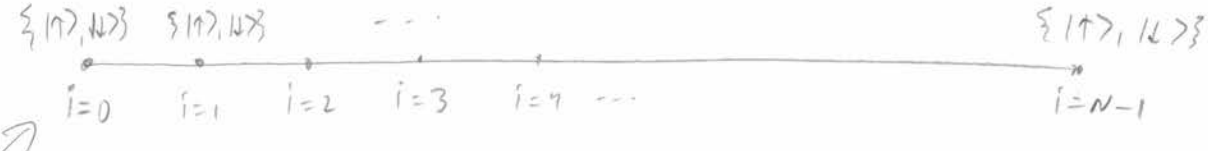
Lecture 2: Solving Ising using Exact Diagonalization

Last time, we reviewed our spin-$\frac{1}{2}$ knowledge, and we carefully thought about how to work with a system of 2 × spin-$\frac{1}{2}$.

Today we will solve the Ising model:

$$H = -J \sum_i S_i^z S_{i+1}^z - h \sum_i S_i^x$$

What does this actually mean?



Chain of spin-$\frac{1}{2}$ particles.

States live in the tensor product space

$\mathcal{H}_0 \otimes \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_N$ w/ basis states like $|\uparrow\uparrow\dots\uparrow\rangle$, $|\uparrow\uparrow\dots\uparrow\downarrow\rangle$, ...

(count from 0 b/c easier for programming)

$|\uparrow\rangle_0 \otimes |\uparrow\rangle_1 \otimes \dots \otimes |\downarrow\rangle_{N-2} \otimes |\uparrow\rangle_{N-1} \rightarrow |\uparrow\uparrow\dots\downarrow\uparrow\rangle$

$|\downarrow\downarrow\dots\downarrow\rangle$

in total there are 2^N different basis states

The Hamiltonian H is an operator that acts on these states:

S_j^x means $Id_0 \otimes Id_1 \otimes Id_2 \otimes \dots \otimes S_j^x \otimes \dots \otimes Id_N$

$\sum_j S_j^x$ means we take the sum of all these operators, with all possible positions j from 0 to N

Similarly $S_j^z \; S_{j+1}^z$ means

$$Id_0 \otimes \cdots \otimes Id_{j-1} \otimes S_j^z \otimes S_{j+1}^z \otimes Id_{j+2} \otimes \cdots \otimes Id_{N-1}$$

Before we do anything else, let's get rid of stupid $\hbar/2$ and $\frac{1}{2}s$:

$$H = -\underbrace{\left(J\frac{\hbar^2}{4}\right)}_{\text{new } J} \sum_i \sigma_i^z \sigma_{i+1}^z - \underbrace{\left(h\frac{\hbar}{2}\right)}_{\text{new } h} \sum_i \sigma_i^x$$

i.e. $$H = -J \sum_i \sigma_i^z \sigma_{i+1}^z + h \sum_i \sigma_i^x$$

We can put $\hbar$ back at the end if we want. But this is nice because now J and h both have the same units: energy.

~~Now, let's be concrete! construct a matrix for H~~

Now, we'll introduce the method of <u>exact diagonalization</u>:

<u>Step 1</u>: Pick a basis for the many-spin Hilbert space

<u>Step 2</u>: Find the matrix of H in this basis

<u>Step 3</u>: Find the eigenstate w/ the lowest energy as a vector

<u>Step 4</u>: Calculate things, like expectation values.

With any model, the first step is to get some conceptual understanding. What do you think should happen?

Here we have "two parameters," but actually only one.

→ H and $2 \times H$ will have the same eigenstates, only change is energies multiplied by 2.

→ so J and h are not really independent, only care about ratio $\frac{h}{J} \equiv g$

→ so look at $$\frac{H}{J} = -\sum_i \sigma_i^z \sigma_{i+1}^z - g \sum_i \sigma_i^x$$

[Warning: if $J<0$, this is bad, eigenstates the same, but which one is GS changes]

Then there are 2 easy limits:

① $g \to 0$: $H = -J \sum_i \sigma_i^z \sigma_{i+1}^z$

Note that all of these terms commute, so can find overall ground state by finding ground state for each pair of spins separately

② $g \to \infty$: $H = -h \sum_i \sigma_i^x$

Again all terms commute, so can find overall GS by finding GS for each spin individually

Exercise: what is the GS in each limit?

