第六章选做作业文档说明

Author: YanMi

说明:该程序利用python3.8编写,并且使用了numpy，scipy,matplotlib库,所以在使用过程中要保证下载好了numpy，scipy,matplotlib库和python的版本对应，并且python是一个缩进较为严格的语言，故在运行时对缩进格式需要注意。

1.DMRG（密度矩阵重整化群）算法

Steven R. White

1.1.简介

密度矩阵重整化群 (Density Matrix Renormalization Group)，简称DMRG，是一种数值算法，于公元1992年由美国物理学家Steven R. White提出。密度矩阵重整化群是用来计算量子多体系统（例如：Hubbard model、t-J模型、海森堡模型，等等）的一个非常精准的数值算法，在一维或准一维的系统可以得到系统尺寸很大且很准确的计算结果，但是在二维的量子多体系统中却很难达到所需要的精确度。此算法仍无法计算三维的量子系统。

1.2 算法

DMRG算法的核心点是，将研究对象（即一个超块）分为系统和环境两个部分。

在该Heisenberg Model：

$$H = \sum_i S_i S_{i+1}$$

在计算$L = 4$的格点时，我们可以将两个格点作为系统，两个格点作为环境。

接下来的计算步骤为：

- 对超块的矩阵进行对角化，得到该系统的波函数（通常为基态波函数）ψ
- 将波函数表征为系统和环境波函数的矩阵ψ_{ij}即指标i表征系统，j标征环境
- 利用ψ_{ij}分别求出系统和环境的约化密度矩阵：

$$\rho^s_{ii'} = \sum_{j,j'} \psi_{ij}\psi_{ij'}$$
$$\rho^s_{jj'} = \sum_{j,j'} \psi_{ij}\psi_{i'j}$$

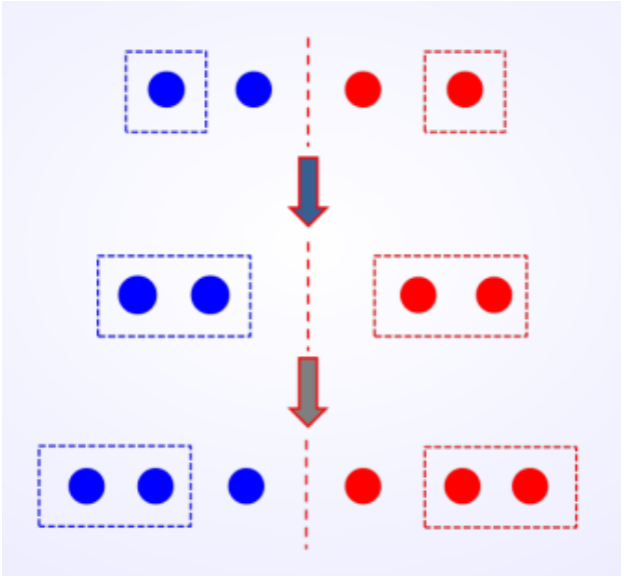
- 将约化密度矩阵进行对角化：

$$\rho^s = U^s \Lambda^s U^{s\dagger}$$
$$\rho^e = U^e \Lambda^e U^{e\dagger}$$

  对对角化后的特征向量进行截断，取特征值较大的前t项，得到新的基矢。
- 利用新基矢将系统和环境的各个矩阵进行表示，需要表示的为

$$H^s, H^e, S_2, S_3$$

- 在上述步骤的基础上，在系统块和环境块中分别插入两个点：



使得超块和系统的哈密顿矩阵为：

$$H^{Super} = \bar{H}^s + \vec{\bar{S}}_2 \cdot \vec{S}_3 + \vec{S}_3 \cdot \vec{S}_4 + \vec{S}_4 \cdot \vec{\bar{S}}_5 + \bar{H}^e$$

