Density Matrix Renormalization Group

李梓瑞

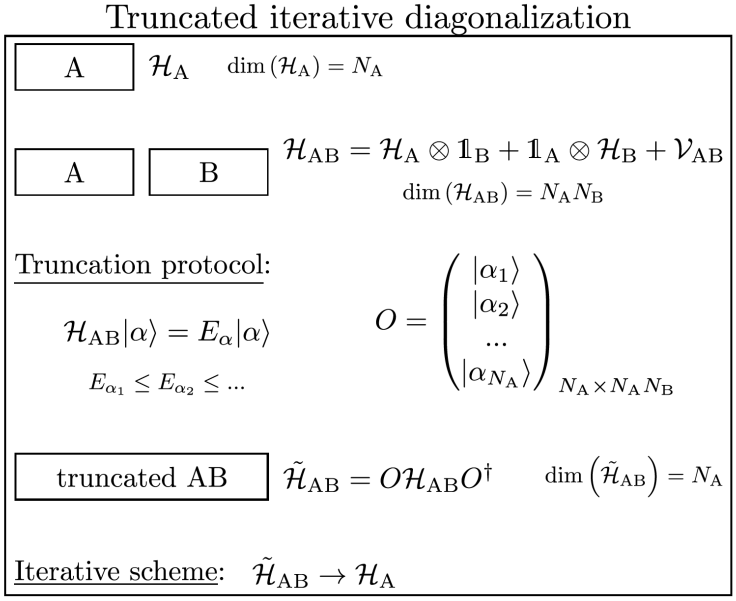
2025 年 4 月 11 日

1 Introduction

由 Wilson 提出的数值重整化群 NRG 启发, 密度矩阵重整化群 DMRG 被提出用以求解量子系统的静态性质. 目前 1D 量子大系统的基态能很好地通过 DMRG 求解, 而更高维的 DMRG 还有很多问题待解决. DMRG 与 RSRG 的思想一脉相承, 都是通过对体系的粗粒化实现求解, 只不过一个是对密度矩阵, 一个是对自旋. DMRG 可以更现代地用张量网络 tensor network 的语言表示, 本文中也会简要介绍张量网络包括其中一部分数值方法. 最后给出实现的 python 代码.

2 Original formulation

2.1 TID



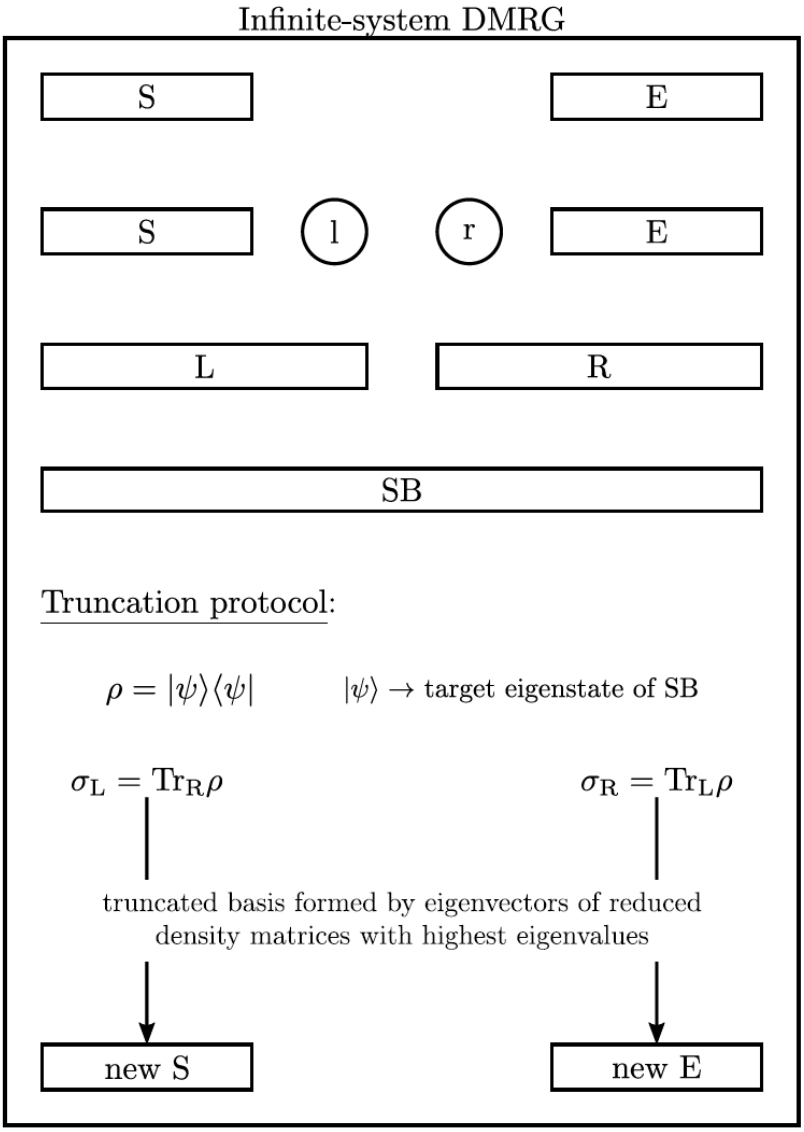
从历史的发展出发, 截断迭代对角化 Truncated iterative diagonalization 更早被提出. 这种方法通过 cut 掉一些高能态来降低系统希尔伯特空间的维数, 进而降低求解哈密顿量谱的 time cost. 具体操作如图, 我们先选取一小块系统, 然后添加环境块, 这样系统的哈密顿量中包含 $\mathcal{H}_A, \mathcal{H}_B$ 和相互作用项. 在系统加环境的子空间内对角化哈密顿量, 得到本征态和本征值. 如果此时将原本 $N_A N_B$ 个本征态 (包含简并态) cut 到只剩 N_A 个低能态, 那么我们就可以保持投影后哈密顿量的内存不变

$$\tilde{\mathcal{H}}_{AB} = O\mathcal{H}_{AB}O^\dagger, \quad \dim\tilde{\mathcal{H}}_{AB} = \dim\mathcal{H}_A \tag{1}$$

经过 N 步操作后, 我们就得到了近似后哈密顿量的谱, 即系统的低能近似谱. 通过 $\{O_i\}$, 可以还原回原本的基. 尽管 TID 方案很直观, 但其缺陷也很明显, 我们只能在低能高能分离相当明显的体系中使用它, 其甚至不能求出势阱中单粒子的基态.

2.2 iDMRG

本节讨论无限系统 DMRG(infinite-system DMRG). 从 TID 得到启发, 我们可以把整个体系分为系统块 S 和环境块 E, 通过迭代更新哈密顿量并每次截断一部分高能态来实现近似.



iDMRG 的方案如图, 定义系统 S 和环境 E, 分别在两边加入 1 个 physical site, 形成 L 块和 R 块. 此时若将 L 块和 R 块合并 (考虑相互作用), 那么形成的 Super Block 就是描述整个体系波函数的块. 对于每一步扩充, 我们之后都做一步截断, 大体就是提取体系的波函数形成密度矩阵并把它 trace 到 reduced 密度矩阵 (L/R), 取其贡献最大的态截断 L 块和 R 块形成新的 S 块和 E 块.

更详细地解释截断方案, 考虑超块的波函数 (已归一)

$$|\psi\rangle = \sum_{i_L=1}^{N_L}\sum_{i_R=1}^{N_R}\psi_{i_L,i_R}\,|i_L\rangle \otimes |i_R\rangle \tag{2}$$

变分波函数

$$|\tilde{\psi}\rangle = \sum_{\alpha_L=1}^{D_L}\sum_{i_R=1}^{N_R}c_{\alpha_L,i_R}\,|\alpha_L\rangle \otimes |i_R\rangle \tag{3}$$

其中 $|\alpha_L\rangle$ 表示截断后 L 块的基. 对于固定的 D_L, 为了得到最佳近似方案, 考虑最小化

$$\| |\psi\rangle - |\tilde{\psi}\rangle \|^2 = 1 - \sum_{i_L,i_R,\alpha_L} \left(\psi^*_{i_L,i_R} c_{\alpha_L,i_R} \langle i_L|\alpha_L\rangle + c^*_{\alpha_L,i_R} \langle \alpha_L|i_L\rangle\right) \tag{4}$$

$$+ \sum_{\alpha_L,i_R} |c_{\alpha_L,i_R}|^2 \tag{5}$$

取其对 c 的导数并令其为 0, 得到

$$c_{\alpha_L,i_R} = \sum_{i_L=1}^{N_L} \psi_{i_L,i_R} \langle \alpha_L|i_L\rangle \tag{6}$$

$$\| |\psi\rangle - |\tilde{\psi}\rangle \|^2 = 1 - \sum_{\alpha_L=1}^{D_L} \langle \alpha_L|\sigma_L|\alpha_L\rangle \tag{7}$$

where

$$\sigma_L = \mathrm{Tr}_R \rho = \sum_{i_R=1}^{N_R} \langle i_R|\rho|i_R\rangle, \quad \rho = |\psi\rangle\langle\psi| \tag{8}$$

于是为了最小化 $\| |\psi\rangle - |\tilde{\psi}\rangle \|^2$, 需要使 σ_L 对截断基的部分迹最大. 由 Schur-Horn theorem 可知, 此时 $|\alpha_L\rangle$ 应取约化密度矩阵的本征态

$$\sigma_L |\alpha_L\rangle = \lambda_{\alpha_L} |\alpha_L\rangle \tag{9}$$

and

$$\| |\psi\rangle - |\tilde{\psi}\rangle \|^2 = 1 - \sum_{\alpha_L=1}^{D_L} \lambda_{\alpha_L} \tag{10}$$

至此我们已经截断了 L 块, 对于 R 块截断只需要执行完全相同的操作.

