$$\beta = \frac{1}{kT} \qquad T = \frac{1}{k_B \beta}$$

① 1) $F = U - TS$

$$U = \sum E_i P(E_i) = \sum E_i \frac{e^{-\beta E_i}}{Z} = -\partial_\beta \ln Z$$

$$S = -\sum P(E_i) \ln P(E_i) = -\sum \frac{e^{-\beta E_i}}{Z} \ln \frac{e^{-\beta E_i}}{Z}$$

$$d \ln Z = \frac{\partial \ln Z}{\partial \beta} d\beta + \frac{\partial \ln Z}{\partial \mu} d\mu = -U d\beta + \mu \beta d\mu$$

$$n = \sum P(E_i) n = \frac{1}{\beta} \frac{\partial \ln Z}{\partial \mu}$$

$$d(\beta U) = d\beta U + U d\beta \rightarrow U d\beta = d(\beta U) - U d\beta$$

$$d \ln Z = -d(\beta U) + U d\beta + \mu \beta d\mu$$

$$\Rightarrow dU = \frac{1}{\beta} d \ln Z + \frac{1}{\beta} d(\beta U) - \mu d\mu$$

$$= \frac{1}{\beta} d(\ln Z + \beta U) - \mu d\mu$$

bzw in $N \rightarrow \infty$ $\qquad dU = T dS + \mu d\mu$

$$\Rightarrow S = k_B \ln Z + \frac{U}{T} \rightarrow F = U - TS$$

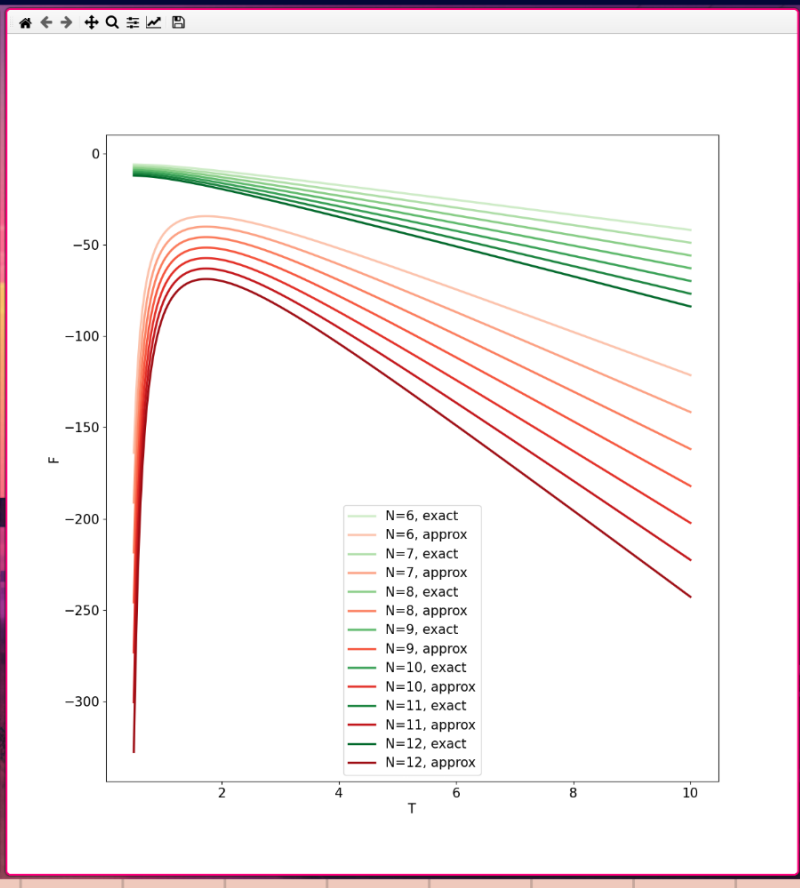
$\Rightarrow F = \cancel{U} - k_B T \ln Z + \cancel{U}$

Yazi: ~/Physics/Gra… 1% 25% 40°C English (US) Alon (84%) 10% 59% 25-11 18:52

ex1q1.py

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import colors
plt.rcParams["font.size"] = 15