$$H^s = \bar{H}_s + \vec{\bar{S}}_2 \cdot \vec{S}_3$$

$$H^e = \bar{H}_e + \vec{S}_4 \cdot \vec{\bar{S}}_5 + \bar{H}^e$$

在此基础上继续上述步骤，直到能量收敛。

2.程序算法与结果展示

2.1.结果展示

在计算格点数目$L = 1000$,截断指标t不同的情况下，各自的能量收敛情况为：

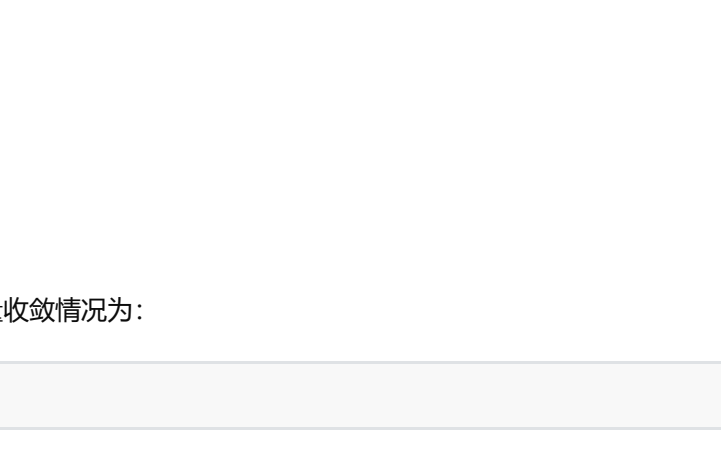
截断指标t	**基态能量E_g**
5	-0.44181701506398807
10	-0.4427732988065388
25	-0.44294457391870545
40	-0.4429549012128679

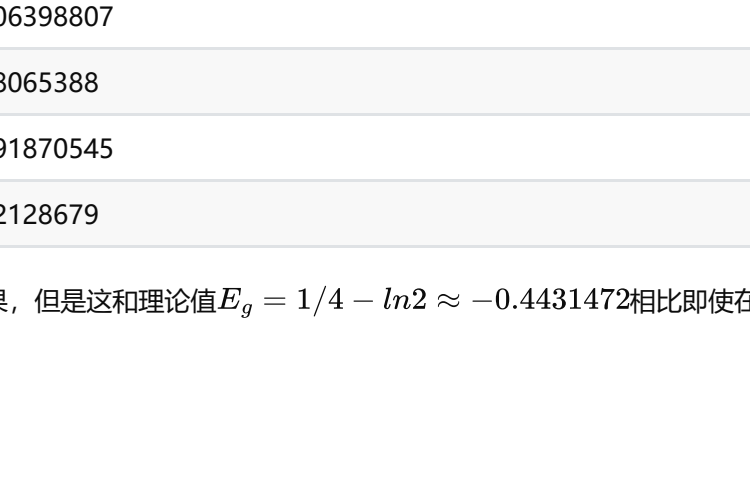
可以发现上述结果即使在截断指标较小的情况下，也会具有较好的结果，但是这和理论值$E_g = 1/4 - ln2 \approx -0.4431472$相比即使在截断指标$t = 40$的情况下，精度依然不够。