下面讨论上述方案的有效性. 为了知道截断数量 D_L 取多少合适, 考察可测量的纠缠熵 (vN)

$$S = S(\sigma_{L/R}) = -\mathrm{Tr}\left(\sigma_{L/R} \ln \sigma_{L/R}\right) \tag{11}$$

focus on L 块

$$S = -\sum_{\alpha_L=1}^{N_L} \lambda_{\alpha_L} \ln \lambda_{\alpha_L} \simeq -\sum_{\alpha_L=1}^{D_L} \lambda_{\alpha_L} \ln \lambda_{\alpha_L} \tag{12}$$

此时如果本征值取 $\lambda_{\alpha_L<D_L} = 1/D_L$, $\lambda_{\alpha_L>D_L} = 0$, 熵最大

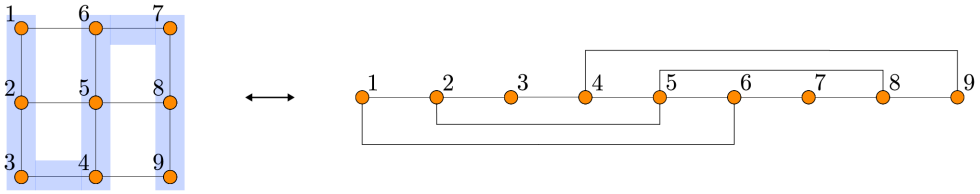
$$S \le \ln(D_L) \Rightarrow D_L \ge e^S \tag{13}$$

于是对于一个良好的截断, 我们可以估计 $D_L \sim e^S$. 对于不同的体系 (local)

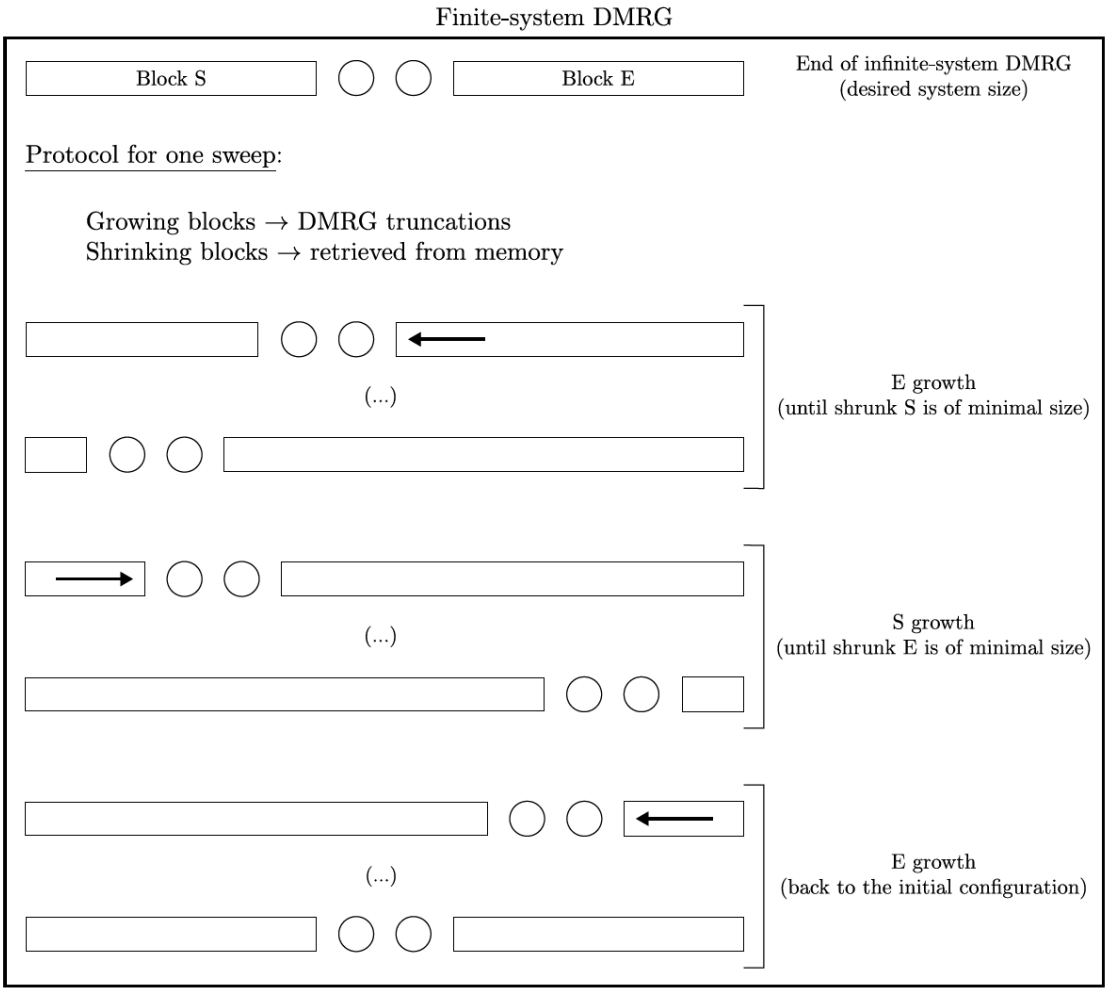
- 1D gapped, $S \sim const$, $D \sim e^{const}$

- 1D gapless, $S \sim c \ln L$, $D \sim L$
- 2,3D gapped, $S \sim L^{d-1}$, $D \sim \exp\left(L^{d-1}\right)$

1D 体系 DMRG 都能很好的模拟, 而高维需要指数增长的计算资源, 这实际上无法直接实现. 此外, 对于有阻挫的体系或自旋液体这种基态高度简并的体系,DMRG 难以求解. 对于长程相互作用,DMRG 也难以模拟.



2.3 DMRG



在 iDMRG 中仍然存在一些问题, 比如缺乏热化过程而收敛到亚稳态. 本节介绍有限系统的 DMRG(finite-system DMRG). 作为 iDMRG 的改进方案, 其初始波函数可以取 iDMRG 的结果. 之后, 使其中一个块增长而另一个块缩小, 每做一步移动就截断 (方法和上节相同) 增长的块一次, 增长到极限时翻转, 这么扫描两次后还原到原来的状态, 称为一次 sweep. 进行多次 sweep 后,iDMRG 得到的结果能被明显改善.

值得一提,DMRG 可以采用固定 D_L 截断, 也可以使用固定误差 ϵ 截断, 通常后者用的更多.

3 TN

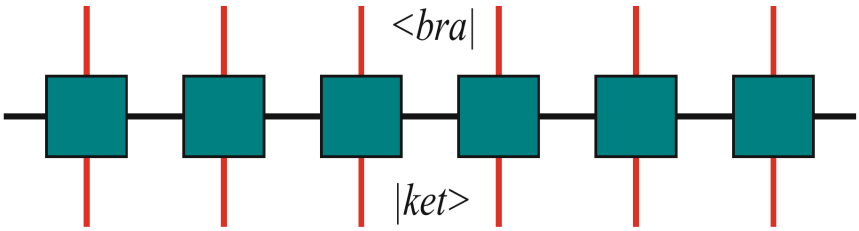
3.1 Basis

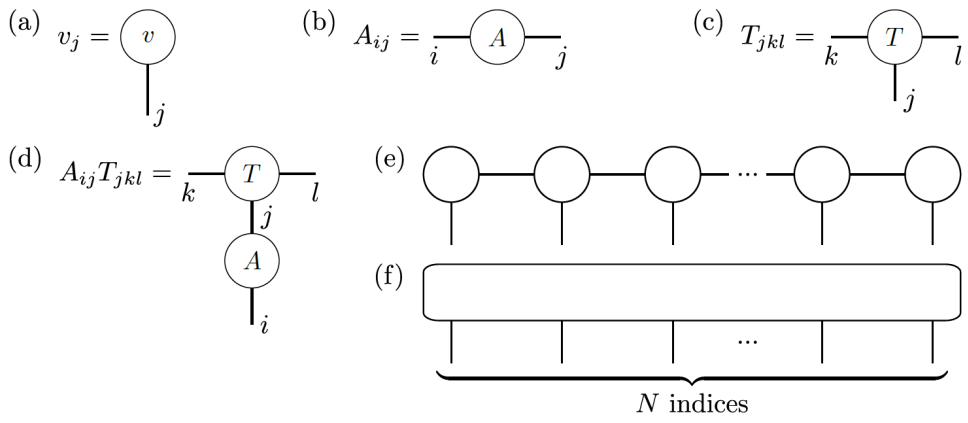
张量网络 tensor network 的核心就是将量子态和算符表示为张量, 利用高维张量 (如 np.array) 的算法 (表示为张量图) 计算体系的性质. 实际上,TN 比他的名字更直观. 更详细的介绍见 [2].