Answer: ① 2 spin GS is 2-fold: $|\uparrow\uparrow\rangle$ and $|\downarrow\downarrow\rangle$

look at 2 neighboring terms

1, 2	and	2, 3
$\|\uparrow\uparrow\rangle$		$\|\uparrow\uparrow\rangle$
$\|\downarrow\downarrow\rangle$		$\|\downarrow\downarrow\rangle$

To satisfy both, state of 1, 2, 3 can be
$|\uparrow\uparrow\uparrow\rangle$ or $|\downarrow\downarrow\downarrow\rangle$

Extrapolate: GSs are $|\uparrow_0 \ldots \uparrow_{N-1}\rangle$ and $|\downarrow_0 \ldots \downarrow_{N-1}\rangle$

② Want max σ^x pig state on each state, GS is

$|\rightarrow\rightarrow\rightarrow\ldots\rightarrow\rangle$

Away from these two limits, the model is very hard to solve.

But we can do it numerically!

start with 2 sites:

$$H = -J(\sigma_0^z \sigma_1^z) - h(\sigma_0^x + \sigma_1^x)$$

↓ precise version

$$H = -J(\sigma_0^z \otimes \sigma_1^z) - h(\sigma_0^x \otimes Id_1 + Id_0 \otimes \sigma_1^x)$$

Exercise: (on paper) construct a 4×4 matrix for H in the basis

$\begin{pmatrix}1\\0\\0\\0\end{pmatrix} = |\uparrow\uparrow\rangle$, $\begin{pmatrix}0\\1\\0\\0\end{pmatrix} = |\uparrow\downarrow\rangle$, $\begin{pmatrix}0\\0\\1\\0\end{pmatrix} = |\downarrow\uparrow\rangle$, $\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = |\downarrow\downarrow\rangle$

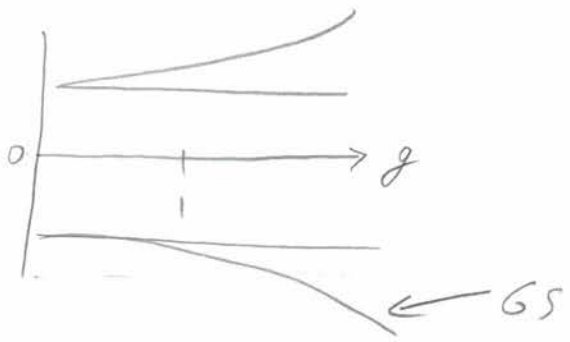
You can do this with the Kronecker product method or by applying H to each basis state to get the matrix cols.

Answer: $-J\begin{pmatrix}1&&&\\&-1&&\\&&-1&\\&&&1\end{pmatrix} - h\left(\begin{pmatrix}1&&&\\&1&&\\&&-1&\\&&&-1\end{pmatrix} + \begin{pmatrix}1&&&\\&-1&&\\&&1&\\&&&-1\end{pmatrix}\right) = -\begin{pmatrix}J&h&h&0\\h&-J&0&h\\h&0&-J&h\\0&h&h&J\end{pmatrix}$

Although in general 4×4 matrices cannot be solved analytically, this one can be.

Using Mathematica, result is

Energies (for $\frac{H}{J}$)



Ground state is

$$\begin{pmatrix} 1 \\ \frac{-1+\sqrt{4g^2+1}}{2g} \\ \frac{-1+\sqrt{4g^2+1}}{2g} \\ 1 \end{pmatrix} \quad / \text{ normalization}$$

Let's check our limits: $g=0$: $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}/\sqrt{2} = \frac{|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle}{\sqrt{2}}$

spins aligned along z, yes!

$g \to \infty$: $\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}/2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}/\sqrt{2} \otimes \begin{pmatrix} 1 \\ 1 \end{pmatrix}/\sqrt{2} = |\rightarrow\rightarrow\rangle$

spins along x, yes!

Away from these limits, some expectation values:

all 2, 3, 5

It was nice that in this case the calculation could be done exactly, but let's pretend it couldn't be and use numerics. We're all going to do some python programming together.

Programming exercise, see ipynb

To go beyond this 2 site limit, it will quickly become too hard to make H by hand. So we have to write a program not just to solve H but also to build it in the first place.

↑ it will be $2^N \times 2^N$ → gets big fast!