2.2 程序改进

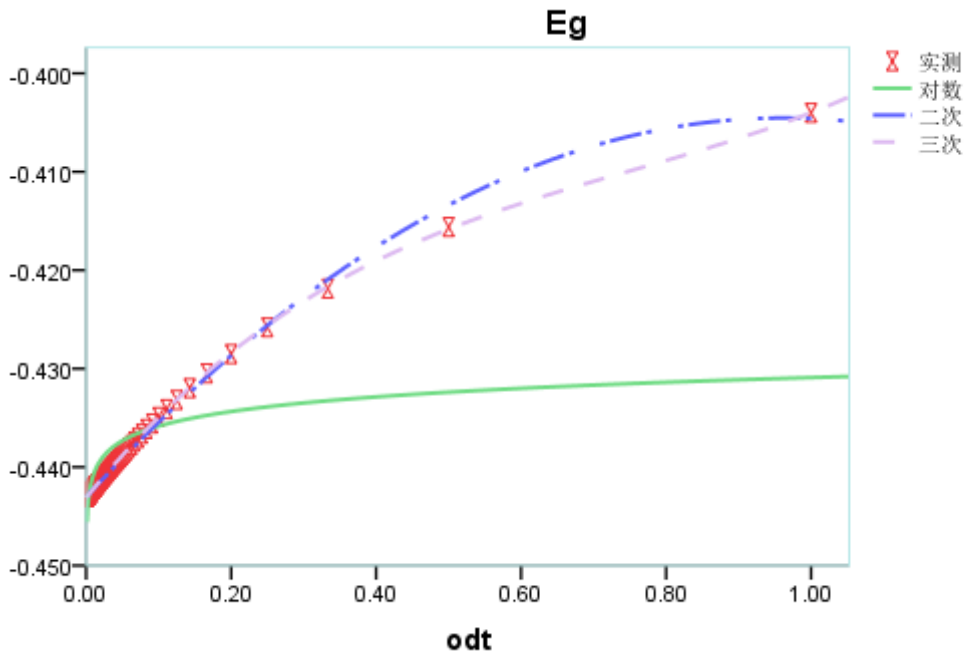
2.2.1 采用拟合收敛到定值

各指标的收敛图（下图中的横轴坐标为迭代次数$n = (L-4)/2$，图中出现了错误）如下：





从上面的四张图中能够看出，随着迭代次数的增加，收敛速度成直线下降，而下降的这个在$L=100$左右，在$L=100$左右之后几乎成一条直线，虽然无法进行线性拟合（线性拟合的斜率是一定的，所以导致无法进行$L=\infty$的预测）。所以为了进行拟合操作，可以对E_g-1/L进行拟合，利用二次，三次和对数拟合可以得到如下结果（odt为$1/L$）：



其模型参数如下所示：

方程	R^2	F	自由度1	自由度2	显著性	常量	b_1	b_2	b_3
对数	0.548	601.847	1	496	0.000	-0.430905	0.002		
二次	0.997	80666.954	2	495	0.000	-0.443034	0.080	-0.042	
三次	1.000	2591456.445	3	494	0.000	-0.443108	0.091	-0.093	0.041

从上表可以发现，三次形式的拟合完全符合$R^2=1$,故可以使用该形式，由于$L\to\infty$，$1/L\to 0$,所以基态能量为：-0.443108

2.2.2 利用White论文中的方法

查看White的论文，可以发现，他在文章中有一句话：

Table I shows results for the ground-state energies of the infinite $S=\frac{1}{2}$ and $S=1$ chains. The energy per site was determined from the difference in total energy of the system $A\cdots A$ from one iteration to the next. The procedure was iterated until the energy converged to about eight digits, about 100 iterations for the $S=1$ case.

也就是，他本人使用的是这个方法去计算能量。在100个格点下，利用该方法进行测试得到的结果为：

截断指标t	E_g	E_g with first method
5	-0.4421720109180107	-0.44181701506398807
10	-0.44291169771029004	-0.4427732988065388
20	-0.443115048287261	-0.44294457391870545
40	-0.4431277298395493	-0.4429549012128679

考虑到第二列的数据比第三列的数据整整少900个格点，所以White的方法确实可以对收敛到真实结果起到一定的作用。所以计算结果为：-0.4431277298395493