不同于微分几何的张量, 这里介绍的张量可以简单理解为具有多个指标 index 的数学对象, 比如标量是 0 阶张量, 矢量是 1 阶张量, 矩阵可以视为 2 阶张量 (这里有一些不太重要的微妙区别). 形式上, 张量 T 具有 r 个指标, 每个指标可以取 $d_i, i = 1, 2, \ldots, r$ 种取值 (如 1,2,3,a,b,c). 我们称该张量的 (维度) 阶数 rank 为 r, 其有 r 个 leg, 每个 leg 有 d_i 个取值. 对于高阶张量,reshape 是一个常用的方法, 比如将指标 $(\alpha, \beta) \equiv \gamma$, 于是新的指标取值数就等于原指标取值数之和. 这在张量图上相当于把两个 leg 写成一个 leg

对于单个 site, 量子态有一个 leg(1 physical index), 算符有两个 leg(2 physical indices), 例如自旋 1/2 系统

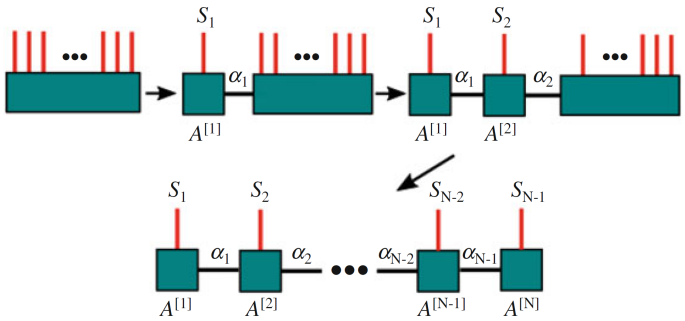
$$|\psi\rangle = \sum_{s_1 \ldots s_N=0}^{1} C_{s1 \ldots s_N} |s_1\rangle \ldots |s_N\rangle, \quad \hat{O} = \sum_{s,a} \prod_n W_{s_n s'_n, a_n a_{n+1}} |s'_n\rangle \langle s_n| \tag{14}$$



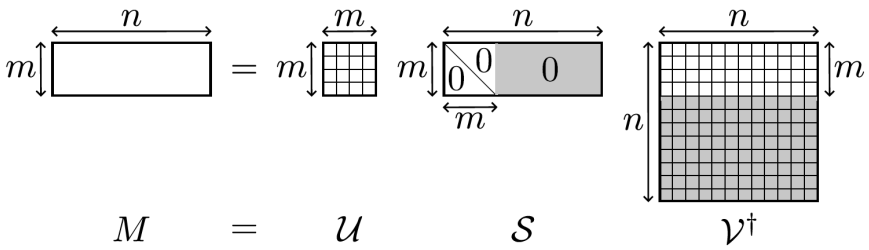


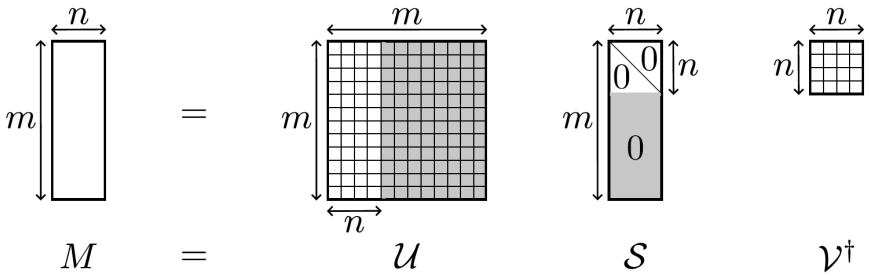
如果对系数做 SVD 分解 (其他分解方法如 QR 也可以), 那么可以得到矩阵乘积态 MPS

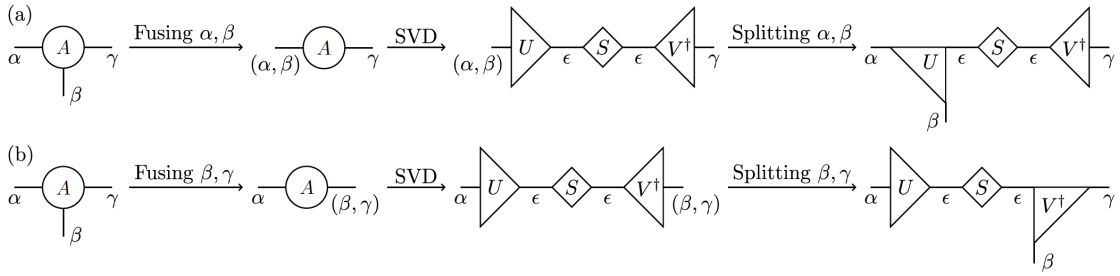
$$C_{ss'} = \sum_{a,a'} U_{sa}\lambda_{aa'}V^*_{a's'}, \quad C_{s_1\ldots s_{N-1}s_N} = \sum_{a_1\ldots a_{N-1}} A^{[1]}_{s_1,a_1}A^{[2]}_{s_2,a_1a_2}\ldots A^{[N]}_{s_N,a_{N-1}} \tag{15}$$



其中 λ 是矩阵奇异值的谱, 包含 χ 个奇异值 (通常按降序存储),χ 称为矩阵的 rank, $\{a_i\}$ 称为 geometrical or virtual indices.







截断误差 (χ' 阶近似, 即截断 χ' 个奇异值) 为

$$\epsilon = \sqrt{\sum_{a=\chi'}^{\chi-1} \lambda_a^2} \tag{16}$$

于是量子态就可以表示为

$$|\psi\rangle = \mathrm{tTr} A^{[1]\dots A^{[N]}} |s_1 \dots s_N\rangle = \mathrm{tTr} \prod_{n=1}^{N} A^{[n]} |s_n\rangle \tag{17}$$

其中 tTr 表示对所有共同指标求和. 同样的,2D 中也可以定义类似于 MPS 的结构, 比如 TTNS 和 PEPS, 这里不展开.

对于算符, 我们可以定义矩阵乘积算符 MPO, 一般将单个 MPO 构造为上三角或下三角形式

$$W^{[n]}_{s_n s'_n} = \begin{pmatrix} C^{[n]} & 0 \\ B^{[n]} & A^{[n]} \end{pmatrix}, \quad O = \sum_{n=1}^{N} \left(\bigotimes_{i=1}^{n-1} A^{[i]} \right) \otimes B^{[n]} \otimes \left(\bigotimes_{j=n+1}^{N} C^{[j]} \right) \tag{18}$$

例如

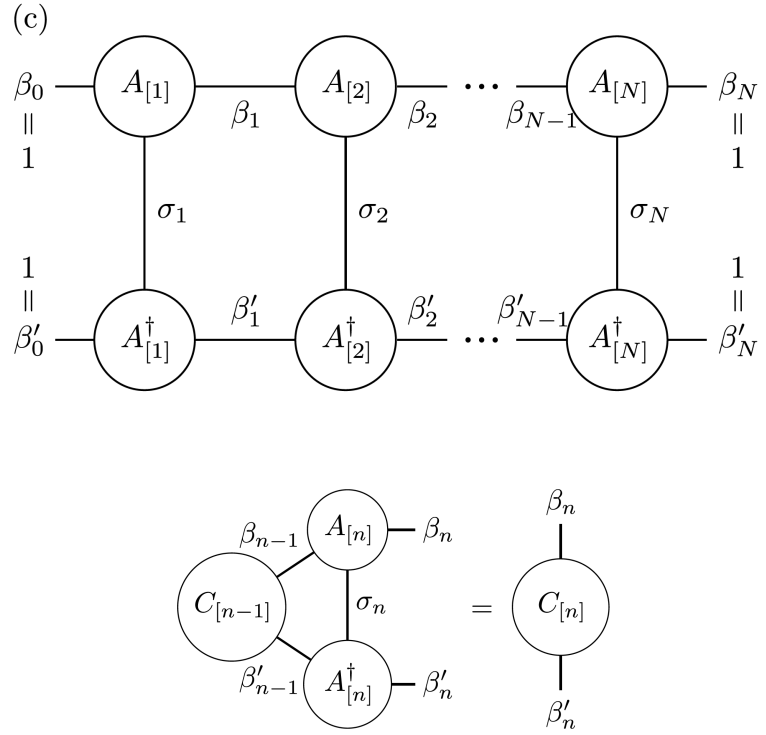
$$H = J \sum_n S^z_n S^z_{n+1} + h \sum_m S^x_m \Rightarrow W^{[n]} = \begin{pmatrix} I & 0 & 0 \\ S^z & 0 & 0 \\ hS^x & JS^z & I \end{pmatrix} \tag{19}$$

$$n = \sum_{m,n=1}^{N} e^{i(m-n)k} b^\dagger_m b_n \Rightarrow W_n = \begin{pmatrix} I & 0 & 0 & 0 \\ b^\dagger & e^{ik} I & 0 & 0 \\ b & 0 & e^{-ik} I & 0 \\ b^\dagger b & e^{ik} b^\dagger & e^{-ik} b & I \end{pmatrix} \tag{20}$$

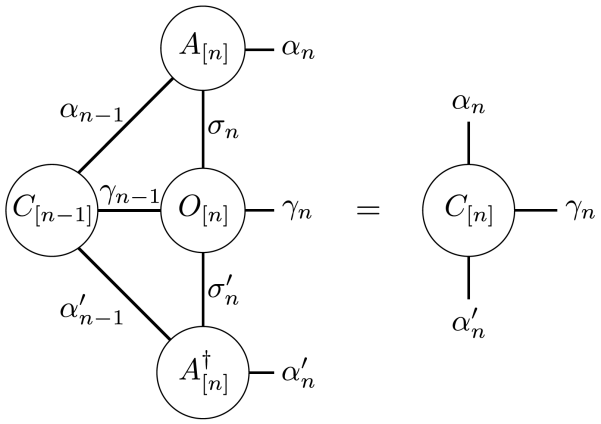
计算细节可以参考 [1].

