Appendix A

Appendix

A.1 Code Usage

The routines presented in the github repo are used to determine a set of effective Hamiltonian model parameters from another set of parameters representing the full system. The code is written as a mixture of Julia and C++ programs where a short Julia script is used to setup the main calculation which is later done in the C++ ITensor library[18]. Note that there is now a Julia version of the ITensor library(`https://github.com/ITensor/ITensors.jl`), which can be used to streamline the overall process.

A.1.1 Setting up a Calculation

The first step is to define the structure of the new Hamiltonian and initialize it with values from the original system. This will decide things such as the extent of the hopping terms(nearest-neighbor, next nearest-neighbor, etc.) and the range of the two-body interactions. We begin with `PrepEffH.jl` which produces a set of output files that can then be fed into `EffHubbardBFGS`.

PrepEffH.jl

A list of all the inputs and descriptions can be seen by calling `julia PrepEffH.jl --help`. For convenience we list these here

```
usage: PrepEffH.jl [-o O] [--sym] [--cap_edge CAP_EDGE] [-h] N fn_tij
                   fn_Vij N_terms_tij N_terms_Vij

positional arguments:
  N                     System size(number of atoms)
  fn_tij                File containing the single-particle terms for
                        the full Hamiltonian in the form [...; i j tij;
                        i (j+1) ti(j+1); ...]
  fn_Vij                File containing the two-particle terms for the
                        full Hamiltonian in the form [...; i j Vij; i
                        (j+1) Vi(j+1); ...]
  N_terms_tij           Number of variational terms in tij(including
                        onsite term) (type: Int64)
  N_terms_Vij           Number of variational terms in Vij(including
                        onsite term) (type: Int64)

optional arguments:
  -o O                  Directory into which to output results
                        (default: ".")
  --sym                 Flag, when called, symmetrized tij and Vij
```

```
                        about the center of the 1D chain
  --cap_edge CAP_EDGE  Bounds the maximum edge region to cap_edge
                        (type: Int64, default: 10000)
  -h, --help            show this help message and exit
```

The `N_terms_tij` and `N_terms_Vij` require some more clarification. These are essentially the range of the one- and two-particle interactions, respectively. For example, setting `N_terms_tij` to 2 will generate two variational parameters for the single-particle terms, one that acts as a chemical potential the other acting as a nearest-neighbor hopping

$$t_ij = \begin{cases} \mu & i = j \\ t & |i - j| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

The same is done for the two-particle terms defined by `N_terms_Vij`. This is a consequence of the translational invariance of the 1D system.

The code then creates a directory named `EffH*`(name varies depending on the input values) which is saved to the path specified by the `-o` flag and outputs several files listed below

`tij_eff.txt`

The initialized one-particle terms of the new model in a three-column format `[i j tij]`

`Vij_eff.txt`

The initialized two-particle terms of the new model in a three-column format `[i j Vij]` `tij_full.txt`

The original one-particle terms in a three-column format `[i j tij]`

`Vij_full.txt`

The original two-particle terms in a three-column format `[i j Vij]`

`tijvar.txt`

The variable IDs(1 for the first variational parameter, 2 for the second variational parameter, etc.) for the one-particle terms in a three-column format `[i j <variable ID>]`

`Vijvar.txt`

The variable IDs(1 for the first variational parameter, 2 for the second variational parameter, etc.) for the two-particle terms in a three-column format `[i j <variable ID>]`

Walking through the Code The main function is called `main` and begins with importing the one- and two-particle terms from the provided file names, calling a helper function `to_mat` which converts these into $N \times N$ matrices, `tij` and `Vij` in the code. Next, the variable matrices, `tij_var_mat` and `Vij_var_mat`, are created using the function `get_var_mats` which define the structure of the model Hamiltonian. The routine takes into account any differences the user wants to include near the edges(specified by `--cap_edge`) and produces matrices which take the form

