# Inhomogeneous Lattice Systems

Generated by Doxygen 1.8.6

Fri Apr 4 2014 16:35:02

# Contents

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 arguments Struct Reference

### Public Attributes

- unsigned **sites**
- double **latticeDepth**
- double **inhStrength**
- char $*$ **outPath**

### 4.1.1 Detailed Description

responsible for program parameters

**See Also**

'argp' from gnu

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

- inhLatticeSystem.cpp

## 4.2 blochFindMultipleRoots Struct Reference

### Public Member Functions

- void **searchDownUp** ()
- void **findMin** (mt start, mt end, mt precision)

### Public Attributes

- mt **intervalStart**
- mt **intervalEnd**
- mt **rootValueAccuracy**
- mt **takenZero**
- mt($*$ **rf** )(mt)
- mt($*$ **mf** )(mt, void $*$)
- set$<$ mt $>$ **allRoots**
- set$<$ mt $>$ **allMins**
- mt **intervalForMinSearch**

### 4.2.1 Detailed Description

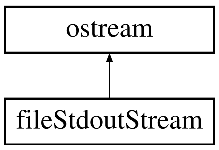
Searches for roots in an input function

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

- inhLatticeSystem.cpp

## 4.3 fileStdoutStream Class Reference

Inheritance diagram for fileStdoutStream:



### Public Types

- typedef std::ostream &($*$ **stream_function** )(std::ostream &)

### Public Member Functions

- void **open** (const char $*$filename)
- template$<$typename T $>$
  fileStdoutStream & **operator**$<<$ (const T &arg)
- fileStdoutStream & **operator**$<<$ (stream_function func)
- fileStdoutStream & **flush** ()

### Public Attributes

- ofstream **file**

### 4.3.1 Detailed Description

This class introduces the functionality for the object 'fout' which is similar to the standard c++ object 'cout' but different in that it streams all the output to a file in addition to 'cout'.

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

- inhLatticeSystem.cpp

## 4.4 FsOverTS Struct Reference

### Public Member Functions

- **FsOverTS** (unsigned n)

### Public Attributes

- Col< mt > **Heigenvalues**
- Col< mt > **HeigenvaluesDeriv**
- Col< mt > **PositionOverTrapStrength**
- Col< mt > **BpPositionOverTrapStrength**

### 4.4.1 Detailed Description

needed to return information from the calcFsOverTS procedure

**See Also**

calcFsOverTS

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

- inhLatticeSystem.cpp

## 4.5 inhLatticeSystem Class Reference

### Public Member Functions

- void outputPropagatedWavefunctionEndValueOverE ()
- void outputPotential0 ()
- void outputPotential ()
- void calculateH0eigenvalues ()
- void calculateH0eigenfunctions ()
- void calculateOverlapWithH0eigenfunctionsFromFile ()
- void calculateSchroedinger0Error ()
- void H0H0Overlap ()
- void hopping0OverDl ()
- void calculateCenteredWannier0Functions ()
- void calculateWannier0Functions ()
- void calculateWannier0Overlap ()
- void calculateGaussianInWannier0Basis ()
- void calculateOverlapWithWannierStatesFromFile ()
- void hopping0viaInt ()
- bool calculateHinW0 (unsigned n, unsigned m, unsigned &count)
- void calculateHinW0 ()
- void calculateHeigenvaluesESorted ()
- void calculateBand0ProjectorsExpectationInHeigenstates ()
- void calculateHeigenvaluesOverTrapStrength ()
- void calculateXinW0 ()
- void calculateHlocalization ()
- void assignBandOrderEqualHeigenvalueOrder ()
- void assignBandOrderUsingLocalization ()
- void calculateWannier0Eigenvalues ()
- void calculateFOverTrapStrength ()
- void calculateChangeInLocalizations ()
- void calculateLeftRightSideOfHinW0EigenvalueEquation ()
- void calculateProductOfHWannier0overlapMatrixWithItselvesAdjoint ()
- void calculateBandOrderedVariables ()