下面介绍两种算法, 分别针对态的 overlap 和算符平均. 如果要计算 $\langle\psi|\psi\rangle$, 我们首先可以想到一个 naive 的方法: 直接把所有的键收缩, 然而这种方法的复杂度是 $O(d^N)$, 其中 d 是 physical index 的维度,D 是 virtual index 的维度,N 是 site 的数量. 这种指数复杂度的算法显

然不是我们所期望的, 幸运的是确实有一种收缩方法给出线性依赖的复杂度, 即想拉拉链一样挨个 site 收缩, 如图.



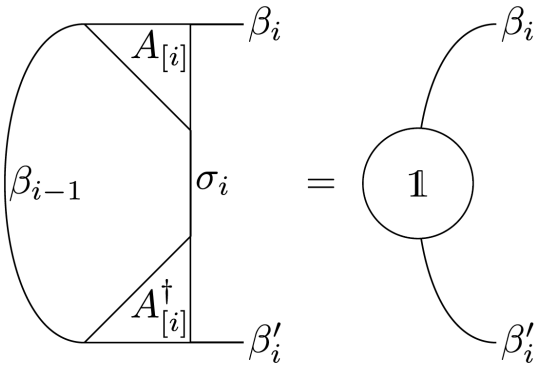
对于第 n 步, 最优策略是: 先将 $C_{[n-1]}, A_{[n]}$ 做收缩, 再和 $A^\dagger_{[n]}$ 收缩. 第一步和最后一步通过添加 trivial 的指标来开始和结束, 即 $C_{[0]} = I$. 这个算法的复杂度 $\sim O(ND^3d)$. 类似的, 如果在两个 state 中间加一个 local 的算符, 等同于计算 $\langle\psi|O|\psi\rangle$, 我们也可以使用这种拉拉链的策略.



对于第 n 步, 最优策略是: 先将 $C_{[n-1]}, A_{[n]}$ 做收缩, 再依次和 $O_{[n]}, A^\dagger_{[n]}$ 收缩. 这个算法的复杂度 $\sim O(ND^3wd)$, 其中 w 是 O 的 virtual index 的维度.

另外, 如果采用正则形式的 MPS, 上述操作会大大简化, 相当于做了 normalization $\langle\psi|\psi\rangle = 1$. 左正则形式如图, 表达式为

$$\sum_{\beta_{i-1},\sigma_i} A^*_{\beta'_i,\sigma_i,\beta_{i-1}} A_{\beta_{i-1},\sigma_i,\beta_i} = \delta_{\beta'_i,\beta_i} \tag{21}$$



右正则形式同理, 表达式为

$$\sum_{\beta_i,\sigma_i} A_{\beta_{i-1},\sigma_i,\beta_i} A^*_{\beta_i,\sigma_i,\beta'_{i-1}} = \delta_{\beta_{i-1},\beta'_{i-1}} \tag{22}$$

site-canonical MPS 定义为同时满足左正则和右正则的 MPS. 可以通过 SVD 分解构造正则形式.

3.2 DMRG in TN

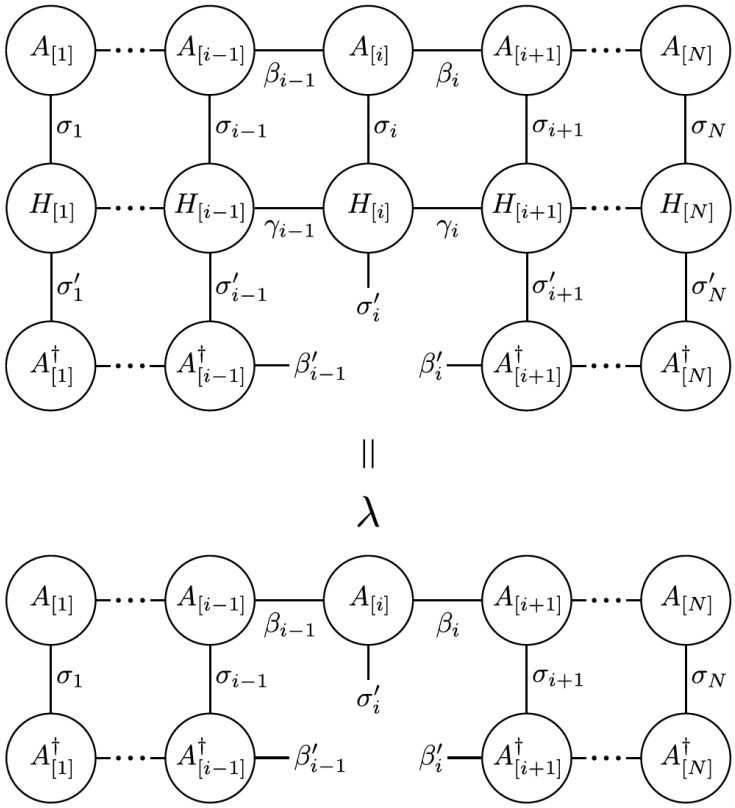
下面就可以在张量网络的框架下实现 DMRG 了. 对于任意一个 1D 量子系统, 我们要求解基态就相当于最小化 cost function with normaliztion

$$F = \langle\psi|\mathcal{H}|\psi\rangle - \lambda\langle\psi|\psi\rangle \tag{23}$$

其就是拉格朗日乘子法的应用, 求解这个问题只需要

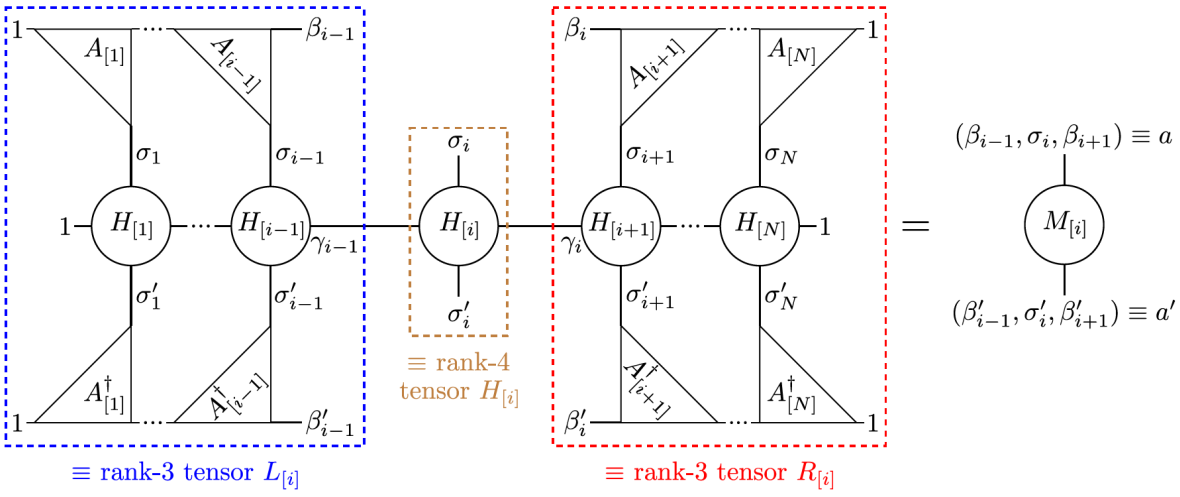
$$\frac{\partial}{\partial A^\dagger_{[i]}}\left(\langle\psi|\mathcal{H}|\psi\rangle - \lambda\langle\psi|\psi\rangle\right) = 0, \quad \forall i \tag{24}$$

在张量图表示中, 就求导相当于把 $A^\dagger_{[i]}$ 扣掉



令每个 $A_{[i]}$ 和 $A^\dagger_{[i]}$ 都是 site-canonical 的, 等价于特征值方程

$$\sum_a M^{a',a}_{[i]} A^a_{[i]} = A^{a'}_{[i]}, \quad a \equiv (\beta_{i-1}, \sigma_i, \beta_{i+1}) \tag{25}$$



之后, 对于每个 site,(对于求解基态) 取具有最小本征值的本征态, 无论通过直接对角化还是 Lanczos 算法实现. 之后用这个本征态更新 $A_{[i]}$. 于是整个流程就是从一个随机的

MPS 开始做 sweep, 直到收敛到基态. 采用我们前面提到的收缩策略, 整个 DMRG 的复杂度 $\sim O(ND^3)$(采用矩阵形式复杂度 $\sim O(ND^4)$). 需要注意, 每次对 site 求解本征值之后都需要通过 SVD 分解或其他分解重新保证 MPS 是 site-canonical 的.

DMRG 求解激发态也是可以的. 比如我们要求第一激发态, 只需要改造 cost function 为

$$F = \langle\psi|\mathcal{H}|\psi\rangle - \lambda\langle\psi|\psi\rangle - \lambda_0\langle GS|\psi\rangle \tag{26}$$

其中 $|GS\rangle$ 就是我们前面求解的基态波函数. 实质上就是通过在 cost function 加入惩罚项来保证 $\langle GS|\psi\rangle = 0$, 即在与 $|GS\rangle$ 正交的子空间中做 DMRG. 求解激发态的具体实践可以参考 [4].

