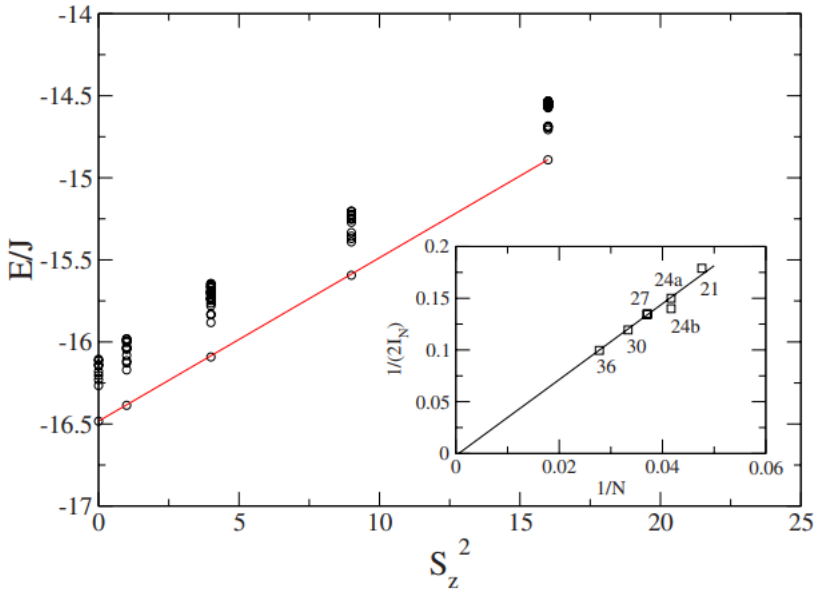
変遷

PhysRevB.78.140405 (1).pdfのFIG2の再現を行う



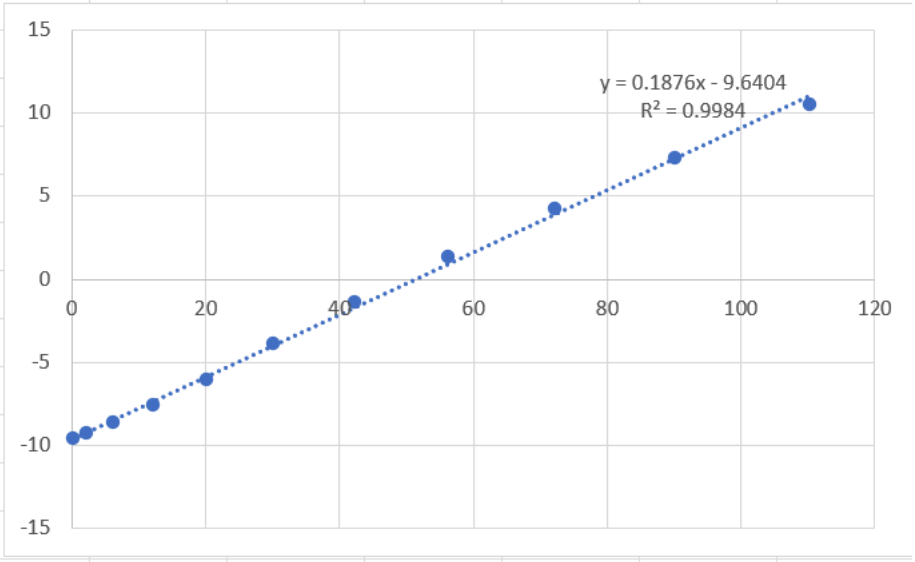
論文内の再現したいグラフ

このグラフ内のグラフにおけるN=21,N=24の再現に挑む。S_{TOT}^{Z}-最小エネルギー固有値のグラフの傾きを1/Nを横軸にプロットしたものが、この図である。 N=21では0.175,N=24では0.15程度になるはずである。