$$V = \begin{pmatrix} v_1 & v_4 & v_6 & 0 & \cdots & 0 & 0 & 0 & 0 \\ v_4 & v_2 & v_5 & v_6 & \cdots & 0 & 0 & 0 & 0 \\ v_6 & v_5 & v_3 & v_5 & \cdots & 0 & 0 & 0 & 0 \\ 0 & v_6 & v_5 & v_3 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & v_3 & v_5 & v_6 & 0 \\ 0 & 0 & 0 & 0 & \cdots & v_5 & v_3 & v_5 & v_6 \\ 0 & 0 & 0 & 0 & \cdots & v_6 & v_5 & v_2 & v_4 \\ 0 & 0 & 0 & 0 & \cdots & 0 & v_6 & v_4 & v_1 \end{pmatrix} \tag{A.2}$$

for `N_terms_Vij = 3`. In the example above, the main diagonal contains two sites near the edge which are allowed to break translational invariance as well as one site for the $|i-j| = 1$ terms. In this case, the edge regions are defined by the procedure outlined in section 4.3 for a large system size of 100 atoms. The classification of "edge" versus "bulk" for this system is shown in Fig. A.1. However, if the same is done for the one-particle terms, the procedure fails.

As shown in the blue curve in Fig. A.1(b), the "edge" region determined by using the previous method for the 0^{th} diagonal of t_{ij} is quite large and therefore ill-defined. This can be traced back to the incorporation of the electron-lattice interaction terms from the original hydrogen chain into the lattice model.

The original hydrogen chain accounts for the electron-lattice interaction by adding an on-site potential of the form

$$U_n = \sum_{n'} V_{nn'} \tag{A.3}$$

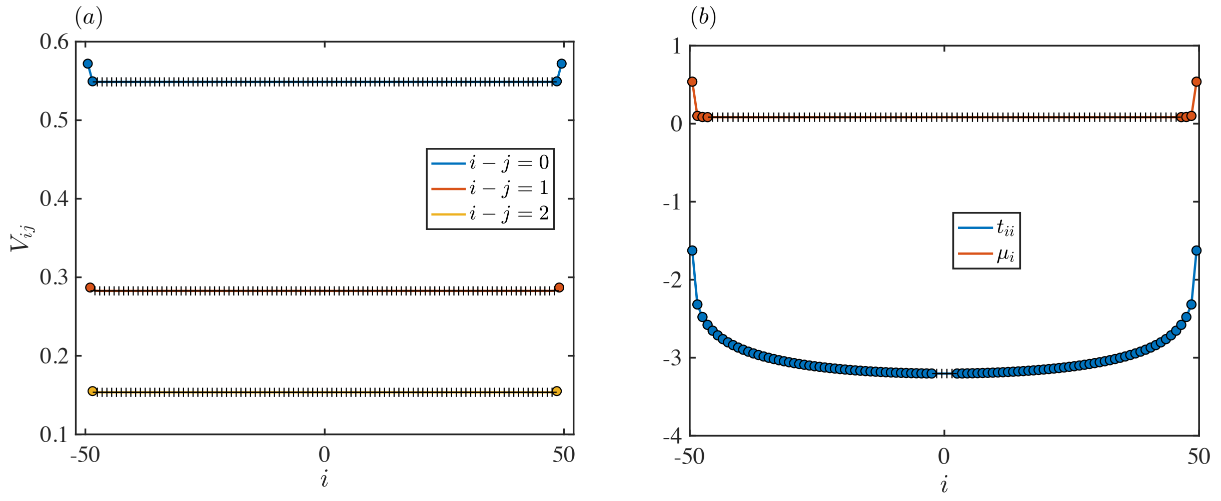

Figure A.1: (a) The edge regions for the two-particle interactions defined by the procedure in section 4.3. The colored circles denote "edges" and the black crosses, the "bulk". (b) The edge regions for the one-particle terms defined in the same scheme for the original values of the on-site terms in blue and the new terms calculated as a chemical potential by measuring the density-density interactions with respect to half-filling in red. Note that in the latter, the edge regions are much more well defined.