尽管 TN 中 DMRG 和原始的 DMRG 能很明显地看出相似点, 这里还是稍微讨论一下这两者的相同和不同. 在 MPS 表述中, 进行的 SVD 分解和截断就相当于原来 DMRG 中将增长的块对应的 reduced 密度矩阵截断 (投影到子空间), SVD 中保留的奇异值 (MPS 的键数 D) 就对应于密度矩阵保留的本征态, 其本征值就是对应奇异值的平方. 然而基于 MPS 的 DMRG 有一个优点: 过程中不需要截断哈密顿量矩阵,$\mathcal{H}$ 的 MPO 表述总是精确的, 反观原始版本就需要不断截断哈密顿量.

上述操作都是 DMRG 的 1 site 更新版本, 对于 2 site 版本只需要将需要求解的中间的单个 MPS 换成 2 site 的 MPS 就可以了. 事实上 2 site 更新的 DMRG 比单 site 版本能遍历更多态, 因此一般前几次 sweep 先用 2 site 版本避免亚稳态, 再换成 1 site 版本做 sweep 直到收敛 (也可以采用量子涨落修正过的 1 site 版本 DMRG, 理论上计算成本更低 [5]). 此外, 有更复杂的 DMRG 切空间方法 [6][7], 针对动力学问题和有限温也有对应的 DMRG 版本, 也可以利用对称性加速计算, 这里不展开.

4 Other methods

顺便简单介绍一下其他和 TN 有关的方法.

4.1 ED

精确对角化 (exact diagonalization) 是思路最直接的求解量子系统 (静态和动力学) 的方法, 实际上就是把哈密顿量矩阵对角化然后提取本征值和本征态. 常用的程序包是 python 的 quspin 或 qutip, 下面介绍的大多数 ED 方法都集成进去了.

如此简单直接的方法就能求解. 那么, 代价是什么呢? 考虑解一个 16 个格点上的量子体系, 相当于对角化一个 $2^{16} \times 2^{16}$ 大小的矩阵, 如果只做一些普通的优化这样的计算量在小型计算机上勉强可以跑动, 但是我们一般期望计算热力学极限下的性质. 为了观察到更明显的行为或做更强的 argument, 我们需要计算更多的格点, 但是复杂度随着系统大小指数增长, 我们很快会遇到指数墙 (exponential wall). 事实上, 正是为了解决这个问题, 各种方法相继被提出, 如 DMRG,TN,QMC,DMFT 等等.

另外, 在不改变 ED 基本思想的前提下, 我们也可以利用对称性和 Lanczos 算法降低 cost. 对称性大概就是先把矩阵块对角化, 然后针对某个具有对称性的块对角化, 最常见的就是取中间最大的那个块 (即半满态), 更进一步可以取不同动量、不同宇称的块. 详细的介绍见 [8]. 下面介绍 Lanzcos 算法.

在希尔伯特空间中, 我们可以选择一个初始态 $|\psi\rangle$ 来构造由基 $|\psi\rangle, H|\psi\rangle, \ldots, H^{N-1}|\psi\rangle$ 张成的 Krylov 子空间, 我们的目的就是通过幺正变换把原始的哈密顿量矩阵分解到维度更小更容易计算的 Krylov 子空间对应的矩阵 (三对角阵) $H_{M\times M} = P_{M\times N} T_{N\times N} P^{\dagger}_{N\times M}$. 投影算符满足 $P^{\dagger}_{N\times M} P_{M\times N} = I_{N\times N}, P_{M\times N} P^{\dagger}_{N\times M} = I_{N\times N} \otimes 0_{M-N\times M-N}$. 具体迭代操作为

$$|\psi'\rangle = H|\psi_n\rangle - \beta_n |\psi_{n-1}\rangle \tag{27}$$

$$\alpha_n = \langle\psi_n|\psi'\rangle \tag{28}$$

$$|\psi''\rangle = |\psi'\rangle - \alpha_n |\psi_n\rangle \tag{29}$$

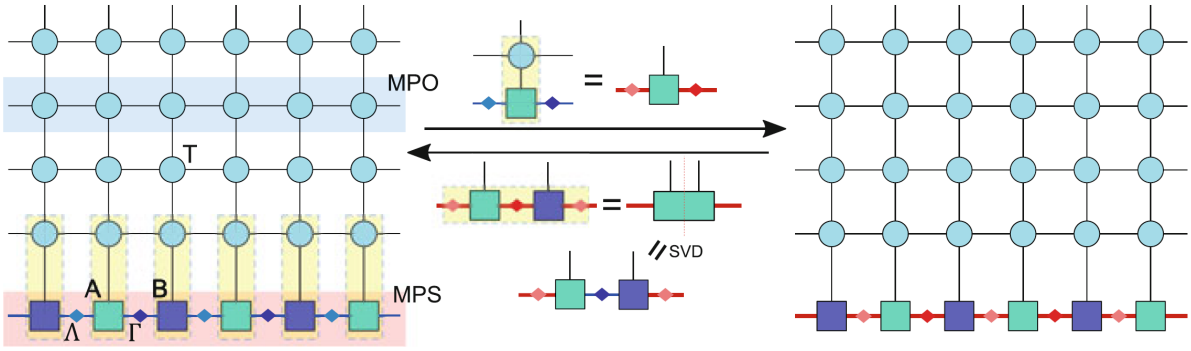
$$\beta_{n+1} = \sqrt{\langle\psi''|\,|\psi''\rangle\rangle} \tag{30}$$

$$|\psi_{n+1}\rangle = |\psi''\rangle / \beta_{n+1} \tag{31}$$

其中 $|\psi_{n-1}\rangle = 0, ||\,|\psi_0\rangle\,|| = 1$. 这一系列操作可以构造一组正交归一基, α, β 就是三对角阵的元素. 这种方法对谱中最大本征值和最小本征值的收敛速度非常快, 因此常用来捕捉基态或能带结构的性质. 更详细的讨论见 [9].

4.2 TEBD

TEBD(即 Time-Evolving Block Decimation) 是一种计算时间演化的 TN 边缘态 (boundary-state) 方法, 下面以 Z_2 对称 MPS 的无限系统的 TEBD(iTEBD) 为例.



初态可表示为

$$|\psi_0\rangle = \sum_{\{a\}} \cdots \Lambda_{a_{n-1}} A_{s_{n-1},a_{n-1}a_n} \Gamma_{a_n} B_{s_n,a_n a_{n+1}} \Lambda_{a_{n+1}} \cdots \tag{32}$$

可以看成是做完 SVD 分解之后的 MPS. 每一步 TEBD 都做如下操作: 缩并一层 MPO, 形成新的未截断的 MPS, 其中单层演化算符可由 Trotter-Suzuki 分解得到

$$A_{s,\tilde{a}\tilde{a}'} = \sum_{s'} T_{sbs'b'} A_{s',aa'}, \quad B_{s,\tilde{a}\tilde{a}'} = \sum_{s'} T_{sbs'b'} B_{s',aa'} \tag{33}$$

and

$$\Lambda_{\tilde{a}} = \Lambda_a \mathbf{1}_b, \quad \Gamma_{\tilde{a}'} = \Gamma_{a'} \mathbf{1}_{b'} \tag{34}$$

where $\tilde{a} \equiv (b, a)$ and $\tilde{a}' \equiv (b', a')$. 缩并完之后再对这层做 SVD 分解并截断 virtual index

$$M_{s_1\tilde{a}_1,s_2\tilde{a}_2} = \sum_{\tilde{a}} \Lambda_{\tilde{a}_1} A_{s_1,\tilde{a}_1\tilde{a}} \Gamma_{\tilde{a}} B_{s_2,\tilde{a}\tilde{a}_2} \Lambda_{\tilde{a}_2} = \sum_{a=0}^{\chi-1} U_{s_1,\tilde{a}_1 a} \Gamma_a V_{s_2,a\tilde{a}_2} \tag{35}$$

$$A_{s_1,\tilde{a}a} = (\Lambda_{\tilde{a}})^{-1} U_{s_1,\tilde{a}a}, \quad B_{s_2,a\tilde{a}} = V_{s_2,a\tilde{a}} (\Lambda_{\tilde{a}})^{-1} \tag{36}$$

这样就完成了一次演化. 只要不断地和 MPO 缩并截断就能实现演化. 该方法的误差只由 SVD 的截断产生. 如果 MPO 是幺正的, 那么 MPS 会自动收敛到正则形式; 如果 MPO 不是幺正的, 那么每一步 TEBD 之后还要把 MPS 化成正则形式. 对于最近邻相互作用, 另外一个常用的方案是把哈密顿量分解成一系列短时短程作用的叠加并分为奇次演化组和偶次演化组, 放在 TN 中交叉做演化.

iTEBD 常用于捕捉 1D 量子系统在热力学极限下的特征. 此外还有虚时 TEBD 通过长时间演化抑制激发态, 从而计算基态和低能激发态. 由于 TEBD 需要 Trotter-Suzuki 分解来实现, 因此对于 nonlocal 的哈密顿量 TEBD 并不能很好地模拟. 更详细的介绍见 [10].

