第四章选做作业文档说明

Author: Yan Mi

说明:该程序利用python3.8编写,并且使用了numpy库,所以在使用过程中要保证下载好了numpy库和python的版本对应

1.ITEBD算法

(1).幂法求解最大本征值

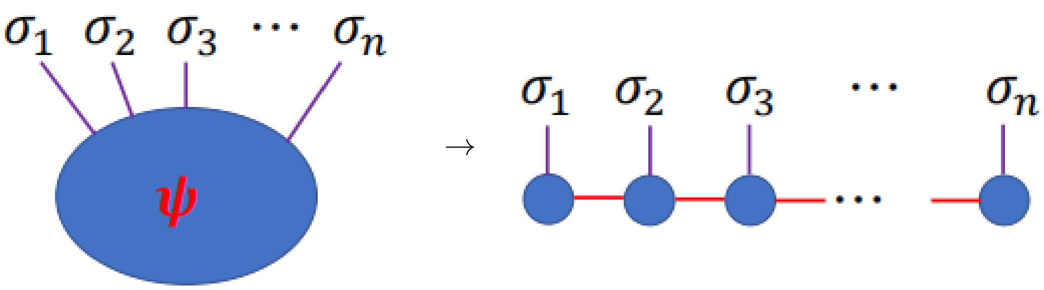
对于一个算符$\hat{H}$有本征值和本征向量：$|h_i\rangle$，h_i。其中$h_1 > h_2 > h_3 > \cdots > h_n$，那么对于任意态$|\psi\rangle = \sum c_i|h_i\rangle$有:

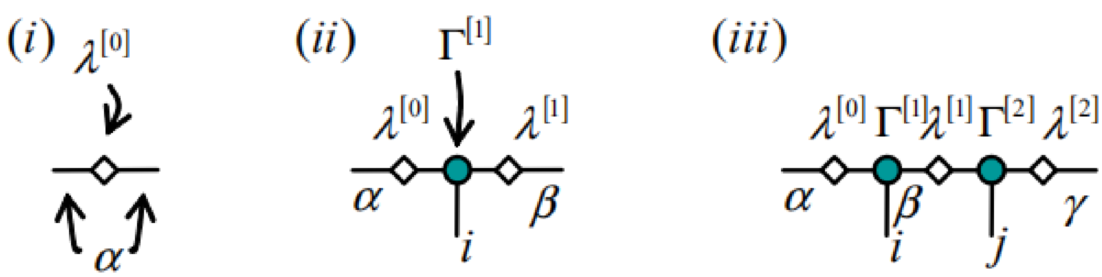
$$\hat{H}^k|\psi\rangle = h_1^k(c_1|h_1\rangle + c_2(\frac{h_2}{h_1})^k|h_2\rangle + \cdots + c_2(\frac{h_n}{h_1})^k|h_n\rangle)$$

如果让$k \to \infty$,那么由于$\frac{h_n}{h_1} < 1$,则会使得它的本征波函数贡献趋近于0，最终只剩下本征值最大的波函数。

而如果需要求解基态，那么只需要将$\hat{H}$进行一个递减的函数作用即可，比如求解H^{-1},$e^{-\beta\hat{H}}$,在后面的ITEBD算符中便使用的是后者。