- void calculateHeigenstates ()
- void outputSelectedFunctions ()
- void compareWithHeigenstatesFromFile ()
- void calculateSchroedingerError ()
- void calculateOverlapWithHarmonicOscillator ()
- void calculateWinE ()
- void calculateWannierEigenvalues ()
- void calculateLeftRightSideOfWannierInHEigenvalueEquation ()
- void calculateProductOfWannierHoverlapMatrixWithItselvesAdjoint ()
- void calculateWannierFunctions ()
- void calculateWannierFunctionsRephased ()
- void calculateHopping ()
- void calculateU ()
- void calculateHeigenstatesFromWannier ()
- void calculateWannierFunctionValueUsingHeigenfunctions ()
- void calculateWannierOverlap ()
- void calculateBorderEffects ()
- void calc ()

## Public Attributes

- ofstream **file**
- ifstream **ifile**
- mt **sampleWidth**
- mt **odeSampleWidth**
- mt **wannier0FunctionCutoff**
- mt **HinW0cutoff**
- unsigned **kSteps**
- unsigned **nrSites**
- blochFindMultipleRoots **fR**
- unsigned **minBands**
- clock_t **tMark**
- Col< mt > **H0eigenvalues**
- Col< mt > **H0kvalues**
- vector< fct< complex< mt > > > **H0eigenfunctions**
- vector< fct< complex< mt > > > **centeredWannier0Functions**
- vector< fct< complex< mt > > > **wannier0Functions**
- vector< fct< complex< mt > > > **wannier0FunctionsPeriod**
- Col< mt > **wannier0Eigenvalues**
- Mat< cx_double > **HinW0partV**
- Mat< cx_double > **HinW0partH0**
- Mat< cx_double > **HinW0**
- Col< mt > **HeigenvaluesESorted**
- Mat< cx_double > **W0HOverlapESorted**
- Mat< complex< mt > > **XinW0**
- Col< mt > **Hlocalization**
- Col< mt > **bandToEindex**
- Col< mt > **Heigenvalues**
- Mat< cx_double > **W0HOverlap**
- vector< fct< complex< mt > > > **Heigenfunctions**
- Mat< cx_double > **wannierInH**
- Col< mt > **wannierEigenvalues**
- Mat< cx_double > **HWannierOverlap**
- Mat< cx_double > **HWannierOverlapRephased**
- vector< fct< complex< mt > > > **wannierFunctions**

### 4.5.1 Detailed Description

This is the most essential class of the program. It includes most of the functionality for solving inhomogeneous lattice systems.

Thus, it contains methods for

-finding the homogeneous system eigenvalues, eigenfunctions and Wannierfunctions

-diagonalizing the complete (inhomogeneous) Hamiltonian in the homogeneous system Wannier basis, giving the inhomogeneous system eigenvalues and eigenfunctions

-systematically assigning states to bands

-applying Kivelson's generalized definition of the Wannierfunctions to obtain inhomogeneous Wannier functions

-calculating inhomogeneous system Hubbard parameters

### 4.5.2 Member Function Documentation

#### 4.5.2.1 void inhLatticeSystem::assignBandOrderEqualHeigenvalueOrder ( ) `[inline]`

assume energy gap still present i.e. define band projectors in terms energy ordered eigenstates

#### 4.5.2.2 void inhLatticeSystem::assignBandOrderUsingLocalization ( ) `[inline]`

define band projectors using the localization of the energy eigenstates

#### 4.5.2.3 void inhLatticeSystem::calc ( ) `[inline]`

MAIN EXECUTION PROCEDURE: uncomment everything you want calculated

#### 4.5.2.4 void inhLatticeSystem::calculateBand0ProjectorsExpectationInHeigenstates ( ) `[inline]`

calculate the expectation value of multiple primary lattice band projection operators for every eigenstate

#### 4.5.2.5 void inhLatticeSystem::calculateBandOrderedVariables ( ) `[inline]`

resort quantities taking band assigning into account

#### 4.5.2.6 void inhLatticeSystem::calculateBorderEffects ( ) `[inline]`

calculate system edge influences shape of Wannier state

#### 4.5.2.7 void inhLatticeSystem::calculateCenteredWannier0Functions ( ) `[inline]`

calculate one primary lattice Wannier function per band

#### 4.5.2.8 void inhLatticeSystem::calculateChangeInLocalizations ( ) `[inline]`

calculate how the localization changes as function of the inhomogenity strength

#### 4.5.2.9 void inhLatticeSystem::calculateFOverTrapStrength ( ) [inline]

calculate eigenvalues, derivative of eigenvalues, expectation of position and expectation of band projected position operator for a range of inhomogenity strength