3.源代码

```
# -*- coding: utf-8 -*-
#author:Miyan
#Start data:2021.6.6
#End data:2021.6.7

import numpy as np
import time
from scipy.sparse.linalg import eigsh
import scipy
import time
import matplotlib.pyplot as plt
from scipy.linalg import norm
```

```
from scipy.optimize import curve_fit

def dagger(Array):
    return np.conj(np.transpose(Array))

def dim(Array):
    return max(Array.shape)

def getRandomVector(dimension):
    res = np.random.random(dimension)
    res /= np.sqrt(norm(res))
    return res

class Solution:
    def __init__(self):
        self.sx = np.array([[0,1/2+0j],[1/2,0]])
        self.sy = np.array([[0,-0.5j],[0.5j,0]])
        self.sz = np.array([[1/2+0j,0],[0,-1/2+0j]])

        self.bo = np.array([
                    [0.25,0,0,0],
                    [0,-0.25,0.5,0],
                    [0,0.5,-0.25,0],
                    [0,0,0,0.25]
                    ])
        self.truncationIndex = 40
        self.eps = 1e-5
        # self.loop = 2**31 - 1
        self.loop = 100

    def dmrg(self):
        Sys = np.copy(self.bo)
        Env = np.copy(self.bo)

        '''
        超块矩阵： S1 S2 I3 I4 + I1 S2 S3 I4 + I1 I2 S3 S4
        '''
        Super = np.kron(self.bo,np.eye(4)) + np.kron(np.kron(np.eye(2),self.bo),np.eye(2)) + np.kron(np.eye(4),self.bo)
        lastEnergy = 0
        e = []
        for L in range(4, self.loop, 2):
            startVector = getRandomVector(dim(Super))
            [va,ve] = eigsh(Super,k = 1,v0 = startVector)

            # energy = va[0]
            # print(energy/L)
            energy = (va[0] - lastEnergy) / 2
            lastEnergy = va[0]
            print(energy)
            length = int(np.sqrt(dim(ve)))
            psi = np.reshape(ve,(length,-1))

            SystemRho = psi @ dagger(psi)
            EnvironmentRho = dagger(psi) @ psi
            [SystemValue,SystemVector]=np.linalg.eigh(-SystemRho)
            [EnvironmentValue,EnvironmentVector] = np.linalg.eigh(-EnvironmentRho)
            SystemVector = SystemVector[:, 0 : self.truncationIndex]
            EnvironmentVector = EnvironmentVector[:, 0 : self.truncationIndex]

            #系统和环境各自插入一个格点，构造下一个系统和环境的哈密顿量
            length = int(length / 2)
            eye = np.eye(length)

            sxbar = dagger(SystemVector) @ np.kron(eye,self.sx) @ SystemVector
            sybar = dagger(SystemVector) @ np.kron(eye,self.sy) @ SystemVector
            szbar = dagger(SystemVector) @ np.kron(eye,self.sz) @ SystemVector
            Scoupling = np.kron(sxbar, self.sx) + np.kron(sybar, self.sy) + np.kron(szbar, self.sz)
            Sys = np.kron(dagger(SystemVector) @ Sys @ SystemVector,np.eye(2)) + Scoupling

            sxbar = dagger(EnvironmentVector) @ np.kron(self.sx,eye) @ EnvironmentVector
            sybar = dagger(EnvironmentVector) @ np.kron(self.sy,eye) @ EnvironmentVector
            szbar = dagger(EnvironmentVector) @ np.kron(self.sz,eye) @ EnvironmentVector
            Scoupling = np.kron(self.sx,sxbar) + np.kron(self.sy,sybar) + np.kron(self.sz,szbar)
            Env = np.kron(np.eye(2),dagger(EnvironmentVector) @ Env @ EnvironmentVector) + Scoupling

            spaceDimension = int(dim(Env)/2)
            Super=np.kron(Sys,np.eye(spaceDimension*2)) + np.kron(np.eye(spaceDimension),np.kron(self.bo,np.eye(spaceDimension))) +
np.kron(np.eye(spaceDimension*2),Env)

```

```
            # lastEnergy = energy
            # e.append(lastEnergy/L)

            e.append(lastEnergy)

        return [lastEnergy / L,L,e]


s = Solution()
time_start = time.time()
[energy,L,e] = s.dmrg()
time_end = time.time()

Lens = np.array([2*i+4 for i in range(0,dim(np.array(e)))])
plt.figure(figsize=(14,14*0.618))
plt.plot(Lens,e,c = "red",linewidth=3.0)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
plt.title(r"$E_g - n$ in $t = %d$ "%s.truncationIndex,fontsize = 25)
plt.ylabel(r"$E_g$",fontsize=20)
plt.xlabel(r"$n$",fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend([r"$E_g$"],fontsize=20)

print("Go for length = %d  and  the energy is %2.12f"%(L,energy))
```