(2).ITEBD





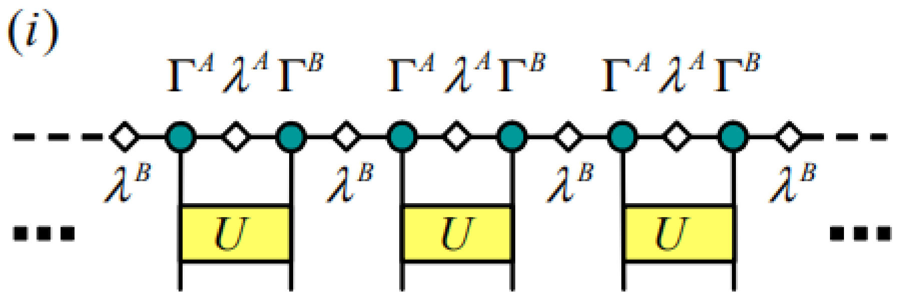
采用如上图的形式进行分解，得到MPS。

定义：

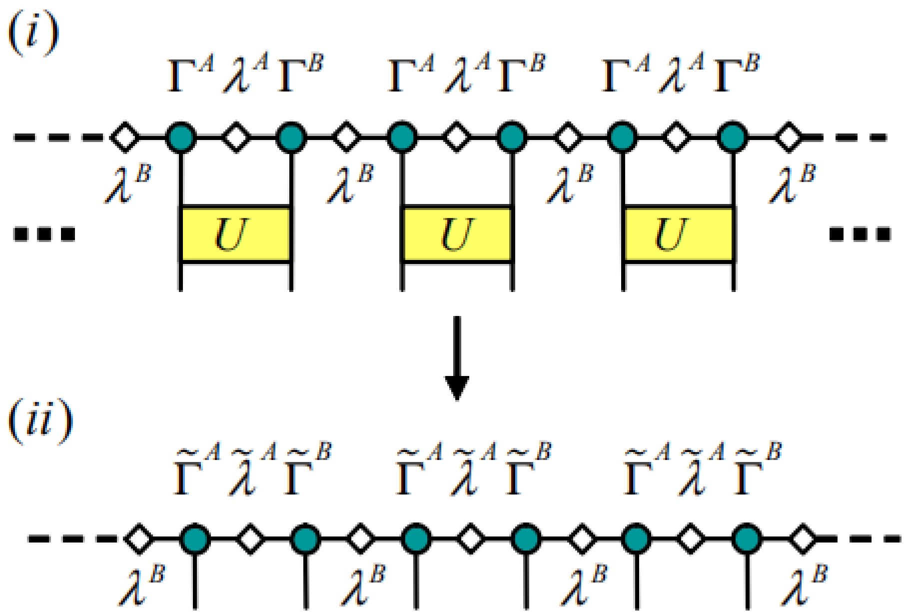
$$U^{[r,r+1]} \equiv \exp(-ih^{[r,r+1]}\delta t), \qquad \delta t \ll 1,$$

$$U^{AB} \equiv \bigotimes_{r\in\mathbb{Z}} U^{[2r,2r+1]}, \qquad U^{BA} \equiv \bigotimes_{r\in\mathbb{Z}} U^{[2r-1,2r]}. \quad (12)$$

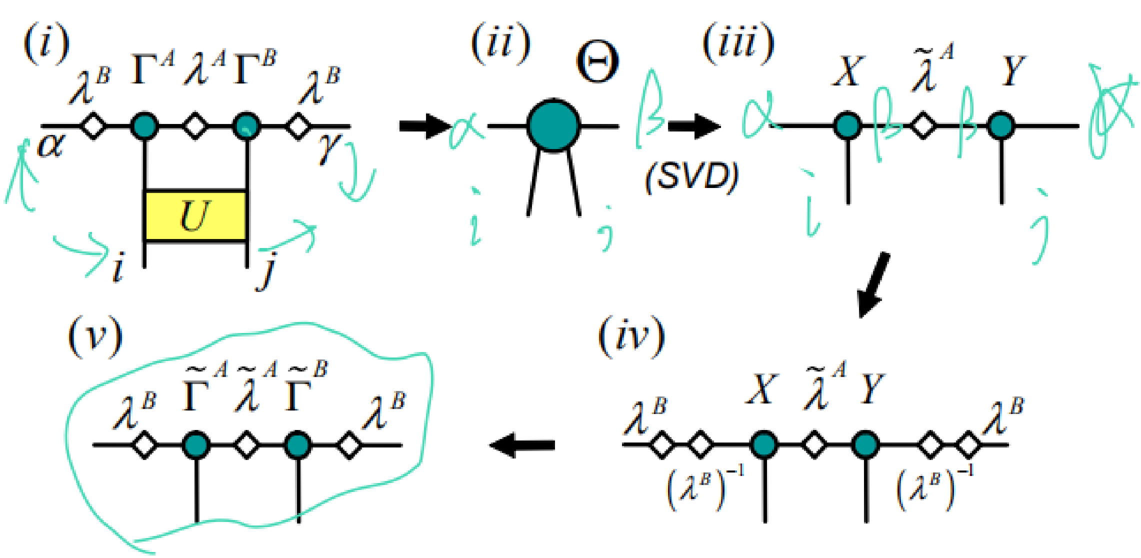
由于平移对称性的存在，分解成MPS只需要保存四个点即可如下图（$U^{AB}|\psi\rangle$作用下示意图）：



即只需要保存$\Gamma^A, \Gamma^B, \lambda^A, \lambda^B$,即可，在利用$\hat{U}$进行作用也只需更新这四个部分就可以使得全链进行更新。如下图所示:



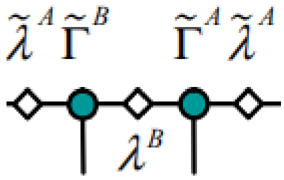
具体的分解和更新过程为:



1. 利用算符$\hat{U}$对一个平移单元进行作用，即利用$\hat{U}$和$|\psi\rangle$对指标i, j进行收缩得到一个四阶张量$\Theta_{\alpha ij\gamma}$

