Bose-Hubbard Model Transition Study

1.0

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BH Class Reference

Class representing the Bose-Hubbard Hamiltonian.

`#include <hamiltonian.h>`

Public Member Functions

- BH (const std::vector< std::vector< int > > &neighbours, int m_, int n_, double J_, double U_, double mu_)
  Constructor for the Bose-Hubbard Hamiltonian.
- Eigen::SparseMatrix< double > getHamiltonian () const
  Return the Hamiltonian sparse matrix.

Public Attributes

- Eigen::VectorXd interaction_matrix

3.1.1 Detailed Description

Class representing the Bose-Hubbard Hamiltonian.

This class implements the Hamiltonian for the Bose-Hubbard model, which describes interacting bosons on a lattice.

Warning

This class is not thread-safe.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BH()

```
BH::BH (
            const std::vector< std::vector< int > > & neighbours,
            int m_,
            int n_,
            double J_,
            double U_,
            double mu_ )
```

Constructor for the Bose-Hubbard Hamiltonian.

CONSTRUCTOR ///.

Parameters

neighbours	Vector that contains the neighbours of each site of the lattice.
m	Number of sites in the lattice.
n	Number of bosons in the lattice.
J	Hopping parameter of the BH model.
U	Interaction parameter of the BH model.
mu	Chemical potential of the BH model.

3.1.3 Member Function Documentation

3.1.3.1 getHamiltonian()

```
Eigen::SparseMatrix< double > BH::getHamiltonian ( ) const
```

Return the Hamiltonian sparse matrix.

UTILITY FUNCTIONS ///.

Returns

Eigen::SparseMatrix<double> The Hamiltonian sparse matrix.

3.1.4 Member Data Documentation

3.1.4.1 interaction_matrix

```
Eigen::VectorXd BH::interaction_matrix
```

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

- include/hamiltonian.h
- src/hamiltonian.cpp

3.2 Neighbours Class Reference

Class to generate the list of neighbours for a 1D chain, 2D square lattice, or 3D cubic lattice.

```
#include <neighbours.h>
```

Public Member Functions

- Neighbours (int m)

  Constructor for the Neighbours class.
- ~Neighbours ()

  Destructor for the Neighbours class.
- void chain_neighbours (bool closed=true)

  Generate the list of neighbours for a 1D chain.
- void square_neighbours (bool closed=true)

  Generate the list of neighbours for a 2D square lattice.
- void cube_neighbours (bool closed=true)

  Generate the list of neighbours for a 2D square lattice.
- std::vector< std::vector< int > > getNeighbours () const

  Return the list of neighbours.

3.2.1 Detailed Description

Class to generate the list of neighbours for a 1D chain, 2D square lattice, or 3D cubic lattice.

Warning

This class is not thread-safe.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Neighbours()

```
Neighbours::Neighbours (
            int m )
```

Constructor for the Neighbours class.

Parameters

m	Number of sites in the lattice.

3.2.2.2 ~Neighbours()

```
Neighbours::~Neighbours ( )
```

Destructor for the Neighbours class.

3.2.3 Member Function Documentation

3.2.3.1 chain_neighbours()

```
void Neighbours::chain_neighbours (
            bool closed = true )
```

Generate the list of neighbours for a 1D chain.

1D ///

Parameters

m	Number of sites in the chain.
closed	By default, closed = true for periodic boundary conditions, closed = false for open boundary conditions.

Warning

Ensure that the number of sites is greater than 1.

3.2.3.2 cube_neighbours()

```
void Neighbours::cube_neighbours (
            bool closed = true )
```

Generate the list of neighbours for a 2D square lattice.

3D ///

Parameters

m	Number of sites in the square.
closed	By default, closed = true for periodic boundary conditions, closed = false for open boundary conditions.

Warning

Ensure that the number of sites is a perfect cube.