T = np.linspace(5e-01, 10, 1000)
norm = colors.Normalize(vmin=0, vmax=9)
cmap1 = cm.get_cmap('Greens', 10)
cmap2 = cm.get_cmap('Reds', 10)

for N in range(6, 13):
    Fexact = -T * N * np.log(2 * np.cosh(1 / T))
    L = np.exp(1 / T) * np.cosh(1 / T) + np.sqrt(
        np.exp(2 / T) * (np.cosh(1 / T)) ** 2 - 2 * np.sinh(2 / T)
    )
    Faprox = -T * N * L

    plt.plot(T, Fexact, lw=2.5, label=f"N={N}, exact", color=cmap1(nor
m(N-4)))
    plt.plot(T, Faprox, lw=2.5, label=f"N={N}, approx", color=cmap2(no
rm(N-4)))

plt.ylabel("F")
plt.xlabel("T")
plt.legend(loc="best")
plt.show()
```

NORMAL main ex1q1.py utf-8 python 80% 20:76

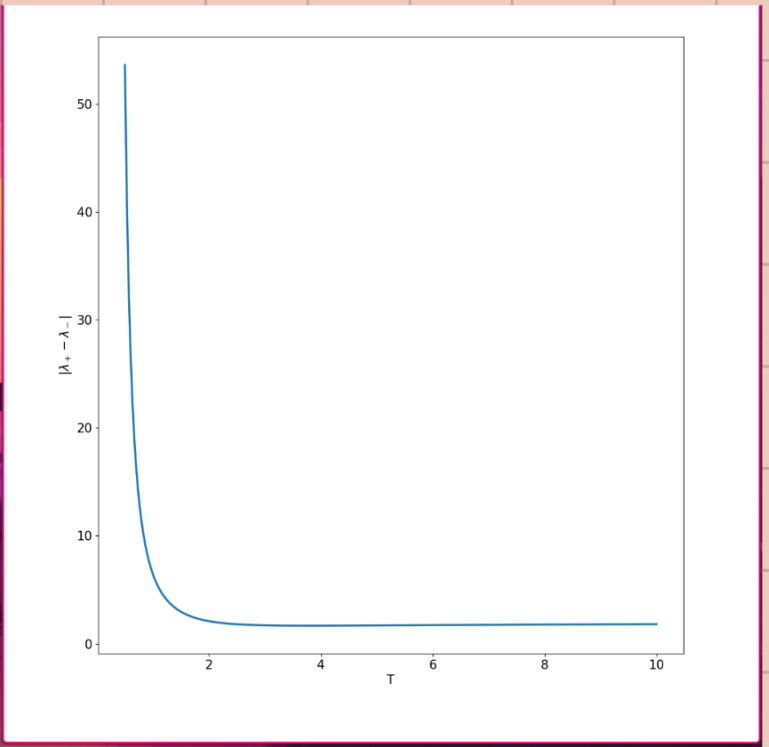


The approximation fails at $T=0$.

AT $T=0$ both $\lambda_\pm$ should have the same value cuz there's only one state for the system.

But clearly this is not the case. AT $T=0$

$\lambda_+ - \lambda_- \xrightarrow{T \to 0} \infty$



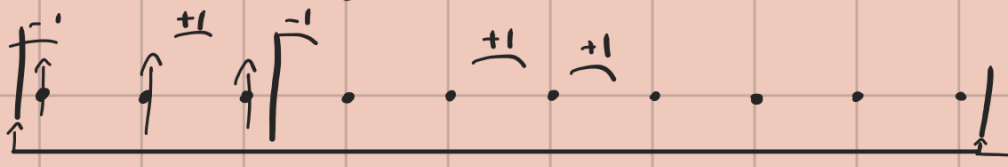
(2)

$$Z = \sum_{\mu\text{-states}} e^{-\beta H}$$

$$H = -J \sum_i \sigma_i \sigma_{i+1}$$

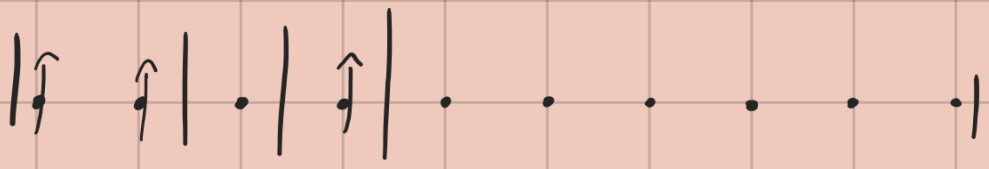
(periodic boundry condition $\sigma_{N+1} \equiv \sigma_1$)

Its easier to count domain walls then options.



2 domain walls

same domain wall



4 domain walls

6 domain walls

number of options: $\begin{pmatrix} \text{number of domain walls} \\ \text{number of positions} \end{pmatrix} = \binom{2}{10}, \binom{4}{10}, \binom{6}{10}$