2. 将$\Theta_{\alpha i j \gamma}$进行指标合并称为一个二阶张量$\Theta_{[\alpha i][j\gamma]}$
3. 对于二阶张量$\Theta_{[\alpha i][j\gamma]}$进行奇异值分解，保留一定个数$(r)$的奇异值，并且得到两个左右二阶张量$X, Y$
4. 将左右二阶张量的指标进行拆分
5. 利用$(\lambda^B)^{-1}$将左右矩阵进行更新使得λ^B，重新出现，又变成原来处理的形式

由于上述过程处理的是$U^{AB}|\psi\rangle$，而我们仍需要进行处理$U^{BA}|\psi\rangle$，故须在这次处理后再对如下结构进行相同步骤的处理，来更新四个单元：



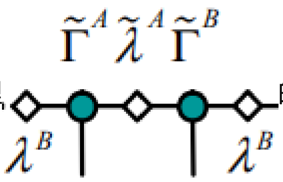
将上述步骤不断进行重复循环，直到达到收敛的能量。上述过程要注意指标的对应。

2.程序数值算法

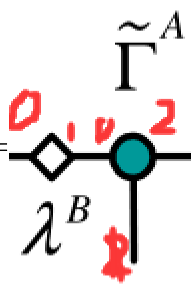
(1).产生随机的初始态$|\psi\rangle$

要产生随机的$|\psi\rangle$，只需要产生随机的四个待更新单元即可$\Gamma^A, \Gamma^B, \lambda^A, \lambda^B$：

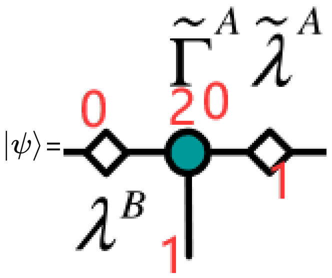
```
self.sigma=np.random.rand(2,self.r)
self.g=np.random.rand(2,self.r,3,self.r)
```

然后首先第一次处理是处理的结构，在该处理中需要进行张量指标的收缩运算：

```
Psi = np.tensordot(np.diag(self.sigma[B,:]),self.g[A,...],axes=(1,0))
```

上述代码使$|\psi\rangle =$

```
Psi = np.tensordot(Psi,np.diag(self.sigma[A,:]),axes=(2,0))
```



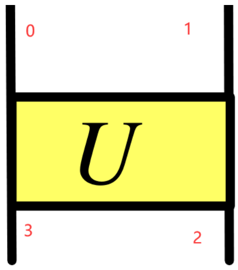
整体代码为：

```
Psi = np.tensordot(np.diag(self.sigma[B,:]),self.g[A,...],axes=(1,0))
Psi = np.tensordot(Psi,np.diag(self.sigma[A,:]),axes=(2,0))
Psi = np.tensordot(Psi,self.g[B,...],axes=(2,0))
Psi = np.tensordot(Psi,np.diag(self.sigma[B,:]),axes=(3,0))
```

$np.tensordot(A, B, (i, j))$将$A$的第$i$位和$B$的第$j$位指标进行收缩，指标收缩后将产生新的指标位。

(2).对态进行$\hat{U}$作用

四阶张量$\hat{U}$的形式和指标位为：



将$\hat{U}$和$|\psi\rangle$进行收缩指标可得到结果：

```
Psi = np.tensordot(Psi,self.u,axes=((1,2),(0,1)))
```

(3).奇异值分解

首先必须得将$\Theta_{\alpha ij\gamma}$四阶张量合并成$\Theta_{[\alpha i][j\gamma]}$二阶张量，这样进行奇异值分解后得到得左右矩阵的形式为：

$$X_{[\alpha i]k}, Y_{k,[j\gamma]}$$

代码为：

```
Psi = np.reshape(np.transpose(Psi,(0,2,3,1)),(self.r*3,3*self.r))
[Left,newsigma,Right] = np.linalg.svd(Psi)
```

要进行$np.transpose$的是因为在使用$\hat{U}$进行收缩之后态的指标顺序不再是$\alpha ij\gamma$,而变成了$\alpha\gamma ij$