For 2 sites, we used a function

```
def H(g):
    return ____________
```

For the general case we don't actually want to do this. With ~15 sites, generating H could take hours each time. We only want to do this work once, but fortunately we can do that:

① generate matrix for $-\sum_i \sigma_i^z \sigma_{i+1}^z \equiv ZZ$

② generate matrix for $-\sum_i \sigma_i^x \equiv X$

Then we store these two, for each g, just compute

$ZZ + g \cdot X$, takes ≤ 1 second

What can we measure in the ground state?

→ measure tendencies that are in H

$$H \sim ZZ + X$$

ZZ: makes spins point parallel to each other (along Z) ← important b/c Z and X don't commute.

X: makes spins point along X

We can measure tendency to point in X by $\frac{\langle \sigma_1^x \rangle + \langle \sigma_2^x \rangle}{2}$

We can measure tendency to be // along Z by

$$\langle \sigma_1^z \sigma_2^z \rangle$$

But can we do something easier?

if spins are $\uparrow\uparrow\uparrow\uparrow\ldots\uparrow$ (all parallel and $\uparrow$),

maybe we can just do $\frac{\langle \sigma_1^z \rangle + \langle \sigma_2^z \rangle}{2}$ instead?

No look at $|\varphi\rangle = \frac{|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle}{\sqrt{2}}$

$$\langle \sigma_1^z \rangle = \frac{1}{2}\Big(\langle\uparrow\uparrow|\sigma_1^z \otimes Id|\uparrow\uparrow\rangle + \langle\downarrow\downarrow|\sigma_1^z\otimes Id|\downarrow\downarrow\rangle + \underbrace{\langle\uparrow\uparrow|\sigma_1^z\otimes Id|\downarrow\downarrow\rangle}_{0} + \underbrace{\langle\downarrow\downarrow|\sigma_1^z\otimes Id|\uparrow\uparrow\rangle}_{0}\Big)$$

$$= \frac{1}{2}\Big(\langle\uparrow|\sigma_1^z|\uparrow\rangle + \langle\downarrow|\sigma_1^z|\downarrow\rangle\Big) = \frac{1}{2}(1-1) = 0$$

But with $\sigma_1^z\sigma_2^z$ this doesn't happen:

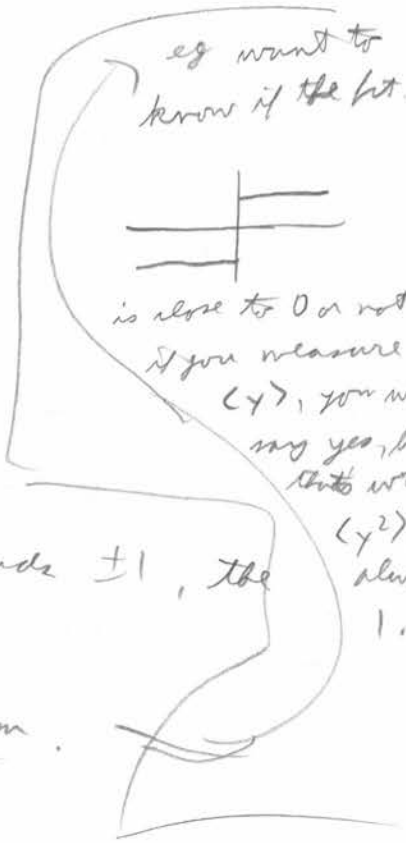
$$\langle \sigma_1^z\sigma_2^z\rangle = \frac{1}{2}\Big(\langle\uparrow\uparrow|\sigma_1^z\sigma_2^z|\uparrow\uparrow\rangle + \langle\downarrow\downarrow|\sigma_1^z\sigma_2^z|\downarrow\downarrow\rangle + 0 + 0\Big)$$

$$= \frac{1}{2}\Big(\langle\uparrow\uparrow|\uparrow\uparrow\rangle + \langle\downarrow\downarrow|\downarrow\downarrow\rangle\Big) = \frac{1}{2}(2) = 1 \quad \checkmark$$

This expectation value is called a correlation function because it measures the extent to which the two operators behave the same way.

Usually $\langle \sigma_1^z \sigma_2^z \rangle \approx \langle \sigma_1^z \rangle \cdot \langle \sigma_2^z \rangle$, but a state like

$$\frac{|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle}{\sqrt{2}}$$ is an exception.



