Qbasis Manual

Zhentao Wang

August 30, 2017

Contents

Chapter 1

Introduction

Chapter 2

Model representation

2.1 Matrix represenataion of local Hilbert Space

Fermi-Hubbard Model

$$\mathcal{H} = -t \sum_{\langle ij \rangle} \sum_{\sigma} (c^{\dagger}_{i\sigma} c_{j\sigma} + h.c.) + U \sum_{i} n_{i\uparrow} n_{i\downarrow}. \tag{2.1.1}$$

Local Hilbert space is 4-dimensional:

$$\{|0\rangle, |\uparrow\rangle, |\downarrow\rangle, |\uparrow\downarrow\rangle\}. \tag{2.1.2}$$

In this basis:

$$c_{\uparrow} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.1.3a}$$

$$c_{\downarrow} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.1.3b}$$

From these two opeators as input, the code is able to automatically derive the following operators:

$$c^{\dagger}_{\uparrow} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{2.1.4a}$$

$$c^{\dagger}_{\downarrow} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \tag{2.1.4b}$$

$$n_\uparrow = \begin{pmatrix} 0 & & & \\ & 1 & & \\ & & 0 & \\ & & & 1 \end{pmatrix}, \tag{2.1.4c}$$

$$n_\downarrow = \begin{pmatrix} 0 & & & \\ & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}. \tag{2.1.4d}$$

t-J Model

$$\mathcal{H} = -t \sum_{\langle ij \rangle} \sum_{\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + h.c.) + J \sum_{\langle ij \rangle} \left[\frac{S^+_i S^-_j + S^-_i S^+_j}{2} + S^z_i S^z_j - \frac{1}{4} n_i n_j \right], \tag{2.1.5}$$

where

$$S^+_i = c^\dagger_{i\uparrow} c_{i\downarrow}, \tag{2.1.6a}$$

$$S^-_i = c^\dagger_{i\downarrow} c_{i\uparrow}, \tag{2.1.6b}$$

$$S^z_i = \frac{1}{2} \left(c^\dagger_{i\uparrow} c_{i\uparrow} - c^\dagger_{i\downarrow} c_{i\downarrow} \right). \tag{2.1.6c}$$

Local Hilbert space is 3-dimensional:

$$\{|0\rangle, |\uparrow\rangle, |\downarrow\rangle\}. \tag{2.1.7}$$

In this basis:

$$c_\uparrow = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{2.1.8a}$$

$$c_\downarrow = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{2.1.8b}$$

and all other derived operators can be derived automatically by the code.

Spinless Fermion

$$\mathcal{H} = -t \sum_{\langle ij \rangle} (c^\dagger_i c_j + h.c.) + V_1 \sum_{\langle ij \rangle} n_i n_j. \tag{2.1.9}$$

Local Hilbert space is 2-dimensional:

$$\{|0\rangle, |1\rangle\}. \tag{2.1.10}$$

In this basis:

$$c = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{2.1.11}$$

Bose-Hubbard Model

$$\mathcal{H} = -t \sum_{\langle ij \rangle} (b_i^\dagger b_j + h.c.) + \frac{U}{2} \sum_i n_i (n_i - 1) \tag{2.1.12}$$

Local Hilbert space restricted to at most N_{max} bosons:

$$\{|0\rangle, |1\rangle, \ldots, |N_{max}\rangle\}. \tag{2.1.13}$$

In this basis:

$$b = \begin{pmatrix} 0 & \sqrt{1} & 0 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \sqrt{N_{max}} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.1.14}$$

and other operators should be automatically derived.

2.2 Lin table

Chapter 3

Finite size cluster

Chapter 4

Symmetry

4.1 Translation symmetry

4.2 Weisse table

Chapter 5

Eigenvalue problem

5.1 Lanczos algorithm

5.1.1 Description

The m-step Lanczos algorithm performs a factorization in the Krylov subspace:

$$HV = VT + b_m \boldsymbol{v}_m \boldsymbol{e}_m^T, \tag{5.1.1}$$

where H is the $N \times N$ matrix of our problem, $V = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1})$ is $N \times m$ matrix formed by the basis in the Krylov subspace, T is the tridiagonal Hessenberg matrix

$$T = \begin{pmatrix} a_0 & b_1 & & \\ b_1 & a_1 & \ddots & \\ & \ddots & \ddots & b_{m-1} \\ & & b_{m-1} & a_{m-1} \end{pmatrix}, \tag{5.1.2}$$

and $\boldsymbol{e}_m^T = (0, 0, \ldots, 0, 1)$ has only one non-zero element.

