Fermion Bag Hamiltonian Lattice Field Theory Code Manual

Joseph Buck
Emilie HuffmanDepartment of Physics, Wake Forest University, Winston-Salem, NC

August 15, 2025

Contents

1 Introduction

2 Theoretical Background

3 Software Overview

4 Installation and Setup

5 Input Parameters

5.1 Input File Format

The input files for this code are YAML files, so parameter specifications are in the form of key-value pairs. Variables are often nested under a few keys for increased clarity. There are four primary categories of variables that can be set in these input files: control, lattice, configuration, and constants.

5.2 Input Parameters

- `control` Control parameters are any variables determining the overall execution of the code.
  - `algorithm` The bond switching algorithm to use in the relaxation process. Currently, the only option is "random", but others will be introduced as development proceeds.
    - `random` This algorithm takes a starting configuration and randomly removes a bond from the configuration, randomly chooses a new tau, and randomly (using the bond type proportions) chooses the new bond size and lattice sites. This proceeds for nmoves iterations. For more information, see 7.
  - `nmoves` The number of times the configuration should be updated.
- `lattice` Lattice parameters include all variables needed to fully determine the lattice structure.
  - `type` The Bravais lattice type. The only supported type is "simple-cubic" in one dimension at the moment. Support will be added for up to three dimensions and a variety of Bravais lattice types.
  - `a, b, c` The axial lengths of the Bravais lattice in Bohr. If only a is supplied, the lattice will be one dimensional. If a and b are supplied it will be two, and if all three are supplied it will be three dimensional.

  - `alpha, beta, gamma` The angles completing the Bravais lattice definition in radians. If only a is supplied, these angles will be ignored. If a and b are supplied, only alpha is required. If a, b, and c are supplied, all angles are required.
  - `lims` The lims section is for defining information on the overall spatial extent of the lattice.
    * `x, y, z` Variables x, y, and x are subcategories of the lims section. If a is supplied, x must be set, etc.
      · `min` The minimum value in Bohr that the lattice should extend to in the given dimension.
      · `max_factor` Instead of setting a maximum limit for the given dimension, you choose the maximum number of multiples of the unit cell as a way of specifying the upper limit. This parameter, along with "base", determines the upper limit of the lattice.
      · `base` The multiplier which acts as the "unit" that max_factor is multiplied by to determine the upper boundary.

- `configuration` Configuration parameters fully determine the attributes of configurations throughout the progression of the code. For definitions of configurations and related concepts, see Sections 2 and 7.
  - `float_tol` The float tolerance determines how precise the tau variable needs to be when retrieving the bond at a given tau. Bonds are stored in a hashmap, mapping from the tau float variable to a bond object. If no floating point tolerance is specified, machine precision discrepancies could contribute to the code not recognizing a given tau. The default float tolerance is 1e-5.
  - `nbonds` The total number of bonds to be included in a configuration. Note that the actual number of bonds in a given configuration may be slightly less than nbonds in order to accomodate the bond type proportions (bond_type_props). The actual number of bonds will never be greater than nbonds. This variable by definition also determines the number of unique taus in a configuration.
  - `tau_max` The tau variable in the code is an "imaginary" time variable. This tau_max input parameter determines the maximum allowed value that tau is allowed to take.
  - `bond_type_props` Bonds in the code can be made up of one or more adjacent lattice sites. The purpose of this parameter is to specify which bond sizes should be included in the simulation, and

the relative proportions of each size. Bond sizes are given as keys under the bond_type_props section. The relative proportions are given as the values corresponding to each key. The proportions can be given as floats or integers. Renormalization and computation of the actual number of each type of bond is done in the code in conjunction with nbonds.

- `constants` This parameter section is the place to define useful constants that may be needed to fully specify the input. For example, Bravais angles for a simple-cubic lattice need to be set to pi/2. So, pi/2 would be a useful constant to store here. The code does not rely on these user constants to perform its calculations. Where those are needed, they are defined in the code.

5.3 Environment Variables

6 Output Files and Data

7 Physics Implementations

7.1 Data Structures

`Bond` Stores information about a single bond including the indices of its lattice sites and its size.