奇异值分解后得到的R, L张量由于它们的指标顺序都为类似Γ的指标顺序，所以只需要进行维度变化就可以得到对应的三阶张量

(4).能量计算

对于一个确定的态$|\psi\rangle$，它的基态能量估算为

$$e^{-\tau h}|\psi\rangle = e^{-\tau E_0}|\psi\rangle$$

两边进行内积可以得到

$$e^{-2\tau E_0}\langle\psi|\psi\rangle = (e^{-\tau h}|\psi\rangle)^2$$

所以：

$$E_0 = -\frac{1}{2\tau}ln((e^{-\tau h}|\psi\rangle)^2)$$

即利用代码进行表示为：

```
Energy = -np.log(np.sum(Psi**2))/(self.deltat*2)
```

3.结果展示

运行结果为

```
迭代次数为100时的能量为-0.915943503121
迭代次数为833时的能量为-1.398068682760
迭代次数为5000时的能量为-1.401491456686
迭代次数为10000时的能量为-1.401491456757
```

可以发现最后基态能量收敛到$E_0 = -1.401491456757$

4.源代码

```
# -*- coding: utf-8 -*-
#author:Miyan
#Start data:2021.5.12
#End data:2021.5.12

import numpy as np

#精度和时间间隔
eps = 0.0000000001
deltat = 0.01
maxtimes = 10000

def beautifulPrintMatrix(A):
    n = A.shape[0]
    m = A.shape[1]
    for i in range(0,n):
        for j in range(0,m):
            oe = A[i,j]
            print("%.4f + %.1fi "%(oe.real,oe.imag),end="  ")
        print("")

def getSeoMatrix():
```

```
    ts = 2 ** 0.5
    sz = np.array([[1,0,0],[0,0,0],[0,0,-1]])
    sx = np.array([[0,ts/2,0],[ts/2,0,ts/2],[0,ts/2,0]])
    sy = np.array([[0,ts,0],[-ts,0,ts],[0,-ts,0]])/2j
    res = np.kron(sz,sz) + np.kron(sx,sx) + np.kron(sy,sy)
    return np.real(res)

# beautifulPrintMatrix(getSeoMatrix())

class Solution():
    def __init__(self,r,precision,deltat,maxtimes):
        self.deltat = deltat
        self.r = r
        self.maxtimes = maxtimes
        self.precision = precision
        self.h = getSeoMatrix()
        self.sigma=np.random.rand(2,self.r)
        self.g=np.random.rand(2,self.r,3,self.r)

    def getTimeOperator(self):
        [va,ve] = np.linalg.eig(self.h)
        '''u = exp(-th)'''
        u = self.u = np.array(np.mat(ve)*np.diag(np.exp(-self.deltat*va))*np.mat(ve).H)
        self.u = np.reshape(self.u,(3,3,3,3))
        return u

    def itebd(self):
        self.getTimeOperator()
        # E1 = 0
        E2 = 0
        Energy = 2
        times = 0
        [A,B] = [1,0]
        # while(abs(Energy - E1) > self.precision):
        while(times < self.maxtimes):
            [A,B] = [B,A]
            Psi = np.tensordot(np.diag(self.sigma[B,:]),self.g[A,...],axes=(1,0))
            Psi = np.tensordot(Psi,np.diag(self.sigma[A,:]),axes=(2,0))
            Psi = np.tensordot(Psi,self.g[B,...],axes=(2,0))
            Psi = np.tensordot(Psi,np.diag(self.sigma[B,:]),axes=(3,0))


            Psi = np.tensordot(Psi,self.u,axes=((1,2),(0,1)))

            #分解
            Psi = np.reshape(np.transpose(Psi,(0,2,3,1)),(self.r*3,3*self.r))
            [Left,newsigma,Right] = np.linalg.svd(Psi)

            #归一化
            self.sigma[A,:] = newsigma[0:self.r]/np.sqrt(np.sum(newsigma[0:self.r]**2))

            #指标断开
            Left = np.reshape(Left[0:3*self.r,0:self.r],(self.r,3,self.r))
            self.g[A,...] = np.tensordot(np.diag(self.sigma[B,:]**(-1)),Left,axes=(1,0))

```

```python
            Right = np.reshape(Right[0:self.r,0:3*self.r],(self.r,3,self.r))
            self.g[B,...] = np.tensordot(Right,np.diag(self.sigma[B,:]**(-1)),axes=(2,0))

            # E1 = E2
            [E2,Energy] = [Energy,-np.log(np.sum(Psi**2))/(self.deltat*2)]

            if(times == self.maxtimes // 12):
                print("迭代次数为%d时的能量为%.12f"%(times,(Energy+E2)/2))
            elif(times == self.maxtimes // 2):
                print("迭代次数为%d时的能量为%.12f"%(times,(Energy+E2)/2))
            elif(times == 100):
                print("迭代次数为%d时的能量为%.12f"%(times,(Energy+E2)/2))

            times += 1
        return (Energy + E2)/2

test = Solution(30,eps,deltat,maxtimes)
print("迭代次数为%d时的能量为%.12f"%(maxtimes,test.itebd()))

```