex$<$ double $>$ $*$, const int $*$)

7.20.2.25 void zgetrf_ (const int $*$, const int $*$, std::complex$<$ double $>$ $*$, int $*$, int $*$, int $*$)

7.20.2.26 void zgetri_ (const int $*$, std::complex$<$ double $>$ $*$, int $*$, int $*$, std::complex$<$ double $>$ $*$, int $*$, int $*$)

7.20.2.27 void zheev_ (const char $*$, const char $*$, const int $*$, std::complex$<$ double $>$ $*$, const int $*$, double $*$, std::complex$<$ double $>$ $*$, const int $*$, double $*$, int $*$)

7.20.2.28 void zhemm_ (const char $*$ *side,* const char $*$ *uplo,* const int $*$ *m,* const int $*$ *n,* const std::complex$<$ double $>$ $*$ *alpha,* const std::complex$<$ double $>$ $*$ *a,* const int $*$ *lda,* const std::complex$<$ double $>$ $*$ *b,* const int $*$ *ldb,* const std::complex$<$ double $>$ $*$ *beta,* std::complex$<$ double $>$ $*$ *c,* const int $*$ *ldc*)

7.20.2.29 void zswap_ (const int $*$ *n,* std::complex$<$ double $>$ $*$ *x,* const int $*$ *incx,* std::complex$<$ double $>$ $*$ *y,* const int $*$ *incy*)

Index