3.2.3.3 getNeighbours()

```
std::vector< std::vector< int > > Neighbours::getNeighbours ( ) const
```

Return the list of neighbours.

UTILITY FUNCTIONS ///.

Returns

std::vector<std::vector<int>> The list of neighbours for each site of the lattice.

3.2.3.4 square_neighbours()

```
void Neighbours::square_neighbours (
            bool closed = true )
```

Generate the list of neighbours for a 2D square lattice.

2D ///

Parameters

m	Number of sites in the square.
closed	By default, closed = true for periodic boundary conditions, closed = false for open boundary conditions.

Warning

Ensure that the number of sites is a perfect square.

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

- include/neighbours.h
- src/neighbours.cpp

3.3 Operator Class Reference

Class representing an operator in a quantum system.

`#include <operator.h>`

Public Member Functions

- Operator (Eigen::SparseMatrix< double > &&smatrix)

  Constructor for the Operator class.
- int size () const

  Get the size of the matrix.
- Operator & operator+ (const Operator &operand)

  Add a matrix to an operand of type SparseMatrix with same size.
- Operator & operator* (const Operator &multiplicand)

  Multiply a sparse matrix by a multiplicand of type SparseMatrix with same size.
- Eigen::VectorXd operator* (const Eigen::VectorXd &vector) const

  Multiply a sparse matrix by a vector with concomitant size.
- Operator & operator* (double scalar)

  Multiply a sparse matrix by a scalar.
- Eigen::VectorXcd IRLM_eigen (int nb_eigen) const

  Calculate the approximate eigenvalues and eigenvectors of the Hamiltonian using the Implicitly Restarted Lanczos Method.
- Eigen::VectorXd FOLM_eigen (int nb_iter, Eigen::MatrixXd &eigenvectors) const

  Calculate the approximate eigenvalues and eigenvectors of the Hamiltonian using the Full Orthogonalization Lanczos Method.
- Eigen::VectorXd exact_eigen (Eigen::MatrixXd &eigenvectors) const

  Calculate the exact eigenvalues and eigenvectors of the Hamiltonian by an exact diagonalization.
- double order_parameter (const Eigen::VectorXd &eigenvalues, const Eigen::MatrixXd &eigenvectors) const

  Calculate the order parameter of the system.
- double gap_ratio ()

  Calculate the energy gap ratio of the system.
- void add_chemical_potential (double mu, int n)

  Add a potential term to the operator.

- void add_interaction (double U, const Eigen::VectorXd &basis)

  Add an interaction term to the operator.
- double partition_function (const Eigen::VectorXd &eigenvalues, double temperature) const

  Calculate the partition function Z for an already diagonalized Hamiltonian.
- void canonical_density_matrix (const Eigen::VectorXd &eigenvalues, double temperature) const

  Calculate the canonical density matrix for an already diagonalized Hamiltonian.
- double boson_density (double dmu, int n)

  Calculate the boson density of the system.
- double compressibility (double dmu, int n)

  Calculate the isothermal compressibility of the system.

Static Public Member Functions

- static Operator Identity (int size)

  Create the operator Identity.

3.3.1 Detailed Description

Class representing an operator in a quantum system.

This class provides methods for initializing, manipulating, and diagonalizing operators represented as sparse matrices.

Warning

This class is not thread-safe.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Operator()

```
Operator::Operator (
            Eigen::SparseMatrix< double > && smatrix )
```

Constructor for the Operator class.

CONSTRUCTOR ///.

Parameters

smatrix	The sparse matrix to initialize the operator.

3.3.3 Member Function Documentation

3.3.3.1 add_chemical_potential()

```
void Operator::add_chemical_potential (
```

```
            double mu,
            int n )
```

Add a potential term to the operator.

SPECIFIC CALCULATIONS ///.

3.3.3.2 add_interaction()

```
void Operator::add_interaction (
            double U,
            const Eigen::VectorXd & basis )
```

Add an interaction term to the operator.