4.3 TDVP

TDVP 是另外一个计算时间演化的方法, 而其比之 TEBD 有几个优点, 比如更精确的变分方法、保持守恒量、可计算 nonlocal interaction, 但其计算效率更低. TDVP 的核心思想就是用 MPS 参数空间 local 的切空间代替真正的流形做近似计算, 然后再做投影到 MPS 流形上. 其中原理和细节的比较复杂, 因此这里只能粗略地介绍, 更细致的介绍见 [11] 以及 [12][7]. 此外我还推荐知乎文章 [13].

理论上 TDVP 可以计算任意变分波函数, 我们只关注 MPS 形式的 TDVP(即对 MPS 张量 $A^{[i]}_{s_i}(t)$ 做变分). 态演化

$$i\frac{\partial}{\partial t}\left|\psi(A)\right\rangle = H\left|\psi(A)\right\rangle \tag{37}$$

其中 MPS 波函数 $|\psi\rangle$ 已参数化. 在切空间上演化并投影

$$i\frac{\partial}{\partial t}\left|\psi(A(t))\right\rangle = P_{A(t)}H\left|\psi(A(t))\right\rangle, \quad \left|\phi(\dot{A};A)\right\rangle \equiv \frac{\partial}{\partial t}\left|\psi(A(t))\right\rangle \tag{38}$$

得到 TDVP 方程

$$\left|\phi(\dot{A};A)\right\rangle = -iP_{A(t)}H\left|\psi(A(t))\right\rangle \tag{39}$$

写成张量形式即为

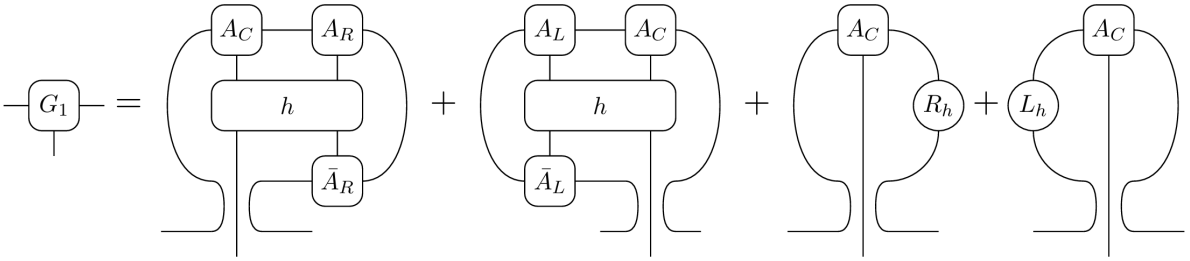
$$\sum_i \dot{A}^{[i]}_{s_i}(t)\partial_{A^{[i]}_{s_i}}\left|\psi(\{A^{[k]}_{s_k}(t)\})\right\rangle = -iP_{A(t)}H\left|\psi(\{A^{[k]}_{s_k}(t)\})\right\rangle \tag{40}$$

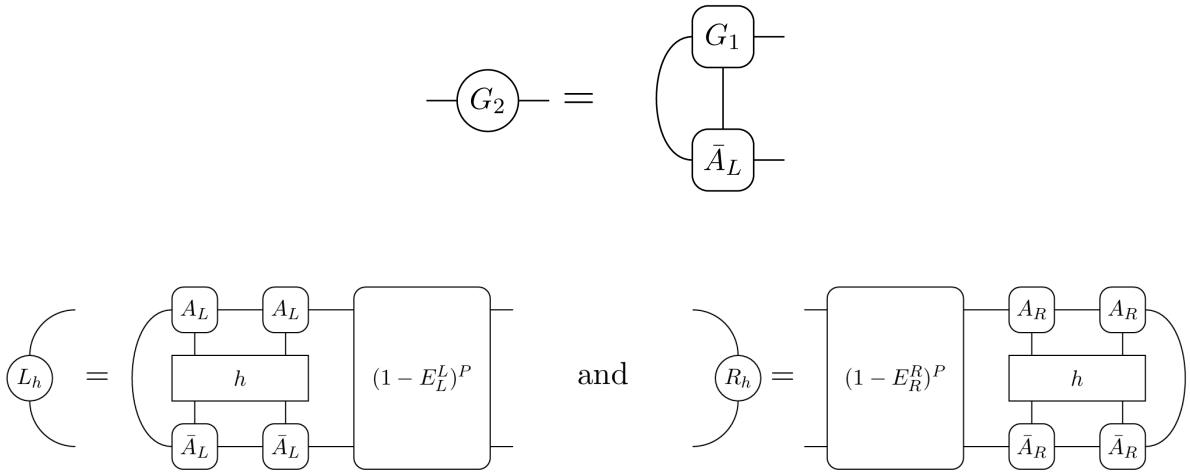
可以看出这种形式和 DMRG 十分相似, 其中 $P_{A(t)}$ 是切空间 $T_{\mathcal{M}}$ 上的投影算符.

简单起见, 我们考虑 site-canonical 形式

$$\frac{\partial}{\partial t}\left|\psi(A_L,A_C,A_R)\right\rangle = -iP_A H\left|\psi(A_L,A_C,A_R)\right\rangle \tag{41}$$

假设从左到右 sweep, 左张量已更新而右张量未更新. 第一步, 更新中心张量 $A^{[n]}_C = e^{-G^{[n]}_1\delta t}A^{[n]}_C$; 第二步, 对中心张量做 QR 分解 $A^{[n]}_C = A^{[n]}_L C^{[n]}$; 第三步, 更新中心矩阵 $C^{[n]} = e^{+iG^{[n]}_2\delta t}C^{[n]}$; 第四步, 将 C 吸收进 A_L, 定义新的中心张量 $A^{[n+1]}_C = C^{[n]}A^{[n+1]}_R$. 其中





至此我们完成了对一个张量的一次演化, 以此类推 (类似于 DMRG) 进行演化. 由于要求 sweep 前后 C 相同, 我们可以提出一个更简单的算法, 即顺时演化 $A_C^{[n]}$ 同时逆时演化 $C^{[n]}$, 再从得到的新 $A_C^{[n]}$ 和 $C^{[n]}$ 中求出 $A_L^{[n-1]}$ 和 $A_R^{[n+1]}$. 另外当然还有其他的 TDVP 版本, 比如单纯将 QR 分解改为 SVD 分解、2-site 版本等等, 这里不展开.

5 Codes

目前主流实现 DMRG 的语言主要有 Python 和 Julia, 前者一般会使用各种方法基于 numpy 包加速计算, 后者本身就专注于科学计算 (就是说一般比 py 快但没 c 快). 这两种语言都有现成的集成 tensor network 的包, 分别是 tenpy 和 itensor [3]. 下文以 tenpy 为例.