20250712の現状

./../output/N=21_J=1_D=0.1.txtの情報をExcelにプロットしたものが以下の図である。

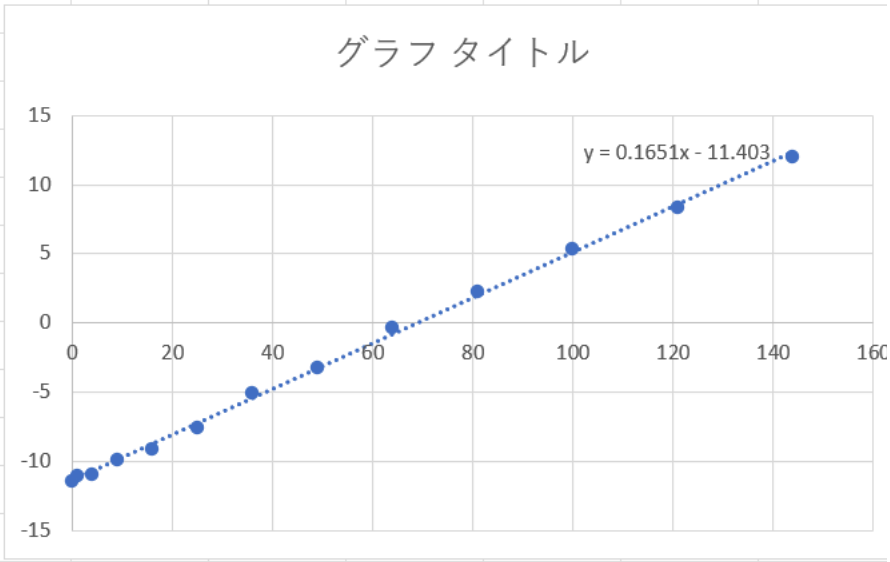
S_z^2	S_Z^2	E/j
21	110.25	10.5
19	90.25	7.326792
17	72.25	4.253632
15	56.25	1.305599
13	42.25	-1.37114
11	30.25	-3.8661
9	20.25	-6.03704
7	12.25	-7.56874
5	6.25	-8.62156
3	2.25	-9.28963
1	0.25	-9.61921



論文内の再現したいグラフ(N=21)

./../output/N=24_J=1_D=0.1.txtの情報をExcelにプロットしたものが以下の図である。

S_Z^2	S_Z^2	E/j
24	144	12
22	121	8.352608
20	100	5.351615
18	81	2.283402
16	64	-0.41045
14	49	-3.22322
12	36	-5.12943
10	25	-7.62378
8	16	-9.16781
6	9	-9.88373
4	4	-10.9582
2	1	-11.0807
0	0	-11.4631



論文内の再現したいグラフ(N=24)

どちらも、傾きが大きく出てしまっている。系の設定なのか、ロジックが違うのか。とりあえずは教授作成のTITPACKと比較検討を進めることとする。

バージョンについて

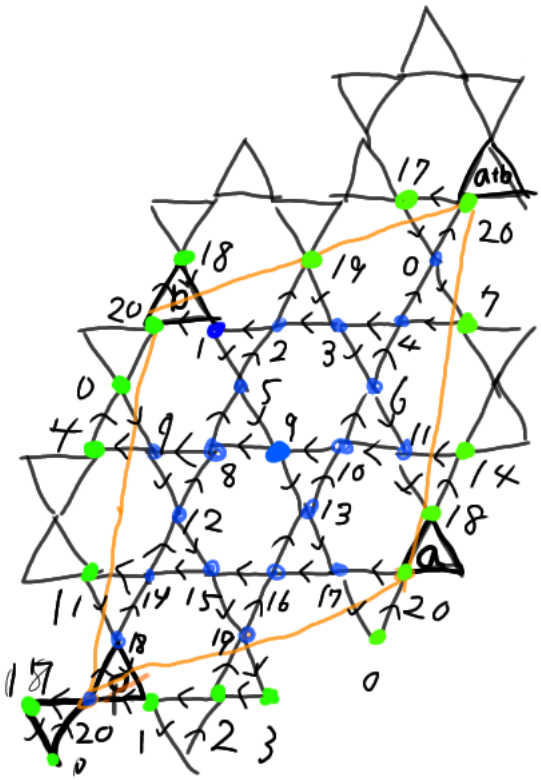
本件のソースは全5バージョン存在する

バージョン	仕様
v1.py	全状態2^Nについてハミルトニアンを構成する
v2.py	v1に対して疎行列によるアプローチを追加→ 頓挫
v3.py	全状態ではなくZ_{tot}で分割し、計算量の削減
v4.py	v3に対して、疎行列アプローチを導入,外部ファイル出力も導入
v5.py	v4のメモリ削減、高速化を行った