$= 45 \quad = 210 \quad = 210$

Tot: $10J$ each domain wall: $-J$

$$Z = \left(45\, e^{8\beta J} + 210\, e^{6\beta J} + 210\, e^{4\beta J}\right) \cdot 2$$

← for spin inversion

(3) 1) $1 - S^2 = \begin{cases} 0 & S=-1 \\ 1 & S=0 \\ 0 & S=1 \end{cases}$

if the spin is in an excited state what so ever

2) $e^{\beta J S_1 S_2 + \frac{\beta B}{2}(S_1+S_2) + \frac{\beta\lambda}{2}(2 - S_1^2 - S_2^2)}$ ↪ Symetric

$S_1=-1 \quad S_2=-1 \;:\; e^{\beta J + 2\beta B - \beta\lambda}$

$=-1 \quad =0 \;:\; e^{-\beta B}$

$=-1 \quad =1 \;:\; e^{-\beta J - \beta\lambda}$

	$S_1=-1$	$S_1=0$	$S_1=1$
$S_2=-1$	$e^{\beta J - \beta B}$	$e^{-\frac{1}{2}\beta B + \frac{1}{2}\beta\lambda}$	$e^{-\beta J}$
$S_2=0$	$e^{-\frac{1}{2}\beta B + \frac{1}{2}\beta\lambda}$	$e^{+2\beta\lambda}$	$e^{+\frac{1}{2}\beta B + \frac{1}{2}\beta\lambda}$
$S_3=1$	$e^{-\beta J}$	$e^{+\frac{1}{2}\beta B + \frac{1}{2}\beta\lambda}$	$e^{\beta J + \beta B}$

$1 - \frac{D}{J}$ $\quad -\frac{1}{2}\frac{D}{J}+1$ $\quad +\frac{1}{2}\frac{D}{J}+1$ $\quad 1+\frac{D}{J}$

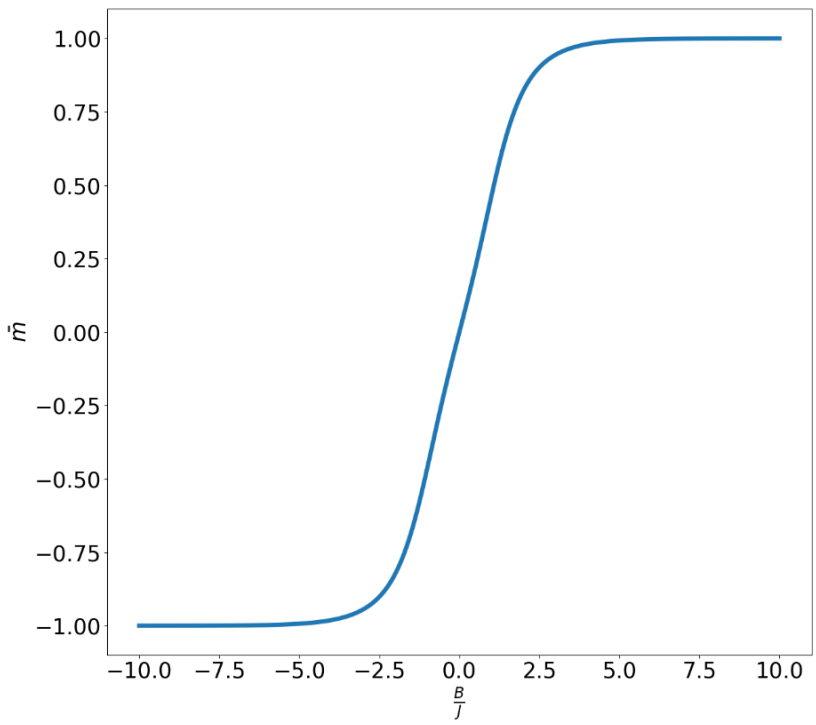
$\lambda = 2J \qquad \beta = \frac{1}{J}$

3) The eigen values + vectors are monsterous.

Anyway $z = \lambda^N_{largest}$ from wolfram

4) $\bar{m} = \frac{1}{N}\frac{\partial f}{\partial B} = \frac{1}{N}\frac{\partial \ln z}{\partial B} = \beta \frac{\partial \ln \lambda}{\partial B}$

Figure 1



ex1.py