5.1 示例代码

只采用 numpy 和 scipy 包就可以写一个简单的示例代码. 计算张量收缩使用 np.tensordot 或 np.einsum, 重构张量形状使用 np.reshape, SVD 分解使用 np.linalg.svd, 计算特征值和特征向量使用 scipy.sparse.linalg.eigsh (已集成 Lanzcos 算法). 示例代码如下

```
1  # finite_DMRG, OBC, 1-site, Ground State
2  import numpy as np
3  from scipy.sparse.linalg import eigsh
4  import matplotlib.pyplot as plt
5
6  # svd分解并截断
```

```
    def cut(chi, psi):
        U, S, V = np.linalg.svd(psi, full_matrices=False)
        U_trunc = U[:, :chi]
        S_trunc = S[:chi]
        V_trunc = V[:chi, :]
        return U_trunc, S_trunc, V_trunc

    # 定义MPS及其操作
    class MPS:

        def __init__(self, mpo):
            self.mpo = mpo
            self.mps = None
            self.N = len(mpo)
            self.S_ent = None

        def random_mps(self, v):
            # 获得随机初态
            self.mps = []
            self.mps.append(np.random.rand(1, self.mpo[0].shape[2], v))
            for i in range(1, self.N - 1):
                self.mps.append(np.random.rand(v, self.mpo[i].shape[2], v))
            self.mps.append(np.random.rand(v, self.mpo[self.N - 1].shape[2],
                1))
            return self.mps

        def left_canonical(self, chi=20):
            # 变为左正则形式
            for i in range(self.N):
                A = self.mps[i] # vL i vR
                l, d, r = A.shape
```

```
37                A = A.reshape((l * d, r)) # (vL i) vR
38                U, S, V = cut(chi, A)
39                self.mps[i] = U.reshape((l, d, U.shape[1])) # vL i vR
40                if i < self.N-1:
41                    SV = np.tensordot(np.diag(S), V, axes=(1, 0)) # vR [vR], [
                          vR*] vR*
42                    self.mps[i + 1] = np.tensordot(SV, self.mps[i + 1], axes
                          =(1, 0)) # vR [vR*], [vL] i vR
43
44        def right_canonical(self, chi=20):
45            # 变为右正则形式
46            for i in reversed(range(self.N)):
47                A = self.mps[i]
48                l, d, r = A.shape
49                A = A.reshape((l, d * r)) # vL (i vR)
50                U, S, V = cut(chi, A)
51                self.mps[i] = V.reshape((V.shape[0], d, r)) # vL i vR
52                if i > 0:
53                    US = np.tensordot(U, np.diag(S), axes=(1, 0)) # vL* [vL*],
                          [vL] vL
54                    self.mps[i - 1] = np.tensordot(self.mps[i - 1], US, axes
                          =(2, 0)) # vL i [vR], [vL*] vL
55
56        def get_entropy(self, chi=20):
57            # 取得半链纠缠熵
58            self.left_canonical()
59            self.right_canonical()
60            mps_reduced = self.mps[:self.N // 2]
61            I = np.ones((1, 1))
62            mps_side = np.ones((1, 1, 1))
63            for i in range(len(mps_reduced)):
```

```
64                mps_side = np.tensordot(mps_side, mps_reduced[i], axes=(2, 0))
                      # vL i [vR], [vL] i vR
65                mps_side = mps_side.reshape((mps_side.shape[0],
66                                             mps_side.shape[1] * mps_side.shape[2],
67                                             mps_side.shape[3]))
68            density_matrix_reduced = np.tensordot(mps_side, mps_side, axes=(2,
                  2)) # vL (i*N//2) [vR], vL (i*N//2) [vR]
69            density_matrix_reduced = np.transpose(density_matrix_reduced, axes
                  =(0, 1, 3, 2)) # vL (i*N//2) (i*N//2) vL
70            density_matrix_reduced = np.tensordot(density_matrix_reduced, I,
                  axes=([0, 3], [0, 1])) # [vL] (i*N//2) (i*N//2) [vL], [v] [v])
71            U, S, V = cut(chi, density_matrix_reduced)
72            S = S[S > 1.e-15]
73            S2 = S * S
74            S2 /= np.sum(S2)
75            self.S_ent = - np.sum(S2 * np.log(S2))
76            return self.S_ent
77
78    # 定义MPO
79    class MPO:
80
81        def __init__(self, N):
82            self.N = N
83            self.mpo = None
84
85        def XY(self):
86            # XY模型
87            # spin-1/2 operators
88            ##  ^+_l
89            sp = np.zeros((2, 2))
90            sp[0, 1] = 1
```

```
91              ##  ^-_l
92              sm = np.zeros((2, 2))
93              sm[1, 0] = 1
94              ## I_l
95              I2 = np.eye(2)
96
97              # MPO Hamiltonian
98              ## H[l]
99              Hl = np.zeros((4, 2, 4, 2))
100             Hl[0, :, 0, :] = I2
101             Hl[1, :, 0, :] = sm
102             Hl[2, :, 0, :] = sp
103             Hl[3, :, 1, :] = -0.5 * sp
104             Hl[3, :, 2, :] = -0.5 * sm
105             Hl[3, :, 3, :] = I2
106             Hl = np.transpose(Hl, (0, 1, 3, 2)) # vL_O i i* vR_O
107             ## H
108             self.mpo = [Hl for l in range(self.N)]
109             self.mpo[0] = Hl[-1:np.shape(Hl)[0], :, :, :]
110             # H[self.N - 1] = Hl[:, :, 0:1, :]
111             self.mpo[self.N - 1] = Hl[:, :, :, 0:1]
112             return self.mpo
113
114     # 集成DMRG
115     class DMRG:
116
117         def __init__(self, mps, mpo, Nsweep, chi, eps):
118             self.mps = mps # mps[i]: vL i vR
119             self.mpo = mpo # mpo[i]: vL_O i i* vR_O
120             self.N = len(self.mps)
121             self.Nsweep = Nsweep
```

```
122             self.chi = chi
123             self.eps = eps
124             self.C_list = None
125             self.C = None
126             self.M = None
127             self.E_list = None
128             self.E_gs = None
129
130         def zipper_left(self, i):
131             # 从左向右收缩
132             self.C = np.tensordot(self.C, self.mps[i].conj().T, axes=(2, 2)) #
                    vt vR_O [vb], vL* i* [vR*]
133             self.C = np.tensordot(self.C, self.mpo[i], axes=([1, 3], [0, 1]))
                    # vt [vR_O] vL* [i*], [vL_O] [i] i* vR_O
134             self.C = np.tensordot(self.C, self.mps[i], axes=([0, 2], [0, 1]))
                    # [vt] vL* [i*] vR_O, [vL] [i] vR
135             self.C = np.transpose(self.C, (2, 1, 0)) # vt=vR vR_O=vR_O vb=vL*
136
137         def zipper_right(self, i):
138             # 从右向左收缩
139             self.C = np.tensordot(self.C, self.mps[i], axes=(0, 2)) # [vt]
                    vL_O vb, vL i [vR]
140             self.C = np.tensordot(self.C, self.mpo[i], axes=([0, 3], [3, 2]))
                    # [vL_O] vb vL [i], vL_O i [i*] [vR_O]
141             self.C = np.tensordot(self.C, self.mps[i].conj().T, axes=([0, 3],
                    [0, 1])) # [vb] vL vL_O [i], [vL*] [i*] vR*
142             # C_right = np.transpose(C_right, (0, 1, 2)) # vt=vL vL_O=vL_O vb
                    =vR*
143
144         def contract_MPO(self, site):
145             # 获得单点哈密顿量
```

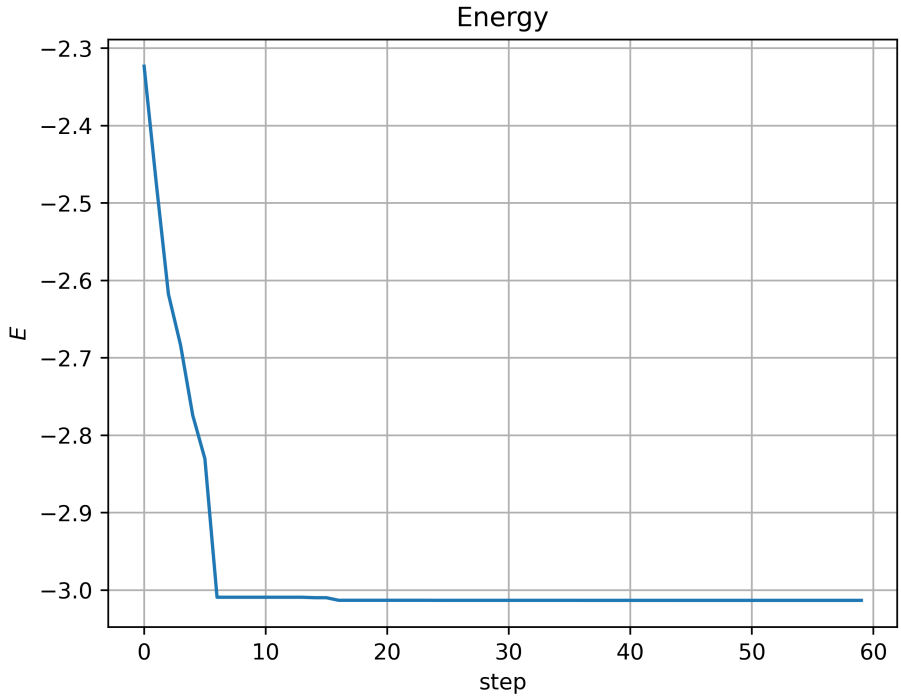
```
146             self.M = np.tensordot(self.C_list[site], self.mpo[site], axes=(1,
                    0)) # vt_L [vR_O] vb_L, [vL_O] i i* vR_O
147             self.M = np.tensordot(self.M, self.C_list[site + 2], axes=(4, 1))
                    # vt_L vb_L i i* [vR_O], vt_R [vL_O] vb_R
148             self.M = np.transpose(self.M, (0, 3, 4, 1, 2, 5)) # vt_L vb_L i i*
                     vt_R vb_R --> vt_L i* vt_R vb_L i vb_R
149
150         def update_mps_left(self, site):
151             # 从左向右单次更新
152             M_matrix = np.reshape(self.M,
153                                   (self.M.shape[0] * self.M.shape[1] * self.M.
                                      shape[2],
154                                    self.M.shape[3] * self.M.shape[4] * self.M.
                                       shape[5])) # (vt_L i* vt_R) (vb_L i vb_R)
155             eigen_val, eigen_vec = eigsh(M_matrix, k=1, which='SA', v0=self.
                    mps[site]) # Lanczos
156             eigen_vec = np.reshape(eigen_vec, (self.M.shape[0] * self.M.shape
                    [1], self.M.shape[2])) # (vL i) vR
157             U, S, V = cut(self.chi, eigen_vec)
158             self.mps[site] = U.reshape((self.M.shape[0], self.M.shape[1], U.
                    shape[1])) # vL i vR
159             if site < self.N - 1:
160                 SV = np.tensordot(np.diag(S), V, axes=(1, 0)) # vR [vR], [vR*]
                        vR*
161                 self.mps[site + 1] = np.tensordot(SV, self.mps[site + 1], axes
                        =(1, 0)) # vR [vR*], [vL] i vR
162             return eigen_val[0]
163
164         def update_mps_right(self, site):
165             # 从右向左单次更新
166             M_matrix = np.reshape(self.M,
```

```
167                                 (self.M.shape[0] * self.M.shape[1] * self.M.
                                        shape[2],
168                                  self.M.shape[3] * self.M.shape[4] * self.M.
                                        shape[5])) # (vt_L i* vt_R) (vb_L i vb_R)
169             eigen_val, eigen_vec = eigsh(M_matrix, k=1, which='SA', v0=self.
                    mps[site]) # Lanczos
170             eigen_vec = np.reshape(eigen_vec, (self.M.shape[0], self.M.shape
                    [1] * self.M.shape[2])) # vL (i vR)
171             U, S, V = cut(self.chi, eigen_vec)
172             self.mps[site] = V.reshape((V.shape[0], self.M.shape[1], self.M.
                    shape[2])) # vL i vR
173             if site > 0:
174                 US = np.tensordot(U, np.diag(S), axes=(1, 0)) # vL* [vL*], [vL]
                        vL
175                 self.mps[site - 1] = np.tensordot(self.mps[site - 1], US, axes
                        =(2, 0)) # vL i [vR], [vL*] vL
176             return eigen_val[0]
177
178         def doDMRG(self):
179             # first: left --> right
180             self.C_list = [np.ones((1, 1, 1)) for i in range(self.N + 2)]
181             for i in reversed(range(self.N)):
182                 self.C = self.C_list[i + 2]
183                 self.zipper_right(i)
184                 self.C_list[i + 1] = self.C
185             # sweep
186             self.E_list = []
187             for i in range(self.Nsweep):
188                 # left --> right
189                 for site in range(self.N):
190                     self.contract_MPO(site) # vt_L i* vt_R vb_L i vb_R
```

```
191                 E = self.update_mps_left(site)
192                 self.E_list.append(E)
193                 self.C = self.C_list[site]
194                 self.zipper_left(site)
195                 self.C_list[site + 1] = self.C
196             if len(self.E_list) > 10:
197                 if (abs(self.E_list[-1] - self.E_list[-10]) < self.eps):
198                     self.E_gs = self.E_list[-1]
199                     return
200             # right --> left
201             for site in reversed(range(self.N)):
202                 self.contract_MPO(site)
203                 E = self.update_mps_right(site)
204                 self.E_list.append(E)
205                 self.C = self.C_list[site + 2]
206                 self.zipper_right(site)
207                 self.C_list[site + 1] = self.C
208             if len(self.E_list) > 10:
209                 if (abs(self.E_list[-1] - self.E_list[-10]) < self.eps):
210                     self.E_gs = self.E_list[-1]
211                     return
212
213         self.E_gs = self.E_list[-1]
214
215 # 主程序
216 def main():
217     N = 10 # 格点数
218     mpo_eng = MPO(N=N)
219     mpo = mpo_eng.XY() # 定义模型
220     mps_eng = MPS(mpo)
221     mps = mps_eng.random_mps(v=8) # 定义初态
```

```
222         mps_eng.left_canonical()
223         mps_eng.right_canonical()
224         dmrg_eng = DMRG(mps, mpo, Nsweep=3, chi=20, eps=0) # 定义DMRG操作
225         dmrg_eng.doDMRG() # 做DMRG
226         mps_eng.mps = dmrg_eng.mps # 基态
227         S_ent = mps_eng.get_entropy() # 纠缠熵
228         print(r'Ground State Energy of XY model:', dmrg_eng.E_gs)
229         print(r'Exact GS Energy of XY model:', 0.5 - 0.5/np.sin(np.pi/(2*N+2)
                ))
230         print(r'von Neumann entropy of XY model:', S_ent)
231
232         # 基态能量收敛过程
233         plt.figure()
234         plt.title(r'Energy')
235         plt.xlabel(r'step')
236         plt.ylabel(r'$E$')
237         plt.grid()
238         plt.plot(np.arange(len(dmrg_eng.E_list)), dmrg_eng.E_list)
239         plt.savefig(r'dmrg_demon_E.png', dpi=500)
240
241     if __name__ == '__main__':
242         main()
```