#### 4.5.2.10 void inhLatticeSystem::calculateGaussianInWannier0Basis ( ) [inline]

calculate overlap between Gaussian and Wannier basis

#### 4.5.2.11 void inhLatticeSystem::calculateH0eigenfunctions ( ) [inline]

calculate the eigenfunctions of the homogeneous lattice system

#### 4.5.2.12 void inhLatticeSystem::calculateH0eigenvalues ( ) [inline]

calculate the eigenvalues of the homogeneous lattice system

#### 4.5.2.13 void inhLatticeSystem::calculateHeigenstates ( ) [inline]

calculate the real space eigenfunctions for the whole system

#### 4.5.2.14 void inhLatticeSystem::calculateHeigenstatesFromWannier ( ) [inline]

recalculate the eigenstates from the Wannier states

#### 4.5.2.15 void inhLatticeSystem::calculateHeigenvaluesESorted ( ) [inline]

calculate system energy eigenvalues and overlap elements with primary lattice Wannier basis

#### 4.5.2.16 void inhLatticeSystem::calculateHeigenvaluesOverTrapStrength ( ) [inline]

calculate energy eigenvalues for a whole range of inhomogenity strength

#### 4.5.2.17 bool inhLatticeSystem::calculateHinW0 ( unsigned *n*, unsigned *m*, unsigned & *count* ) [inline]

calculate single element with index (n,m) of Hamiltonian in primary lattice Wannier basis using primary lattice hopping elements and real space integration with inhomgenity potential

#### 4.5.2.18 void inhLatticeSystem::calculateHinW0 ( ) [inline]

calculate Hamiltonian in primary lattice Wannier basis using primary lattice hopping elements and real space integration with inhomgenity potential

#### 4.5.2.19 void inhLatticeSystem::calculateHlocalization ( ) [inline]

calculate expectation of position operator for every energy eigenstate

#### 4.5.2.20 void inhLatticeSystem::calculateHopping ( ) [inline]

calculate the hopping using the generalized Wannier basis

#### 4.5.2.21 void inhLatticeSystem::calculateLeftRightSideOfHinW0EigenvalueEquation ( ) [inline]

calculate both sides of eigenvalue equation

#### 4.5.2.22 void inhLatticeSystem::calculateLeftRightSideOfWannierInHEigenvalueEquation ( ) [inline]

calculate both sides of eigenvalue equation

#### 4.5.2.23 void inhLatticeSystem::calculateOverlapWithH0eigenfunctionsFromFile ( ) [inline]

calculate the overlap of the homogeneous eigenfunctions with other functions supplied through a file

#### 4.5.2.24 void inhLatticeSystem::calculateOverlapWithHarmonicOscillator ( ) [inline]

calculate overlap between eigenstates and harmonic Oscillator states (verification procedure/relevant if primary lattice depth is zero)

#### 4.5.2.25 void inhLatticeSystem::calculateOverlapWithWannierStatesFromFile ( ) [inline]

calculate overlap between primary lattice Wannier states and states in file

#### 4.5.2.26 void inhLatticeSystem::calculateProductOfHWannier0overlapMatrixWithItselvesAdjoint ( ) [inline]

see if overlaps between eigenstates and primary lattice Wannier states are unitary

#### 4.5.2.27 void inhLatticeSystem::calculateProductOfWannierHoverlapMatrixWithItselvesAdjoint ( ) [inline]

verify unity of overlap matrix

#### 4.5.2.28 void inhLatticeSystem::calculateSchroedinger0Error ( ) [inline]

calculate the error in the eigenfunctions for every band, function and position

#### 4.5.2.29 void inhLatticeSystem::calculateSchroedingerError ( ) [inline]

calculate the band, state and position error of eigenfunctions using the Schroedinger equation

#### 4.5.2.30 void inhLatticeSystem::calculateU ( ) [inline]

calculate the onsite interaction using the generalized Wannier basis

#### 4.5.2.31 void inhLatticeSystem::calculateWannier0Eigenvalues ( ) [inline]

calculate eigenvalues of primary lattice band projected position operator assuming non-periodic states

#### 4.5.2.32 void inhLatticeSystem::calculateWannier0Functions ( ) [inline]

calculate all primary lattice Wannier functions using translation symmetry

#### 4.5.2.33 void inhLatticeSystem::calculateWannier0Overlap ( ) [inline]

calculate overlap between all primary lattice Wannier functions

#### 4.5.2.34 void inhLatticeSystem::calculateWannierEigenvalues ( ) [inline]

calculate eigenvalues and overlaps for generalized Wannier states with energy eigenstates