This is because it's made of 2 parts:

$|\uparrow\uparrow\rangle \rightarrow \langle \sigma_1^z \rangle = +1 \;,\; \langle \sigma_2^z \rangle = +1$

$|\downarrow\downarrow\rangle \rightarrow \langle \sigma_1^z \rangle = -1 \;,\; \langle \sigma_2^z \rangle = -1$

So even though there is a tendency towards ± 1, the average is 0.

The correlation function fixes this problem.

This happened because in H there is a symmetry:

If you swap $|\uparrow\rangle \leftrightarrow |\downarrow\rangle$, then

$$\sigma^x \rightarrow \sigma^x$$

$$\sigma^z \rightarrow -\sigma^z \;, \text{ so } H \rightarrow H$$

If you swap $|\rightarrow\rangle \leftrightarrow |\leftarrow\rangle$, this is not the case, so for σ^x, $\langle \sigma^x \rangle$ is ok, no need for $\langle \sigma^x \sigma^x \rangle$.

So the right thing to calculate in this kind of symmetric Hamiltonian is $\langle \sigma^z \sigma^z \rangle$.

Our next exercise will be to write a program to generate these matrices ZZ and X

Open doing-general.ipynb

You see that there are 2 functions we need to fill in:

```
def ZZ(N):

def X(N):
```

N is the number of sites.

We first pick our basis, which will be the standard one. There are 2^N elements:

$$\begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix} = |\uparrow\ldots\uparrow\rangle, \quad \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix} = |\uparrow\ldots\uparrow\downarrow\rangle, \quad \ldots, \quad \begin{pmatrix}0\\\vdots\\0\\1\\0\end{pmatrix} = |\downarrow\ldots\downarrow\rangle$$

(indices 0, 1, 2, ..., 2^N-1)

In this basis, we will construct ZZ together, in 3 different ways.

1 Kronecker product.

This is the easiest to program.

$A \otimes B \to \begin{pmatrix} a_{11}\cdot B & a_{12}\cdot B & \cdots \\ a_{21}\cdot B & \cdots & \\ \vdots & & \end{pmatrix}$ is built into numpy as np.kron

So for 2 sites, get $S^z \otimes S^z$ by np.kron(Sz, Sz)

Let's do 3 sites

Here the sum has 2 terms:

$$\sigma_0^z \otimes \sigma_1^z \otimes Id_2 \;+\; Id_0 \otimes \sigma_1^z \otimes \sigma_2^z$$

Each term can be computed independently with np.kron, and the results can be added.

It's important to get the order right, so be careful:

$$\sigma_0^z \otimes \sigma_1^z \otimes Id_2 = \left(\sigma_0^z \otimes \sigma_1^z\right) \otimes Id_2$$

$$\downarrow$$

$$np.kron(sz, sz) \otimes Id_2$$

$$\downarrow$$

$$np.kron(np.kron(sz, sz), Id)$$

So $ZZ = np.kron(np.kron(sz, sz), Id) + np.kron(np.kron(Id, sz), sz)$

But how do we instruct our program to do this when there are n sites (n-1 terms in the sum)?

① Create a matrix of zeros that each term will get added to.

$$ZZ = np.zeros((2**N, 2**N)) \leftarrow 2^N \times 2^N$$

② Iterate through $N-1$ terms:

```
for i in range(N-1):
```

③ Make a list of the operators that appears on each site in this term:

$$[\underbrace{Id, Id, \ldots Id}_{i \text{ terms}}] + [sz, sz] + [\underbrace{Id, Id, \ldots, Id}_{N-2-i \text{ terms}}]$$

↑ concatenate lists with "+"

↑ This can be written as $[Id] \cdot i$, so we get $terms = [Id] \cdot i + [sz, sz] + [Id] \cdot (N-2-i$

for other terms, only this part will be different

(4) Iterate through the list with Kronecker product:

```
term = list[0]
for op in list[1:]:
    term = np.kron(term, op)
```

(5) add result to ZZ:

```
ZZ += term
```

---

2 Column-by-column

Find how ZZ acts on each basis vector.

Steps: (1) $\begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix} \to |\uparrow\uparrow\uparrow\cdots\uparrow\rangle$, etc

(2) Act ZZ on $|\uparrow\uparrow\cdots\uparrow\rangle$, see the result

(3) Write as a vector again, put in 1^{st} col.