```python
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["font.size"] = 25

l = 250
B = np.linspace(-10, 10, l)
array = np.zeros(l)

for i, b in np.ndenumerate(B):

    TM = np.array(
        [
            [np.e ** (1 - b), np.e ** (1 - 0.5 * b), np.e ** (-1)],
            [np.e ** (1 - 0.5 * b), np.e ** (2), np.e ** (1 + 0.5 * b)],
            [np.e ** (-1), np.e ** (1 + 0.5 * b), np.e ** (1 + b)],
        ]
    )
    eigv, eigvec = np.linalg.eig(TM)

    array[i] = np.log(eigv.max())

grad = np.gradient(array, B)

plt.plot(B, grad, lw=5)
plt.ylabel(r"$\bar{m}$")
plt.xlabel(r"$\frac{B}{J}$")
plt.show()
```

W E741 ambiguous variable name 'l'

NORMAL main ⚠ 1 ex1.py utf-8 python 81% 22:27

(4) NN + NNN

we don't need: $\langle \sigma_1\sigma_2\sigma_3 | M | \sigma_1\sigma_2\sigma_3\rangle\langle \sigma_1\sigma_2\sigma_3 | M | \sigma_2\sigma_3\sigma_4\rangle \dots$

this is 8 basis vectors, one for each combination

we can make due with 4 basis vectors:

$\langle \sigma_1\sigma_2 | M | \sigma_1\sigma_2\rangle\langle \sigma_1\sigma_2 | M | \sigma_2\sigma_3\rangle\langle \sigma_2\sigma_3 | M | \sigma_3\sigma_4\rangle \dots$

but this is still more free variables than we need. 4x4 matrix is 16 elements, we have 8 combinations which is still less than 10 free elements for a symmetric 4x4 matrix => this will not be a symmetric matrix!

$$H = \frac{J_1}{2}(\sigma_1\sigma_2 + \sigma_2\sigma_3) + J_2\sigma_1\sigma_3 + \frac{B}{3}(\sigma_1+\sigma_2+\sigma_3)$$

the Hamiltonian knows about 3 spins at a time: $\langle \sigma_1\sigma_2 | M | \sigma_2\sigma_3\rangle$

```
❯ python ex1q4.py
```

$$\begin{bmatrix} e^{-\beta\cdot(-B - 2\cdot J_1 - J_2)} & e^{-\beta\cdot\left(-\frac{B}{3} + J_2\right)} & e^{-\beta\cdot(-B - J_2)} & e^{-\beta\cdot\left(-\frac{B}{3} - 2\cdot J_1 + J_2\right)} \\ e^{-\beta\cdot\left(-\frac{B}{3} - J_2\right)} & e^{-\beta\cdot\left(\frac{B}{3} + 2\cdot J_1 + J_2\right)} & e^{-\beta\cdot\left(-\frac{B}{3} + 2\cdot J_1 - J_2\right)} & e^{-\beta\cdot\left(\frac{B}{3} + J_2\right)} \\ e^{-\beta\cdot\left(-\frac{B}{3} + J_2\right)} & e^{-\beta\cdot\left(\frac{B}{3} + 2\cdot J_1 - J_2\right)} & e^{-\beta\cdot\left(-\frac{B}{3} + 2\cdot J_1 + J_2\right)} & e^{-\beta\cdot\left(\frac{B}{3} - J_2\right)} \\ e^{-\beta\cdot\left(\frac{B}{3} - 2\cdot J_1 + J_2\right)} & e^{-\beta\cdot(B - J_2)} & e^{-\beta\cdot\left(\frac{B}{3} + J_2\right)} & e^{-\beta\cdot(B - 2\cdot J_1 - J_2)} \end{bmatrix}$$

```
~/Physics/Grad/StatMech1 main* ❯
```

We need to diagonalize it and then $Z = \tilde{\lambda}_{max}$

ex1q4.py

```python
import itertools
from sympy import symbols, exp, Matrix, pprint

# Define symbolic parameters
beta, J1, J2, B = symbols("beta J1 J2 B")
spin_values = [-1, 1]  # Possible spin values

# Generate all configurations of three spins
configurations = list(itertools.product(spin_values, repeat=2))

# Initialize the transfer matrix
n_configs = len(configurations)
T = Matrix.zeros(n_configs, n_configs)

# Compute each element of the transfer matrix
for i, C in enumerate(configurations):
    for j, C_prime in enumerate(configurations):
        # Unpack spins for C and C_prime
        s_i, s_ip1 = C
        s_ip1p, s_ip2 = C_prime

        # Hamiltonian between these configurations
        H = (
            - J1 * (s_i * s_ip1 + s_ip1p * s_ip2)
            - J2 * (s_i * s_ip2)
            + (B / 3) * (s_i + s_ip1 + s_ip2)
        )

        # Transfer matrix element
        T[i, j] = exp(-beta * H)

# Print the symbolic transfer matrix
pprint(T)
```

NORMAL main ex1q4.py