3.3.3.3 boson_density()

```
double Operator::boson_density (
            double dmu,
            int n )
```

Calculate the boson density of the system.

Parameters

dmu	The difference of chemical potential.
n	The number of bosons.

Returns

double The boson density.

Warning

This function modifies the values of the operator from mu to mu + dmu.

3.3.3.4 canonical_density_matrix()

```
void Operator::canonical_density_matrix (
            const Eigen::VectorXd & eigenvalues,
            double temperature ) const
```

Calculate the canonical density matrix for an already diagonalized Hamiltonian.

Parameters

eigenvalues	The vector of eigenvalues.
temperature	The temperature.

3.3.3.5 compressibility()

```
double Operator::compressibility (
            double dmu,
            int n )
```

Calculate the isothermal compressibility of the system.

Parameters

dmu	The difference of chemical potential.
n	The number of bosons.

Returns

double The compressibility.

Warning

This function modifies the values of the operator from mu to mu + dmu.

3.3.3.6 exact_eigen()

```
Eigen::VectorXd Operator::exact_eigen (
            Eigen::MatrixXd & eigenvectors ) const
```

Calculate the exact eigenvalues and eigenvectors of the Hamiltonian by an exact diagonalization.

EXACT DIAGONALIZATION ///.

Parameters

eigenvectors	An empty matrix to store the eigenvectors.

Returns

Eigen::Matrix<double> The vector of eigenvalues.

Warning

This function is computationally expensive and should be used with caution.

3.3.3.7 FOLM_eigen()

```
Eigen::VectorXd Operator::FOLM_eigen (
            int nb_iter,
            Eigen::MatrixXd & eigenvectors ) const
```

Calculate the approximate eigenvalues and eigenvectors of the Hamiltonian using the Full Orthogonalization Lanczos Method.

Parameters

nb_iter	The number of iterations.
eigenvectors	An empty matrix to store the eigenvectors.

Returns

Eigen::VectorXd The vector of eigenvalues.

Warning

Ensure that nb_iter is greater than 1. The calculation might be wrong if the number of iterations is too low.

3.3.3.8 gap_ratio()

```
double Operator::gap_ratio ( )
```

Calculate the energy gap ratio of the system.

Parameters

eigenvalues	The vector of eigenvalues.

Returns

double The energy gap ratio.

3.3.3.9 Identity()

```
Operator Operator::Identity (
            int size )  [static]
```

Create the operator Identity.

IDENTITY OPERATOR ///.

Parameters

size	The size of the identity matrix.

Returns

Operator The identity operator.

3.3.3.10 IRLM_eigen()

```
Eigen::VectorXcd Operator::IRLM_eigen (
```

```
            int nb_eigen ) const
```

Calculate the approximate eigenvalues and eigenvectors of the Hamiltonian using the Implicitly Restarted Lanczos Method.

IMPLICITLY RESTARTED LANCZOS METHOD (IRLM) ///.

Parameters

nb_eigen	The number of eigenvalues to calculate.
eigenvectors	An empty matrix to store the eigenvectors.

Returns

Eigen::Matrix<double> The vector of eigenvalues.

Warning

Ensure that nb_eigen is greater than 1.

3.3.3.11 operator*() [1/3]

```
Eigen::VectorXd Operator::operator* (
            const Eigen::VectorXd & vector ) const
```

Multiply a sparse matrix by a vector with concomitant size.

Parameters

vector	The vector to multiply.

Returns

Eigen::Matrix<T, Eigen::Dynamic, 1> The result of the multiplication.

3.3.3.12 operator*() [2/3]

```
Operator & Operator::operator* (
            const Operator & multiplicand )
```

Multiply a sparse matrix by a multiplicand of type SparseMatrix with same size.

Parameters

multiplicand	The matrix to multiply.

Returns

Operator The result of the multiplication.

Warning

This function modifies the values of the operator.