---

(1) The basis vector has a 1 in position $i \in \{0, \ldots, 2^N - 1\}$

If $i \in \underbrace{\{0, \ldots, 2^{N-1} - 1\}}_{\text{first half}}$, the first spin is $\uparrow$

If $i \in \{2^{N-1}, \ldots, 2^N - 1\}$, first spin is $\downarrow$

So first spin is $i // 2^{N-1}$ ← "integer division", divide then throw away the remainder

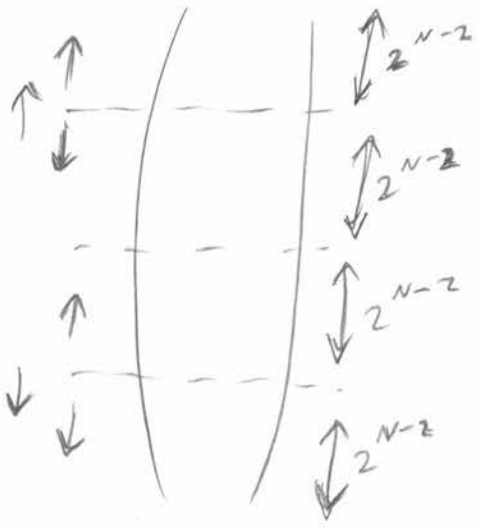
eg $5//3 = 1$, $6//3 = 2$, $7//3 = 2$, ...

So $i // 2^{N-1}$ is 0 if $i \in \{0, \ldots, 2^{N-1} - 1\}$, else 1

Then let $0 \to |\uparrow\rangle$

$1 \to |\downarrow\rangle$

The second spin look at $\frac{1}{4}$ the



for $i // 2^{N-2}$ gives 0

1

2

3

Then we need $0, 2 \longrightarrow \uparrow$

$1, 3 \longrightarrow \downarrow$

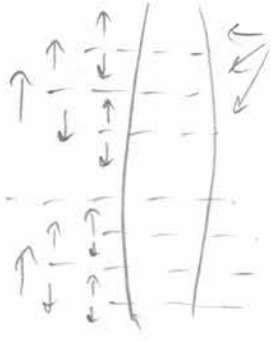
We can do this with <u>mod 2</u> : $0, 2 \to 0$

$1, 3 \to 1$

for $(i // 2^{N-2}) \,\%\, 2 = 0 \Rightarrow \uparrow$

$(i // 2^{N-2}) \,\%\, 2 = 1 \Rightarrow \downarrow$

<u>Third spin</u> :



for $i // 2^{N-3}$ gives 0, 1, 2, 3, 4, 5, 6, 7,

all are spin down [odd]

all are spin ↑ [even]

$\Rightarrow (i // 2^{N-3}) \,\%\, 2 = 0 \Rightarrow \uparrow$

$(i // 2^{N-3}) \,\%\, 2 = 1 \Rightarrow \downarrow$

Now you can see the pattern

basis vector i corresponds to

$$\left[i//2^{N-1} \,,\, (i//2^{N-2})\%2 \,,\, (i//2^{N-3})\%2 \,,\, \ldots \,,\, (i//2^{N-N})\%2 \right]$$

$$\downarrow \quad \begin{matrix} 0 \to \uparrow \\ 1 \to \downarrow \end{matrix}$$

eg $N = 3$

$i=0 \to [0,0,0] \to [\uparrow,\uparrow,\uparrow]$

$i=1 \to [0,0,1] \to [\uparrow,\uparrow,\downarrow]$

$i=2 \to [0,1,0] \to [\uparrow,\downarrow,\uparrow]$

$i=3 \to [0,1,1]$

$\vdots$

It's actually just writing out i in binary!!

function:

```
def d2b(i, N):        # decimal to binary
    b = np.zeros(N)
    for j in range(N):
        b[j] = (i//2**(N-1-j))%2
    return b
```

or

```
def d2b(i, N):
    b = [(i//2**(N-1-j))%2 for j in range(N)]   <- list comprehension
    return np.array(b)
```

Can also represent arrows with $0 \to$ 'u', $1 \to$ 'd', or $0 \to (\uparrow)$, $1 \to (\downarrow)$, etc, but not necessary.