Data Members

- `int numSites` The number of sites in the bond.
- `std::set<int> indices` The indices of lattice sites in the bond. This structure works for one dimension and will need to be updated for three dimensions. For complicated geometries, the approach to storing the lattice sites for each bond will need to be revised.

Methods

- `Bond(const std::set<int>& indices)` This is the constructor for the Bond class. The indices input is a set of integers that represent the indices of the lattice that are participants in the bond. These lattice neighbors must be nearest neighbors. Additional considerations must be made once three dimensions and more complicated geometries are introduced.
- `int getNumSites() const` Returns the number of sites in the bond.
- `bool operator==(const Bond& other) const` This overloaded equality operator tests whether two Bond objects are equivalent by determining whether their indices arrays are equal.

- `bool operator!=(const Bond& other) const` This overloaded operator uses the overloaded equaulity operator to determine inequality.
- `friend std::ostream& operator<<(std::ostream& os, const Bond& bond)` This overloaded operator prints bond data out in a neat, easy to read format.

`Lattice` Stores information about the lattice including the spatial positions of lattice sites and the number of sites. Once higher dimensions are supported, numSites will need to be a vector of integers, and sites will need to be a three dimensional vector of floats.

Data Members

- `int numSites` The number of sites in the lattice.
- `std::vector<float> sites` The spatial positions of each site in one dimension. Only supports a single dimension at the moment.

Methods

- `Lattice(std::map<std::string, std::pair<float, float>> lims, std::map<std::string, int> npts)` This is the constructor for the Lattice class. The lims map it takes in is constructed from the lims input parameters. The string key of the lims map should be x, y, or z, and its value should be a pair with the first element being the minimum spatial position in the lattice, and the second element being the maximum. The npts input map has a string key of x, y, or z, and an integer defining the number of points in that dimension. This setup is currently specific only to a simple-cubic geometry in three dimensions. The return value is a Lattice object.
- `int getNumSites()` Returns the number of sites in the lattice. Once higher dimensions are supported, this will be a vector npts.
- `float operator[](const int index) const` This overloaded operator returns the spatial position contained in the lattice at the given index. Eventually this will support three dimensions: [x_ind, y_ind, z_ind].

`Configuration` Stores information about the current configuration including a map from taus to Bonds, and the tolerance used to determine uniqueness of a taus.

Data Members

- `double tolerance` The tolerance/precision to use when accessing bonds in the configuration by their tau key. The default tolerance is 1e-5.

- `std::map<double, Bond> bonds` A map from the "imaginary" time tau, a double, to a Bond type.

Methods

- `Configuration(double tol)` This is the constructor for the Configuration class. It takes in a tolerance parameter and returns an empty Configuration object.
- `void addBond(double tau, Bond& newBond)` Adds a bond with a given tau and Bond to the configuration.
- `void addBonds(std::vector<double> taus, std::vector<Bond> newBonds)` Adds multiple bonds to a configuration at once using a vector of taus and a vector of Bonds. Calls the addBond method.
- `void delBond(double tau)` Deletes a Bond from the configuration by its tau.
- `void delBonds()` Deletes all bonds from a configuration.
- `const Bond& getBond(double tau) const` Returns a Bond object from its tau value.
- `const std::map<double, Bond>& getBonds() const` Returns the bonds map, which is a map of all Bonds in the Configuration.
- `int getNumBonds() const` Returns the number of Bonds (number of unique taus) in the Configuration.
- `bool operator==(const Configuration& other) const` An overloaded equality operator which tests the equality of two Configurations by testing the equality of the number of bonds in each Configuration and the equality of each Bond associated with each tau.
- `bool operator!=(const Configuration& other) const` An overloaded inequality operator which calls the overloaded equality operator to determine if two Configurations are not equivalent.
- `friend std::ostream& operator<<(std::ostream& os, const Configuration& configuration)` An overloaded output operator which facilitates easy viewing of a Configuration object.
- `private double truncateToTolerance(double key) const` A function which rounds a double to a given tolerance. The tolerance should already be stored in the Configuration before this method is ever called.