#### 4.5.2.35 void inhLatticeSystem::calculateWannierFunctions ( ) [inline]

calculate generalized Wannier states in real space

#### 4.5.2.36 void inhLatticeSystem::calculateWannierFunctionsRephased ( ) [inline]

rephase the generalized Wannier states

#### 4.5.2.37 void inhLatticeSystem::calculateWannierFunctionValueUsingHeigenfunctions ( ) [inline]

calculate value of Wannier function using energy eigenfunctions

#### 4.5.2.38 void inhLatticeSystem::calculateWannierOverlap ( ) [inline]

calculate overlaps between Wannier states

#### 4.5.2.39 void inhLatticeSystem::calculateWinE ( ) [inline]

calculate band projected position operator in the basis of energy eigenstates

#### 4.5.2.40 void inhLatticeSystem::calculateXinW0 ( ) [inline]

calculate band projected position operator in the primary lattice Wannier basis

#### 4.5.2.41 void inhLatticeSystem::compareWithHeigenstatesFromFile ( ) [inline]

compare real space eigenfunctions with functions from a file

#### 4.5.2.42 void inhLatticeSystem::H0H0Overlap ( ) [inline]

calculate overlaps among homogeneous eigenfunctions

#### 4.5.2.43 void inhLatticeSystem::hopping0OverDl ( ) [inline]

calculate primary lattice hopping using Fourier transform of dispersion relation

#### 4.5.2.44 void inhLatticeSystem::hopping0viaInt ( ) [inline]

calculate primary lattice hopping via integration over Wannierfunctions

#### 4.5.2.45 void inhLatticeSystem::outputPotential ( ) [inline]

Output the inhomogeneous lattice potential

#### 4.5.2.46 void inhLatticeSystem::outputPotential0 ( ) [inline]

Output the homogeneous lattice potential

#### 4.5.2.47 void inhLatticeSystem::outputPropagatedWavefunctionEndValueOverE ( ) [inline]

Output the end value of the propagated u function for a range of E values

#### 4.5.2.48 void inhLatticeSystem::outputSelectedFunctions ( ) [inline]

output a desired set of quantities

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

- inhLatticeSystem.cpp

## 4.6 linAssignInfoType Struct Reference

### Public Attributes

- unsigned **n**
- Mat< cx_double > **HinW0partH0**
- Mat< cx_double > **HinW0partV**
- mt **errThreshold**
- vector< fct< complex< mt > > > ∗ **HeigenvaluesOverTrapStrength**
- vector< fct< complex< mt > > > ∗ **HeigenvaluesOverTrapStrengthDeriv**
- Col< mt > **wannier0Eigenvalues**
- Mat< cx_double > **XinW0**
- vector< fct< complex< mt > > > ∗ **PositionOverTrapStrength**
- vector< fct< complex< mt > > > ∗ **BpPositionOverTrapStrength**

### 4.6.1 Detailed Description

needed to pass information to the calcFsOverTS procedure

**See Also**

calcFsOverTS

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

- inhLatticeSystem.cpp

## 4.7 physical_parameters Class Reference

### Public Attributes

- mt **hbar**

- mt **m**
- mt **a**
- mt **lambda**
- mt **k_L**
- mt **E_r**
- mt **s_bound**
- mt **A**
- mt **ld**
- mt **E**
- mt **k**
- mt **inhShift**
- mt **a_0**
- mt **a_s**
- mt **g**
- mt **inhS**
- mt **borderTrapHeight**
- mt **inhLatRatio**
- mt **alpha**
- mt **delta**

### 4.7.1 Detailed Description

All the system information is contained in the variables in this class. One global object called 'pp' is created from this class and used throughout the calculations.