② act with operator:

$$-\sum_i \sigma_i^z \sigma_i^z \; |0,1,0,0,1,\ldots,0\rangle$$

For each term transform it like

$|0,1,0,0\rangle$ $(N=4)$

↓

$$-\left(-|0,1,0,0\rangle - |0,1,0,0\rangle + |0,1,0,0\rangle\right) = +|0,1,0,0\rangle$$

$\sigma^z|0\rangle = |0\rangle$ $\sigma^z|1\rangle = -|1\rangle$ σ^z σ^z σ^z σ^z

As a better example, act with $\sigma_0^x + \sigma_1^z$ on $|0,1\rangle$ $(N=2)$

$$(\sigma_0^x + \sigma_1^z)\,|0,1\rangle = |1,1\rangle + -|0,1\rangle$$

In program:

basis_states = [] ← empty list

coeffs = [] ← empty list

We will come back to this shortly.

③ reverse step 1, from $[0,1,0,0] \to i=4 \to$ 

We follow the same idea in reverse

$[0,1,0,0]$

$\times 2^{N-1}$ $\times 2^{N-2}$ $\times 2^{N-3}$ ... $\cdot 1 = 2^{N-N}$ then sum

i.e.

```
def b2d(basis_state, N):    (note: don't actually need N)
    N = len(basis_state)
    d = np.sum(basis_state[i] * 2**(N-1-i) for i in range(N))
    return d
```

Let's put it together:

"col" is the index for a basis vector → because that determine col in ZZ

We will fill column "col" of matrix ZZ

```
b = d2b(col, N)
for j in range(N-1):        ← iterate through terms
    coeff = (-1)**(b[j]*b[j+1])   ← gives -1 if 0,1 or 1,0,
                                    +1 if 0,0 or 1,1
    new_b = b
    row = b2d(new_b)        ← get row of matrix from index
    ZZ[row,col] += coeff
```

(this is the part where it matters what the operator is. for other terms, only these two lines will be different)

operator does not change the basis state. σ_X however would add effect of this term to the matrix

This gets put inside the function

```
def ZZ_v2(N):
    ZZ = np.zeros((2**N, 2**N))
    for col in range(2**N):
        ...
    return ZZ
```

---

[3] Using our knowledge, in this case that ZZ is diagonal in our basis. Since all off-diag elts are 0, just find diagonal ones, and for each basis state it is given by $\langle b|ZZ|b\rangle$.

$$= -\sum_i \langle b|\sigma_i^z \sigma_{i+1}^z|b\rangle$$

This calculation is quite easy!

index i

↓

binary (basis state)

↓ find expectation value

put in a length $2^{**}N$ vector

↓

make into diagonal matrix

```
def ZZ_v3(N):
    ZZ_diag = np.zeros(2**N)
    for i in range(2**N):
        b = d2b(i,N)
        val = 0
        for j in range(N-1):
            val += (-1)**(b[j]+b[j+1])
        ZZ_diag[i] = -val
    ZZ = np.diag(ZZ_diag)
    return ZZ
```

---

eg for N=4, i=7: [0,1,1,1]

val = −1 + 1 + 1 = 1

Can make this even more efficient:

eg [0,1,1,1]

[0,1,1,1] → shift ⇒ [0,1,1,1] added to [0,1,1,1] = [1,2,2]

$(-1)^{**}$[1,2,2] = [−1,1,1] then sum

ie `val = np.sum((-1)**(b[1:] + b[:-1]))`

for loops in python are slow, so this below will speed

Exercise: Your job is now to reproduce this but for the X term.

Since it is not diagonal, can't use [3].

But you should write programs in the style of both [1] and [2].

Answer: You should check at the end that your programs make the following for $N=2$, $N=3$

$N=2$

$$-\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$N=3$

$$-\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{matrix}$$

- There is a 1 if the basis states differ by a spin flip, otherwise a 0.
- Note the $N=2$ matrix appears in the 4×4 diagonal blocks. That's because spin 0 is fixed in those blocks, so they show what happens to the $N=2$ subsystem of spins 1 and 2.

This can be used to construct that matrix, but it's confusing.

A good weekend exercise if you think it sounds fun.

eg figure out where the 0,1 block is located (spread out), fill in $N=2$ case there also, etc.

if just $N=2$ blocks like $\left(\begin{array}{c|c} 4 & \\ \hline & 12 \end{array}\right)$

Then multiply by change of basis matrix for swapping 0 & 1,
do it again

Then swap 0 & 2, do it again.