where n and n' enumerate the slices. After transforming to the lattice representation using the Wannier functions, this term gets integrated into the diagonal of t_{ij}(among other terms, but less so due to the highly localized nature of the Wannier functions), which couples the values of the interactions with those of the single-particle terms. The Wannier functions mix the two in a complicated way, but we can guess that the electron-lattice interaction of the lattice model takes on a similar form to that of the hydrogen chain. From this perspective, we can express $\hat{H}_{full}$ as,

$$\hat{H}_{WF} = \sum_{i\neq j} t_{ij}\hat{c}^{\dagger}_{\sigma i}\hat{c}_{\sigma j} + \frac{1}{2}\sum_{ij} V_{ij}\hat{n}_i\hat{n}_j + \sum_i \left(\mu_i - \sum_j V_{ij}\right)\hat{n}_i \tag{A.4}$$

where $\mu_i \equiv t_{ii} + \sum_j V_{ij}$. This is nothing more than writing a Hamiltonian with the density-density interactions measured with respect to half-filling.

At first, this seems like a trivial modification, but after calculating μ_i at different system sizes, we can see that it behaves as a positive chemical potential with a clearly defined "bulk" and "edge" region(see Fig. A.1(b)). Together with the "bulk" and "edge" regions defined for $\tilde{t}_{ij}$ and $\tilde{V}_{ij}$, $\hat{H}_{eff}$ can now be expressed in terms of a small number of well regularized variational parameters as,

$$\hat{H}_{eff} = \sum_{i \neq j} \tilde{t}_{ij} \hat{c}^{\dagger}_{\sigma i} \hat{c}_{\sigma j} + \frac{1}{2} \sum_{ij} \tilde{V}_{ij} \hat{n}_i \hat{n}_j + \sum_i \left(\tilde{\mu}_i - \sum_j \tilde{V}_{ij} \right) \hat{n}_i \tag{A.5}$$

In the code, this is taken into account by adjusting the diagonal elements of the effective one-particle terms. Once the variables are defined, all the output files are written to disc. The next step is to call `EffHubbardBFGS` within the newly created directory to run the optimization.

EffHubbardBFGS

After using `PrepEffH.jl` to create the directory `EffH*`, we can call `EffHubbardBFGS` in that same directory to begin running the optimization. The optimization is carried out with the help of the ITensor library and a minimization routine taken from [55]. In the current framework, we choose the Broyden-Fletcher-Goldfarb-Shanno(BFGS) algorithm to minimize 4.4 and find the effective model parameters. The BFGS routine is written in `BFGS.h`. The main inputs are pointers to functions which can evaluate 4.4 and it's derivatives called `*func` and `*dfunc` in the code. The functions themselves are written as sub-routines in `EffHubbardBFGS` called `dmrg_func` and `ddmrg_func`.

Walking through the Code The program begins by first reading in the files generated by `PrepEffH.jl`. It then constructs and stores the full Hamiltonian as an MPO which will later be used by `dmrg_func` and `ddmrg_func` to evaluate the optimization function. An initial Neél state wavefunction is then used as a starting point for the first DMRG calculation on the effective model giving $|\Psi_e ff\rangle_0$. At this stage, we have all the ingredients to begin the optimization and call the BFGS routine.

Each step of BFGS will call `dmrg_func` and `ddmrg_func` a certain number of times depending on the results of the line search algorithm. Each call requires a specific number of DMRG sweeps which can be controlled by modifying these functions. Currently, the code is configured to run one sweep for each call to `dmrg_func` and $2N_{\text{var}}$ sweeps for each call to `ddmrg_func`, where N_{var} is the number of variables.

After the optimization has converged, the number of iterations and function calls are printed to the screen as well as saved to disc. The newly found model parameters are stored in the files `BFGS_newtij_eff.txt` and `BFGS_newVij_eff.txt` in the three-column format described in the previous section.

AltOptimize.jl

Optimizing both the one- and two-particle terms at the same time can cause the procedure to become stuck in local minima. We can mitigate this effect by minimizing the terms seperately, first minimizing with respect to the one-particle terms, then fix these for an optimization of the two-particle terms, and repeating in this fashion until the minimization converges. This is the purpose of `AltOptimize.jl`. It is a short Julia script that calls `EffHubbardBFGS` in the sequential manner just described and keeps track of the terms at each step, saving the output of `EffHubbardBFGS` after each call. After running `PrepEffH.jl`, we can run `AltOptimize.jl` in the `EffH*` directory to begin this alternating optimization.