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

- inhLatticeSystem.cpp

# Chapter 5

# File Documentation

## 5.1 inhLatticeSystem.cpp File Reference

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cfloat>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <set>
#include <map>
#include <sys/stat.h>
#include <argp.h>
#include <armadillo>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_min.h>
#include <nr/nr.h>
#include <nr/nrtypes.h>
#include <nr/nrutil.h>
#include <boost/math/special_functions/hermite.hpp>
#include <utils.cpp>
#include <ODE_integration.cpp>
#include <diag_Householder_QL/diag_hermitian_Householder_QL.cpp>
#include <fct_classes/fct_class.cpp>
#include <fct_classes/fct_class_functions.cpp>
#include <jHelper.cpp>
#include <findMultipleBunchedRoots.cpp>
#include <iterators.cpp>
#include <RK4solver.hpp>
#include <DynStep.hpp>
#include <DGLsolver.hpp>
#include "streamer.cpp"
```

### Classes

- class fileStdoutStream
- class physical_parameters
- struct blochFindMultipleRoots

- struct linAssignInfoType
- struct FsOverTS
- class inhLatticeSystem
- struct arguments

## Macros

- #define **outputWithHeader**(x, header)
- #define **outputWO**(x) outputWithHeader(x,"");

## Functions

- mt Vlat (mt x, physical_parameters &pp)
- complex< mt > Vinh (mt x)
- mt **Vlat** (mt x)
- complex< mt > V (mt x, physical_parameters &pp)
- complex< mt > **testF** (mt x)
- void ODE_derivs (const double x, double ∗y, double ∗dydx, physical_parameters &pp)
- void dglSolve (ODE< physical_parameters > ∗&D, unsigned eqncount, mt const xinit, mt ∗const &yinit)
- mt propagateFunction (mt par)
- mt invPropagateFunction (mt par)
- mt propagateFunction (mt par, void ∗)
- mt invPropagateFunction (mt par, void ∗)
- void outputPropagatedWavefunctionEndValueOverE (mt const Estart, mt const Eend, mt const step)
- unsigned **bandToIndex0** (unsigned band, unsigned stateIndexInBand)
- unsigned **bandToIndex** (unsigned band, unsigned stateIndexInBand)
- void **normalizeFct** (fct< complex< mt > > &par)
- Mat< cx_double > calculateOverlapMatrix (vector< fct< complex< double > > > const &Es)
- Mat< cx_double > calculateEigenstatesBlockwise (Mat< cx_double > const symmetricMatrix, Col< mt > &eigenvals, unsigned statesPerBand)
- Mat< mt > createDiagonalMatrixFromVector (Col< mt > const eigenvalues)
- void **addPhaseToLocalizeOnRealAxis** (fct< complex< mt > > &f)
- template<typename fctRangeType , typename Fret >
  fctRangeType trapez_int_refined_f (fct< fctRangeType > func1, Fret(∗f)(mt), fctDomainType x_lo, fctDomainType x_hi, int n, fctRangeType &s)

  *the same integration procedure for a product of two functions*
- template<typename fctRangeType , typename Fret >
  fctRangeType trapez_int_refined_f (fct< fctRangeType > func1, fct< fctRangeType > func2, Fret(∗f)(mt), fctDomainType x_lo, fctDomainType x_hi, int n, fctRangeType &s)

  *the same integration procedure for a product of two functions*
- template<typename fctRangeType , typename Fret >
  fctRangeType romberg_integral_f (fct< fctRangeType > func1, Fret(∗f)(mt), double romberg_precision=romberg_integration_precision)