v5はv4にビット演算、目盛り節約を組んだだけなので、ロジックはv4と等しいので、v4で解説することとする

系の解説

サイト数：21の系



N=21のときのサイトの様子

各サイトに対して0~20までの番号と矢印で結合を表している。結合はinput/N=21.txtで定義されている

物理理論

本件でのハミルトニアンは以下のように定義される

$$\begin{aligned}\mathscr{H} &= J\sum_{i,j}\boldsymbol{S}_i\cdot\boldsymbol{S}_j+\sum_{i,j}D_{i,j}\cdot(\boldsymbol{S}_i\times\boldsymbol{S}_j)\\ &= J\sum_{i,j}S_i^xS_j^x+S_i^yS_j^y+S_i^zS_j^z+\sum_{i,j}D_{i,j}^z(S_i^xS_j^y-S_i^yS_j^x)\end{aligned}$$

ここで昇降演算子を定義し、S^x, S^yを頑張る

$$\begin{aligned}S^+ &= S^x+iS^y\\ S^- &= S^x-iS^y\end{aligned}$$

したがって

$$\begin{aligned}S^x &= \frac{S^++S^-}{2}\\ S^y &= \frac{S^+-S^-}{2i}\\ S^z = i[S_x,S_y] &= -\frac{1}{2}[S^+,S^-]\end{aligned}$$

スピン演算子の積をまとめる

$$
\begin{aligned}
S_i^x S_j^x &= \frac{1}{4}(S_i^+ + S_i^-)(S_j^+ + S_j^-) \\
&= \frac{1}{4}(S_i^+ S_j^+ + S_i^+ S_j^- + S_i^- S_j^+ + S_i^- S_j^-) \\
S_i^y S_j^y &= -\frac{1}{4}(S_i^+ - S_i^-)(S_j^+ - S_j^-) \\
&= \frac{1}{4}(-S_i^+ S_j^+ + S_i^+ S_j^- + S_i^- S_j^+ - S_i^- S_j^-) \\
S_i^x S_j^y &= \frac{1}{4i}(S_i^+ + S_i^-)(S_j^+ - S_j^-) \\
&= \frac{1}{4i}(S_i^+ S_j^+ - S_i^+ S_j^- + S_i^- S_j^+ - S_i^- S_j^-) \\
S_i^y S_j^x &= \frac{1}{4i}(S_i^+ - S_i^-)(S_j^+ + S_j^-) \\
&= \frac{1}{4i}(S_i^+ S_j^+ + S_i^+ S_j^- - S_i^- S_j^+ - S_i^- S_j^-)
\end{aligned}
$$

それぞれの和に関してもまとめておく

$$
\begin{aligned}
S_i^x S_j^x + S_i^y S_j^y &= \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+) \\
S_i^x S_j^y - S_i^y S_j^x &= \frac{1}{2i}(-S_i^+ S_j^- + S_i^- S_j^+)
\end{aligned}
$$

これをハミルトニアンに適用する

$$
\begin{aligned}
\mathscr{H} &= J\sum_{i,j}(S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z) + \sum_{i,j} D_{i,j}^z (S_i^x S_j^y - S_i^y S_j^x) \\
&= \frac{J}{2}\sum_{i,j}(S_i^+ S_j^- + S_i^- S_j^+) + \frac{J}{4}\sum_{i,j}[S_i^+, S_i^-][S_j^+, S_j^-] \\
&+ \sum_{i,j}\frac{D_{i,j}}{2i}(-S_i^+ S_j^- + S_i^- S_j^+)
\end{aligned}
$$

第一項

i番目とj番目のスピン状態を$|\sigma_i\sigma_j\rangle$のケットで表し、第一項に適用すると以下の表になる

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	0
$\|\uparrow\downarrow\rangle$	$\frac{J}{2}\|\downarrow\uparrow\rangle$
$\|\downarrow\uparrow\rangle$	$\frac{J}{2}\|\uparrow\downarrow\rangle$
$\|\downarrow\downarrow\rangle$	0