3.3.3.13 operator∗() [3/3]

```
Operator & Operator::operator* (
            double scalar )
```

Multiply a sparse matrix by a scalar.

Parameters

scalar	The scalar to multiply.

Returns

Operator The result of the multiplication.

Warning

This function modifies the values of the operator.

3.3.3.14 operator+()

```
Operator & Operator::operator+ (
            const Operator & operand )
```

Add a matrix to an operand of type SparseMatrix with same size.

ADDITION AND MULTIPLICATION ///.

Parameters

operand	The matrix to add.

Returns

Operator The result of the addition.

Warning

This function modifies the values of the operator.

3.3.3.15 order_parameter()

```
double Operator::order_parameter (
            const Eigen::VectorXd & eigenvalues,
            const Eigen::MatrixXd & eigenvectors ) const
```

Calculate the order parameter of the system.

Parameters

eigenvalues	The vector of eigenvalues.
eigenvectors	The matrix of eigenvectors.

Returns

double The order parameter.

3.3.3.16 partition_function()

```
double Operator::partition_function (
            const Eigen::VectorXd & eigenvalues,
            double temperature ) const
```

Calculate the partition function Z for an already diagonalized Hamiltonian.

Parameters

eigenvalues	The vector of eigenvalues.
temperature	The temperature.

Returns

double The partition function.

3.3.3.17 size()

```
int Operator::size ( ) const
```

Get the size of the matrix.

UTILITY FUNCTIONS ///.

Returns

int The size of the matrix.

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

- include/operator.h
- src/operator.cpp

Chapter 4

File Documentation

4.1 include/hamiltonian.h File Reference

```
#include <vector>
#include <Eigen/Dense>
#include <Eigen/SparseCore>
#include "Eigen/src/Core/Matrix.h"
```

Include dependency graph for hamiltonian.h: This graph shows which files directly or indirectly include this file:

Classes

- class BH

  Class representing the Bose-Hubbard Hamiltonian.

4.2 hamiltonian.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include<vector>
00004 #include<Eigen/Dense>
00005 #include<Eigen/SparseCore>
00006 #include "Eigen/src/Core/Matrix.h"
00007
00008
00017 class BH {
00018 private:
00019
00020 // PARAMETERS OF THE BH MODEL
00021
00022     std::vector<std::vector<int» neighbours; // Vector that contains the neighbours of each site of
      the chain
00023
00024     int m; // Number of sites in the chain
00025     int n; // Number of bosons in the chain
00026     int D; // Dimension of the Hilbert space of the system
00027
00028     double J; // Hopping parameter of the BH model
00029     double U; // Interaction parameter of the BH model
00030     double mu; // Chemical potential of the BH model
00031
00032     Eigen::SparseMatrix<double> H; // Sparse matrix representation of the Hamiltonian
00033
00034 // DIMENSION OF THE HILBERT SPACE
00035
00036     /* calculate the dimension of the Hilbert space for n bosons on m sites */
```