  *Romberg integration routine taken from numerical recipes in C++, the file qromb.c.*
- template<typename fctRangeType , typename Fret >
  fctRangeType romberg_integral_f (fct< fctRangeType > func1, fct< fctRangeType > func2, Fret(∗f)(mt), double romberg_precision=romberg_integration_precision)

  *Romberg integration routine taken from numerical recipes in C++, the file qromb.c.*
- mt constantOp (mt par)
- mt unityOp (mt par)
- mt absUnityOp (mt par)
- mt squareOp (mt par)
- FsOverTS calcFsOverTS (mt const inhStrength)

- void saveFsOverTS (mt const inhStrength, FsOverTS const &F)
- unsigned findClosest (mt const &value, Col< mt > const &vec, mt &error)
- void linAssign (mt const start, mt const end, FsOverTS const &startF, FsOverTS &endF)
- template<typename fctRangeType >
  string getFileExt (vector< fct< fctRangeType > > var)

  *determine file extension from variable type*
- template<typename fctRangeType >
  string getFileExt (fct< fctRangeType > var)

  *determine file extension from variable type*
- template<typename matType >
  string getFileExt (Mat< matType > var)

  *determine file extension from variable type*
- template<typename type >
  string getFileExt (type var)

  *determine file extension from variable type*
- template<typename fctRangeType >
  vector< fct< fctRangeType > > outputModifier (vector< fct< fctRangeType > > var)

  *allows to modify file output independence on variable type*
- template<typename fctRangeType >
  fct< fctRangeType > outputModifier (fct< fctRangeType > var)

  *allows to modify file output independence on variable type*
- template<typename matType >
  Mat< matType > outputModifier (Mat< matType > var)

  *allows to modify file output independence on variable type*
- Mat< mt > **outputModifier** (Mat< complex< mt > > var)
- template<typename type >
  type outputModifier (type var)

  *allows to modify file output independence on variable type*
- int main (int argc, char **argv)

## Variables

- string **dirName**
- fileStdoutStream fout
- class physical_parameters **pp**
- unsigned **kmax**
- ODE< physical_parameters > * **D1** =NULL
- ODE< physical_parameters > * **D2** =NULL
- mt **odeStepWidth**
- mt **propStart**
- mt **propEnd**
- fct< complex< mt > > **phiEndPoints**
- map< mt, mt > **calculatedValues**
- mt **lastk** =DBL_MAX
- unsigned statesPerBand0
- unsigned statesPerBand
- struct linAssignInfoType **linAssignInfo**
- const char * **argp_program_version**
- const char * **argp_program_bug_address**

### 5.1.1 Detailed Description

**Author**

Jonathan Enders enders@th.physik.uni-frankfurt.de

**Version**

1.0

### 5.1.2 DESCRIPTION

Calculate Hubbard parameters in inhomogeneous lattices using generalized Wannier functions.

armadillo (linear algebra library) and gsl (gnu scientific library) need to be installed

comp.sh can be executed to compile the program, type "./comp.sh"

the program can be found in a.out, to execute type "./a.out"

the program will try to create a folder named output. If such a folder is already present the program will fail.

after the program finished execution the scripts 'plot.sh' and 'plotm.sh' can be used to plot the data in the output directory:

```
./plot.sh output/Heigenfunctions.gpf
./plotm.sh output/HinW0.gpm
```

(gnuplot needs to be installed for plotting)

### 5.1.3 Macro Definition Documentation

#### 5.1.3.1 #define outputWithHeader( *x*, *header* )

**Value:**

```
file.open((dirName+string("/")+string(#x)+getFileExt(x)).c_str()); \
                                          file << header << endl << outputModifier(x) << endl; \
                                          file.close();
```

### 5.1.4 Function Documentation

#### 5.1.4.1 mt absUnityOp ( mt *par* )

needed as input for romberg integration procedure

#### 5.1.4.2 FsOverTS calcFsOverTS ( mt const *inhStrength* )

calculates measures used for the band assigning

#### 5.1.4.3 Mat$<$cx_double$>$ calculateEigenstatesBlockwise ( Mat$<$ cx_double $>$ const *symmetricMatrix,* Col$<$ mt $>$ & *eigenvals,* unsigned *statesPerBand* )

Given a block-diagonal matrix this calculates the eigenvalues and eigenstates for each block individually

#### 5.1.4.4 Mat$<$cx_double$>$ calculateOverlapMatrix ( vector$<$ fct$<$ complex$<$ double $>$ $>$ $>$ const & *Es* )

calculates the overlap matrix of the argument functions using a real space integration

#### 5.1.4.5 mt constantOp ( mt *par* )

needed as input for romberg integration procedure

#### 5.1.4.6 Mat<mt> createDiagonalMatrixFromVector ( Col< mt > const *eigenvalues* )

Puts the elements of an input vector on the diagonal of an output matrix

#### 5.1.4.7 void dglSolve ( ODE< physical_parameters > *& *D,* unsigned *eqncount,* mt const *xinit,* mt *const & *yinit* )

Interfacing with the differential equation solver.

#### 5.1.4.8 unsigned findClosest ( mt const & *value,* Col< mt > const & *vec,* mt & *error* )

return the index of the element in vec, which is closest to value

#### 5.1.4.9 mt invPropagateFunction ( mt *par* )

**See Also**

propagateFunction This is needed because 'gsl' only supplies a minimum finder. The minimum of this function is a maximum for propagateFunction.

#### 5.1.4.10 mt invPropagateFunction ( mt *par,* void * )

**See Also**

invPropagateFunction

#### 5.1.4.11 void linAssign ( mt const *start,* mt const *end,* FsOverTS const & *startF,* FsOverTS & *endF* )

assign states to bands assuming a linear behaviour with respect to the inhomogeneity strength

#### 5.1.4.12 int main ( int *argc,* char ** *argv* )

THE USER INTERFACE

Here all the important parameters are set to determine the physical system. And to control the numerics.

#### 5.1.4.13 void ODE_derivs ( const double *x,* double * *y,* double * *dydx,* physical_parameters & *pp* )

Setting derivatives for the differential equation solver. Given by the effective stationary Schroedinger equation

#### 5.1.4.14 void outputPropagatedWavefunctionEndValueOverE ( mt const *Estart,* mt const *Eend,* mt const *step* )

This function outputs the results of the propagation procedure for different energies to a file

#### 5.1.4.15 mt propagateFunction ( mt *par* )

Calls the differential equation solver with a certain energy 'par' set and returns the value that needs to be zero for a valid eigenenergy state. This function is used by the root finding procedure.

#### 5.1.4.16 mt propagateFunction ( mt *par*, void $*$ )

**See Also**

propagateFunction

#### 5.1.4.17 template<typename fctRangeType , typename Fret > fctRangeType romberg_integral_f ( fct< fctRangeType > *func1*, Fret($*$)(mt) *f*, double *romberg_precision* = `romberg_integration_precision` )

Romberg integration routine taken from numerical recipes in C++, the file qromb.c.

initially declared as a static variable in trapez_int_refined - pass this ba reference

determines quality: if too low the method seems to sometime have thought that it has converged, where it hasn't

#### 5.1.4.18 template<typename fctRangeType , typename Fret > fctRangeType romberg_integral_f ( fct< fctRangeType > *func1*, fct< fctRangeType > *func2*, Fret($*$)(mt) *f*, double *romberg_precision* = `romberg_integration_precision` )

Romberg integration routine taken from numerical recipes in C++, the file qromb.c.

initially declared as a static variable in trapez_int_refined - pass this ba reference

determines quality: if too low the method seems to sometime have thought that it has converged, where it hasn't

#### 5.1.4.19 void saveFsOverTS ( mt const *inhStrength*, FsOverTS const & *F* )

saves information into the linAssignInfo struct

#### 5.1.4.20 mt squareOp ( mt *par* )

needed as input for romberg integration procedure

#### 5.1.4.21 template<typename fctRangeType , typename Fret > fctRangeType trapez_int_refined_f ( fct< fctRangeType > *func1*, Fret($*$)(mt) *f*, fctDomainType *x_lo*, fctDomainType *x_hi*, int *n*, fctRangeType & *s* )

the same integration procedure for a product of two functions

trapez integration from numerical recipes

#### 5.1.4.22 template<typename fctRangeType , typename Fret > fctRangeType trapez_int_refined_f ( fct< fctRangeType > *func1*, fct< fctRangeType > *func2*, Fret($*$)(mt) *f*, fctDomainType *x_lo*, fctDomainType *x_hi*, int *n*, fctRangeType & *s* )

the same integration procedure for a product of two functions

trapez integration from numerical recipes

#### 5.1.4.23 mt unityOp ( mt *par* )

needed as input for romberg integration procedure

#### 5.1.4.24 complex< mt > V ( mt *x*, physical_parameters & *pp* )

The potential of inhomogeneous system, i.e. lattice + inhomogeneity

#### 5.1.4.25 complex< mt > Vinh ( mt *x* )

The potential of inhomogeneity only quasi disorder

#### 5.1.4.26 mt Vlat ( mt *x*, physical_parameters & *pp* )

The homogeneous system potential

### 5.1.5 Variable Documentation

#### 5.1.5.1 const char∗ argp_program_bug_address

**Initial value:**

```
=
"<jonathanenders@gmx.de>"
```

#### 5.1.5.2 const char∗ argp_program_version

**Initial value:**

```
=
"inWan 1.0"
```

#### 5.1.5.3 fileStdoutStream fout

**See Also**

fileStdoutStream

#### 5.1.5.4 unsigned statesPerBand

The amount of states in a inhomogeneous system band. Equals the amount of inhomogeneous system lattice sites.

#### 5.1.5.5 unsigned statesPerBand0

The amount of states in a homogeneous system band. Equals the amount of homogeneous system lattice sites.

# Index