第二項

同様に以下の表

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	$\frac{J}{4}\|\uparrow\uparrow\rangle$
$\|\uparrow\downarrow\rangle$	$-\frac{J}{4}\|\uparrow\downarrow\rangle$
$\|\downarrow\uparrow\rangle$	$-\frac{J}{4}\|\downarrow\uparrow\rangle$
$\|\downarrow\downarrow\rangle$	$\frac{J}{4}\|\downarrow\downarrow\rangle$

第三項

同様に以下の表

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	0
$\|\uparrow\downarrow\rangle$	$\frac{D_{ij}}{2i}\|\downarrow\uparrow\rangle$
$\|\downarrow\uparrow\rangle$	$-\frac{D_{ij}}{2i}\|\uparrow\downarrow\rangle$
$\|\downarrow\downarrow\rangle$	0

コード全容

まずは定数部分から見ていく

```
#定数**********************************
bond_mod_J = 1
bond_mod_D = 0.1
accuracy = 1e-2 #精度
Sites = []
SITE_NUM = 0
inputFile = "N=21"
```

オブジェクト定義

サイト状態

```
#クラス定義**************************
class Site():
    def __init__(self,Id,bond,mod):
        self.id = Id
        self.bond = bond
        self.mod = mod
```

例えば0番サイトの情報はid=0,bond=[7,20],mod=[-1,+1]である

状態

```
class State():
    def __init__(self,data):
        #dataは0,1配列でdata[i]はId=iのサイトの1,0を表す
        self.data = data
        Id = 0
        for spin in self.data:
            Id = Id*2+spin
        self.id = Id
```

これはスピン状態を捕獲しておくものであり、dataはサイトの数(21)個の要素の配列であり、各要素は0or1になる.Idはdataを二進数だと思ったときに与えられる整数である。有用性はあとで。

メイン関数

```
#inputfileからの読み込み
setter()
SITE_NUM = len(Sites)
min_enegy = float("inf")
#全S_TOTに対して最低Eを取る(E(-1) = E(1))より、半分は割愛
for S_TOT in range(SITE_NUM,-1,-2):
    #状態と逆変換を初期化
    States.clear()
    idToInd.clear()
    StateInd = 0
    #状態と逆変換を列挙
    for ups in combinations(range(SITE_NUM),(S_TOT+SITE_NUM)//2):
        #全部を下向き
        spins = [0]*SITE_NUM
        #選ばれたものを上向き
        for site in ups:
            spins[site] = 1
        #そんなものを配列に捕獲
        States.append(State(spins))
```

```
        #Idとからindexを高速に計算できるように
        idToInd[State(spins).id] = StateInd
        StateInd += 1
    dataDict = {}
    #ハミルトニアン形成{(元のId,変換後のID) = 係数}
    for state in States:
        #Hamilから帰ってくるデータは[[適用後スピンのStateオブジェクト,係数],...]
        col_data = Hamil(state)
        rowId = idToInd[state.id]
        for _col,_data in col_data:
            colId = idToInd[_col.id]
            if((rowId,colId) in dataDict):
                dataDict[(rowId,colId)] += _data
            else:
                dataDict[(rowId,colId)] = _data
    #疎行列用にコンバート
    max_nnz = len(dataDict)
    rows = np.empty(max_nnz, dtype=int)
    cols = np.empty(max_nnz, dtype=int)
    datas = np.empty(max_nnz, dtype=complex)
    ptr = 0
    for key,data in dataDict.items():
        rows[ptr] = key[0]
        cols[ptr] = key[1]
        datas[ptr] = data
        ptr += 1
    print("create list data")
    #結局ハミルトニアンが小さいなら蜜行列を使った方が早いので実装
    if(ALL_STATE_NUM <= 100):
        hamil = coo_matrix((datas, (rows, cols)), shape=(ALL_STATE_NUM, ALL_STATE_NUM)).tocsr()
        eigvals, eigvecs = eig(hamil.toarray())
        mineigval = float("inf")
        for enegy in eigvals:
            mineigval = min(mineigval,enegy.real)
        min_enegy = min(min_enegy,mineigval)
        print(S_TOT,"環境下での最小実部の固有値:", math.floor(mineigval/accuracy*10)*accuracy/10)
    #疎行列の時はこっち
    else:
        hamil = coo_matrix((datas, (rows, cols)), shape=(ALL_STATE_NUM, ALL_STATE_NUM)).tocsr()
        eigvals, eigvecs = eigs(hamil, k=1, which='SR',tol=accuracy)
        mineigval = float("inf")
        for enegy in eigvals:
            mineigval = min(mineigval,enegy.real)
        min_enegy = min(min_enegy,mineigval)
        print(S_TOT,"環境下での最小実部の固有値:", math.floor(mineigval/accuracy*10)*accuracy/10)
    logFile.write(f"{(S_TOT/2)**2},{eigvals[0].real}¥n")
print("全ての状態での最小エネルギー:",min_enegy)
```