Note that $\boldsymbol{v}_i$ are orthonormal, i.e.

$$V^\dagger V = 1. \tag{5.1.3}$$

We can diagonalize the Hessenberg matrix T, giving eigen-pair $(\theta_i,\ \boldsymbol{s}_i)$ satisfying

$$T\boldsymbol{s}_i = \boldsymbol{s}_i \theta_i, \tag{5.1.4}$$

With such information, we can form the Ritz pair $(\theta_i,\ \boldsymbol{y}_i)$ which is a good approximation of the eigenvalues and eigenvectors of the original problem H, where

$$\boldsymbol{y}_i \equiv V\boldsymbol{s}_i. \tag{5.1.5}$$

To estimate the error of the eigenvalues, let's first roughly assume $\boldsymbol{y}_i$ is indeed the true eigenvector with eigenvalue E_i, such that $H\boldsymbol{y}_i = E_i\boldsymbol{y}_i$. Then $\|H\boldsymbol{y}_i - \theta_i\boldsymbol{y}_i\| =$

$|E_i - \theta_i|$,which shows that $\|H\boldsymbol{y}_i - \theta_i \boldsymbol{y}_i\|$ is a good estimate of the error of eigenvalue E_i. Now we relax the assumption about y_i, and calculate $\|H\boldsymbol{y}_i - \theta_i \boldsymbol{y}_i\|$ directly:

$$\begin{aligned} \|H\boldsymbol{y}_i - \theta_i \boldsymbol{y}_i\| &= \|HV\boldsymbol{s}_i - V\boldsymbol{s}_i\theta_i\| = \|HV\boldsymbol{s}_i - VT\boldsymbol{s}_i\| \\ &= \|(HV - VT)\,\boldsymbol{s}_i\| = \left\|b_m \boldsymbol{v}_m \boldsymbol{e}_m^T \boldsymbol{s}_i\right\| \\ &= b_m \left|\boldsymbol{e}_m^T \boldsymbol{s}_i\right|, \end{aligned} \tag{5.1.6}$$

which shows that we can use $b_m \left|\boldsymbol{e}_m^T \boldsymbol{s}_i\right|$ as the error estimation for the Lanczos procedure.

5.1.2 Implementation

The triditional 3-vector version:

Algorithm 5.1 Simple Lanczos algorithm, 3-vector vesion

Require: Space for 3 vectors in RAM: $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2$
Require: An array of pointers $\{\tilde{\boldsymbol{v}}_j \to \boldsymbol{v}_{j \bmod 3}\}$
Require: $\boldsymbol{v}_0$ normalized, $\boldsymbol{v}_1 = \boldsymbol{v}_2 = 0$, $b_0 = 0$.
1: **for** $j = 0, 1, \ldots, m$ **do**
2: $\tilde{\boldsymbol{v}}_{j+1} = -b_j \tilde{\boldsymbol{v}}_{j-1}$ (copy from $\tilde{\boldsymbol{v}}_{j-1}$)
3: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1} + H\tilde{\boldsymbol{v}}_j$
4: $a_j = (\tilde{\boldsymbol{v}}_{j+1}, \tilde{\boldsymbol{v}}_j)$
5: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1} - a_j \tilde{\boldsymbol{v}}_j$
6: $b_{j+1} = \|\tilde{\boldsymbol{v}}_{j+1}\|$
7: calculate $\{\theta_i, s_i\}$
8: for the smallest θ_i, calcualte $b_{j+1}|\boldsymbol{e}_{j+1}^T \boldsymbol{s}_i|$. Stop if small enough
9: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1}/b_{j+1}$
10: **end for**

To save memory, we can actually use just 2 vectors:

Algorithm 5.2 Simple Lanczos algorithm, 2-vector vesion

Require: Space for 2 vectors in RAM: $\boldsymbol{v}_0, \boldsymbol{v}_1$
Require: An array of pointers $\{\tilde{\boldsymbol{v}}_j \to \boldsymbol{v}_{j \bmod 2}\}$
Require: $\boldsymbol{v}_0$ normalized, $\boldsymbol{v}_1 = 0$, $b_0 = 0$.
1: **for** $j = 0, 1, \ldots, m$ **do**
2: $\tilde{\boldsymbol{v}}_{j+1} = -b_j \tilde{\boldsymbol{v}}_{j-1}$ (rescale itself, since $\tilde{\boldsymbol{v}}_{j+1}$ and $\tilde{\boldsymbol{v}}_{j-1}$ overlap in RAM)
3: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1} + H\tilde{\boldsymbol{v}}_j$
4: $a_j = (\tilde{\boldsymbol{v}}_{j+1}, \tilde{\boldsymbol{v}}_j)$
5: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1} - a_j \tilde{\boldsymbol{v}}_j$
6: $b_{j+1} = \|\tilde{\boldsymbol{v}}_{j+1}\|$
7: calculate $\{\theta_i, s_i\}$
8: for the smallest θ_i, calcualte $b_{j+1}|\boldsymbol{e}_{j+1}^T \boldsymbol{s}_i|$. Stop if small enough
9: $\tilde{\boldsymbol{v}}_{j+1} = \tilde{\boldsymbol{v}}_{j+1}/b_{j+1}$
10: **end for**

Chapter 6

Code examples

6.1 Exact diagonalization