其中大部分代码后注释表示收缩顺序.



5.2 TenPy

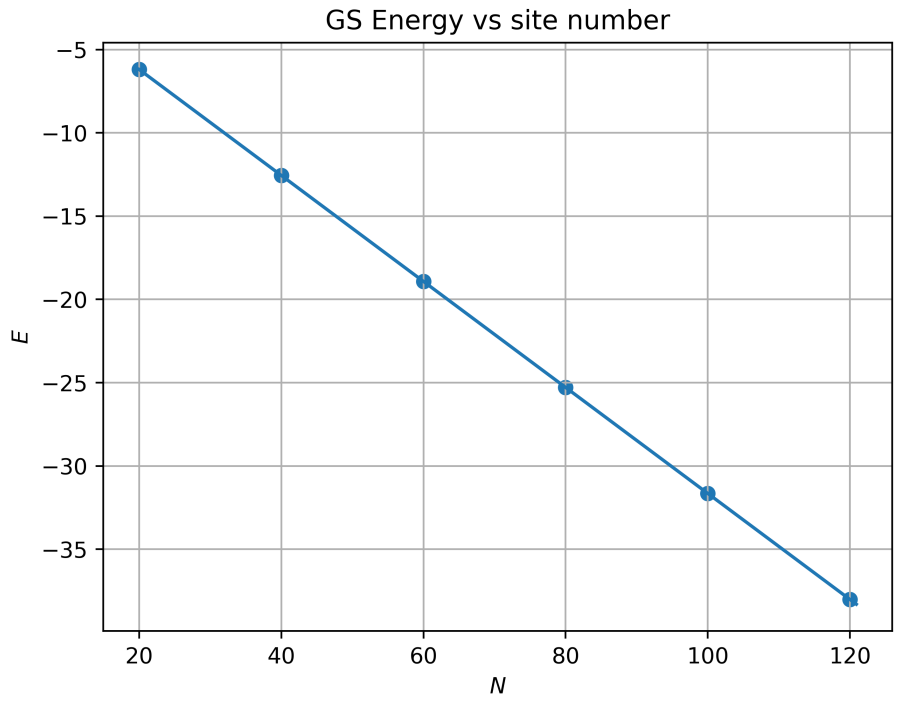
使用 tenpy 包可以更简单更直观地做 DMRG.

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
np.set_printoptions(precision=5, suppress=True, linewidth=100)

import tenpy
from tenpy.algorithms import dmrg
from tenpy.networks.mps import MPS
from tenpy.networks.site import SpinHalfSite
from tenpy.models.xxz_chain import XXZChain
tenpy.tools.misc.setup_logging(to_stdout="INFO")

def main():
    En = []
```

```
15          Nn = np.arange(20, 121, 20)
16          for N in Nn:
17              # 定义模型和初态
18              model_params = {
19                  'Jxx': 1., 'Jz': 0., 'hz': 0,
20                  'L': N,
21                  'bc_MPS': 'finite',
22              }
23              M = XXZChain(model_params)
24              site = SpinHalfSite()
25              psi = MPS.from_random_unitary_evolution([site] * N, 20, ['up', '
                    down'] * (N // 2))
26
27              # 定义DMRG操作
28              dmrg_params = {
29                  'max_E_err': 1.e-8,
30                  'trunc_params': {
31                      'chi_max': 20,
32                      'svd_min': 1.e-10,
33                  }
34              }
35              eng = dmrg.TwoSiteDMRGEngine(psi, M, dmrg_params)
36              # 做DMRG
37              E, psi = eng.run()
38              En.append(E)
39
40          # 基态能量-模型大小
41          N = np.arange(20, 121, 0.1)
42          def energy(N):
43              return 0.5 - 0.5/np.sin(np.pi/(2*N+2))
44          plt.figure()
```

```
45          plt.title(r'GS Energy vs site number')
46          plt.xlabel(r'$N$')
47          plt.ylabel(r'$E$')
48          plt.scatter(Nn, En)
49          plt.plot(N, energy(N))
50          plt.grid()
51          plt.savefig('E_N_XY.png', dpi=500)
52
53      if __name__ == '__main__':
54          main()
```

TEBD 和 TDVP 等算法也可以类似地实现.



参考文献

[1] G. Catarina, B. Murta. Density-matrix renormalization group: a pedagogical introduction. Eur. Phys. J. B 96, 111 (2023).

[2] Shi-Ju Ran, Emanuele Tirrito, Cheng Peng, Xi Chen, Luca Tagliacozzo, Gang Su, Maciej Lewenstein. Tensor Network Contractions: Methods and Applications to Quantum Many-Body Systems. (2019).

[3] https://tenpy.readthedocs.io or https://itensor.github.io/ITensors.jl.

[4] E. Koch. The Lanczos Method (Forschungszentrum Jülich GmbH, 2011), chap. 8, pp. 8.1-8.30.

[5] S.R. White. Density matrix renormalization group algorithms with a single center site. Phys. Rev. B 72, 180,403 (2005).

[6] J. Haegeman, T.J. Osborne, F. Verstraete. Post-matrix product state methods: To tangentspace and beyond. Phys. Rev. B 88, 075,133 (2013).

[7] J. Haegeman, C. Lubich, I. Oseledets, B. Vandereycken, F. Verstraete. Unifying time evolution and optimization with matrix product states. Phys. Rev. B 94, 165,116 (2016).

[8] Jung, JH., Noh, J.D. Guide to Exact Diagonalization Study of Quantum Thermalization. J. Korean Phys. Soc. 76, 670-683 (2020).

[9] H. Fehske, R. Schneider, A. Weiße (Eds.). Computational Many-Particle Physics. (2008).

[10] Adolfo Avella, Ferdinando Mancini. Strongly Correlated Systems: Numerical Methods. (2013).

[11] L. Vanderstraeten, J. Haegeman, F. Verstraete. Tangent-space methods for uniform matrix product states. SciPost Phys. Lect. Notes 7 (2019).

[12] J. Haegeman, J.I. Cirac, T.J. Osborne, I. Pižorn, H. Verschelde, F. Verstraete. Time-Dependent Variational Principle for Quantum Lattices. Phys. Rev. Lett. 107, 070601. (2011).

[13] https://zhuanlan.zhihu.com/p/390816806