第一項(x,yの内積)

さっきの表を再現すればよい。下に再掲

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	0
$\|\uparrow\downarrow\rangle$	$\frac{J}{2}\|\downarrow\uparrow\rangle$
$\|\downarrow\uparrow\rangle$	$\frac{J}{2}\|\uparrow\downarrow\rangle$
$\|\downarrow\downarrow\rangle$	0

```
def xyprod(state):
    ans = []
    spins = state.data
    for site in range(SITE_NUM):
        for bond_site in Sites[site].bond:
```

```
        copy_spins = spins.copy()
        if(copy_spins[site] != copy_spins[bond_site]):
            copy_spins[site],copy_spins[bond_site] = copy_spins[bond_site],copy_spins[site]
            modified_state = States[idToInd[spinToId(copy_spins)]]
            ans.append([modified_state,1/2*bond_mod_J])
    return ans
```

第二項(zの内積)

さっきの表を再現すればよい。下に再掲

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	$\frac{J}{4}\|\uparrow\uparrow\rangle$
$\|\uparrow\downarrow\rangle$	$-\frac{J}{4}\|\uparrow\downarrow\rangle$
$\|\downarrow\uparrow\rangle$	$-\frac{J}{4}\|\downarrow\uparrow\rangle$
$\|\downarrow\downarrow\rangle$	$\frac{J}{4}\|\downarrow\downarrow\rangle$

```
def zprod(state):
    ans = []
    spins = state.data
    for site in range(SITE_NUM):
        for bond_site in Sites[site].bond:
            mag = 1
            if(spins[site] == 1):
                mag *= 1/2
            else:
                mag *= -1/2
            if(spins[bond_site] == 1):
                mag *= 1/2
            else:
                mag *= -1/2
            ans.append([state,mag*bond_mod_J])
    return ans
```

第三項(外積)

さっきの表を再現すればよい。下に再掲

もとの状態	遷移後の状態
$\|\uparrow\uparrow\rangle$	0
$\|\uparrow\downarrow\rangle$	$\frac{D_{ij}}{2i}\|\downarrow\uparrow\rangle$
$\|\downarrow\uparrow\rangle$	$-\frac{D_{ij}}{2i}\|\uparrow\downarrow\rangle$
$\|\downarrow\downarrow\rangle$	0

```
def outProd(state):
    ans = []
    spins = state.data
    conv = {(1,0):1j/2,(0,1):-1j/2}
    for site in range(SITE_NUM):
        for bond_ind in range(len(Sites[site].bond)):
            copy_spins = spins.copy()
            if(copy_spins[site] != copy_spins[Sites[site].bond[bond_ind]]):
                copy_spins[site],copy_spins[Sites[site].bond[bond_ind]] = copy_spins[Sites[site].bond[bond_ind]],copy_spins[site]
                modified_state = States[idToInd[spinToId(copy_spins)]]
                ans.append([modified_state,conv[(spins[site],spins[Sites[site].bond[bond_ind]])]*Sites[site].mod[bond_ind]])
    return ans
```

ハミルトニアン

第1項から第3項をまとめるだけ

```
def Hamil(state):
    in_prod_ans = inProd(state)
    out_prod_ans = outProd(state)
    return in_prod_ans + out_prod_ans
#内積
def inProd(state):
    ans = []
    x_ans = xyprod(state)
    ans += x_ans
    z_ans = zprod(state)
    ans += z_ans
    return ans
```