```
00037     int binomial(int n, int k) const; // Binomial coefficient
00038     int dimension(int m, int n) const; // Dimension of the Hilbert space
00039
00040 // ELEMENTARY FUNCTIONS
00041
00042     /* calculate the sum of the elements of a vector between 2 index */
00043     int sum(const Eigen::VectorXd& state, int index1, int index2) const;
00044
00045 // INITIALIZE THE HILBERT SPACE BASIS
00046
00047     /* calculate the next state of the Hilbert space in lexicographic order */
00048     bool next_lexicographic(Eigen::VectorXd& state, int m, int n) const;
00049
00050     /* creates a matrix that has the vectors of the Hilbert space basis in columns */
00051     Eigen::MatrixXd init_lexicographic(int m, int n) const;
00052
00053 // SORT THE HILBERT SPACE BASIS TO FACILITATE CALCULUS
00054
00055     /* calculate the unique tag of the kth column of the matrix */
00056     double calculate_tag(const Eigen::MatrixXd& basis, const std::vector<int>& primes, int k) const;
00057
00058     /* calculate and store the tags of each state of the Hilbert space basis */
00059     Eigen::VectorXd calculate_tags(const Eigen::MatrixXd& basis, const std::vector<int>& primes)
      const;
00060
00061     /* sort the states of the Hilbert space by ascending order compared by their tags */
00062     void sort_basis(Eigen::VectorXd& tags, Eigen::MatrixXd& basis) const;
00063
00064     /* gives the index of the wanted tag x by the Newton method */
00065     int search_tag(const Eigen::VectorXd& tags, double x) const;
00066
00067 // FILL THE HAMILTONIAN OF THE SYSTEM
00068
00069     /* fill the hopping term of the Hamiltonian */
00070     void fill_hopping(const Eigen::MatrixXd& basis, const Eigen::VectorXd& tags, const
      std::vector<std::vector<int>& neighbours, const std::vector<int>& primes, Eigen::SparseMatrix<double>&
      hmatrix, double J) const;
00071
00072     /* fill the interaction term of the Hamiltonian */
00073     void fill_interaction(const Eigen::MatrixXd& basis, Eigen::SparseMatrix<double>& hmatrix, double
      U) const;
00074
00075     /* fill the chemical potential term of the Hamiltonian */
00076     void fill_chemical(const Eigen::MatrixXd& basis, Eigen::SparseMatrix<double>& hmatrix, double mu)
      const;
00077
00078 public:
00079
00080     Eigen::VectorXd interaction_matrix; // Vector that contains the interaction matrix elements
00081
00082 // CONSTRUCTOR
00083
00094     BH(const std::vector<std::vector<int>& neighbours, int m_, int n_, double J_, double U_, double
      mu_);
00095
00096 // UTILITY FUNCTIONS
00097
00103     Eigen::SparseMatrix<double> getHamiltonian() const;
00104 };
```

4.3 include/neighbours.h File Reference

```
#include <vector>
#include <cmath>
```

Include dependency graph for neighbours.h: This graph shows which files directly or indirectly include this file:

Classes

- class Neighbours

  Class to generate the list of neighbours for a 1D chain, 2D square lattice, or 3D cubic lattice.

4.4 neighbours.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <vector>
00004 #include <cmath>
00005
00006
00013 class Neighbours {
00014 public:
00015
00021     Neighbours(int m);
00022
00026     ~Neighbours();
00027
00035     void chain_neighbours(bool closed = true);
00036
00044     void square_neighbours(bool closed = true);
00045
00053     void cube_neighbours(bool closed = true);
00054
00059     std::vector<std::vector<int» getNeighbours() const;
00060
00061 private:
00062     int m;
00063     std::vector<std::vector<int» neighbours;
00064 };
```

4.5 include/operator.h File Reference

```
#include <cmath>
#include <Eigen/Dense>
#include <Eigen/SparseCore>
#include <Eigen/Eigenvalues>
#include <Spectra/GenEigsSolver.h>
#include <Spectra/MatOp/SparseGenMatProd.h>
```

Include dependency graph for operator.h: This graph shows which files directly or indirectly include this file:

Classes

- class Operator

  Class representing an operator in a quantum system.

4.6 operator.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <cmath>
00004 #include <Eigen/Dense>
00005 #include <Eigen/SparseCore>
00006 #include <Eigen/Eigenvalues>
00007 #include <Spectra/GenEigsSolver.h>
00008 #include <Spectra/MatOp/SparseGenMatProd.h>
00009
00010
00018 class Operator {
00019 private:
00020
00021 // INITIALIZATION :
00022
00023     Eigen::SparseMatrix<double> O;
00024     int D; // dimension of the Hilbert space of the system
00025     double ref; // threshold under which a value is considered null
```

```
00026
00027 // DIAGONALIZATION :
00028
00029     /* implement the Full Orthogonalization Lanczos Method for a sparse matrix for nb_iter iterations
      starting with vector v_0 */
00030     void FOLM_diag(int nb_iter, Eigen::VectorXd& v_0, Eigen::MatrixXd& T, Eigen::MatrixXd& V) const;
00031
00032     /* sort eigenvalues and eigenvectors in descending order */
00033     void sort_eigen(Eigen::VectorXd& eigenvalues, Eigen::MatrixXd& eigenvectors) const;
00034
00035 public:
00036
00037 // CONSTRUCTOR :
00038
00044     Operator(Eigen::SparseMatrix<double>&& smatrix);
00045
00046 //UTILITY FUNCTIONS :
00047
00053     int size() const;
00054
00055 // BASIS OPERATIONS :
00056
00063     static Operator Identity(int size);
00064
00072     Operator& operator + (const Operator& operand);
00073
00081     Operator& operator * (const Operator& multiplicand);
00082
00089     Eigen::VectorXd operator * (const Eigen::VectorXd& vector) const;
00090
00098     Operator& operator * (double scalar);
00099
00100 // DIAGONALIZATION :
00101
00110     Eigen::VectorXcd IRLM_eigen(int nb_eigen) const;
00111
00120     Eigen::VectorXd FOLM_eigen(int nb_iter, Eigen::MatrixXd& eigenvectors) const;
00121
00129     Eigen::VectorXd exact_eigen(Eigen::MatrixXd& eigenvectors) const;
00130
00131
00132 // PHASE TRANSITION CALCULATIONS :
00133
00141     double order_parameter(const Eigen::VectorXd& eigenvalues, const Eigen::MatrixXd& eigenvectors)
      const;
00142
00149     double gap_ratio();
00150
00154     void add_chemical_potential(double mu, int n);
00155
00159     void add_interaction(double U, const Eigen::VectorXd& basis);
00160
00161 // THERMODYNAMICAL FUNCTIONS :
00162
00170     double partition_function(const Eigen::VectorXd& eigenvalues, double temperature) const;
00171
00178     void canonical_density_matrix(const Eigen::VectorXd& eigenvalues, double temperature) const;
00179
00188     double boson_density(double dmu, int n);
00189
00198     double compressibility(double dmu, int n);
00199
00200 };
```

4.7 src/hamiltonian.cpp File Reference

```
#include <vector>
#include <Eigen/Dense>
#include <Eigen/SparseCore>
#include "hamiltonian.h"
```

Include dependency graph for hamiltonian.cpp:

4.8 src/main.cpp File Reference

```
#include <Eigen/Dense>
#include <Eigen/SparseCore>
```

```
#include <Eigen/Eigenvalues>
#include <cmath>
#include <fstream>
#include <iostream>
#include <vector>
#include <chrono>
#include <complex>
#include <getopt.h>
#include "hamiltonian.h"
#include "operator.h"
#include "neighbours.h"
```

Include dependency graph for main.cpp:

Functions

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

4.8.1 Function Documentation

4.8.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

PARAMETERS OF THE MODEL

GEOMETRY OF THE LATTICE

PHASE TRANSITION CALCULATIONS

4.8.1.2 print_usage()

```
void print_usage ( )
```

4.9 src/neighbours.cpp File Reference

```
#include <vector>
#include <stdexcept>
#include <cmath>
#include "neighbours.h"
```

Include dependency graph for neighbours.cpp:

4.10 src/operator.cpp File Reference

```
#include <stdexcept>
#include <cmath>
#include <complex>
#include <Eigen/Dense>
#include <Eigen/SparseCore>
#include <Eigen/Eigenvalues>
#include <Spectra/GenEigsSolver.h>
#include <Spectra/MatOp/SparseGenMatProd.h>
#include "operator.h"
```

Include dependency graph for operator.cpp:

Index