Spin d'Ising

Amaury Maros

10 janvier 2025

Ce document contient les réponses au problème de spin d'Ising dans le cadre de l'UE "Gestion Informatique et Python" du Master 1 Physique à distance de l'Université Aix-Marseille. L'objectif est d'étudier le comportement d'un réseau de spin à deux dimensions à l'aide de la méthode de Monte Carlo en utilisant l'algorithme de Métropolis. Le choix de cette modélisation s'explique par la dimension de notre problème : nous considérons un réseau de dimension 10x10, entrainant 2^{100} configurations possibles. Deux cas de figures seront étudiés : le cas ferromagnétique et le cas antiferromagnétique. Dans le premier cas de figure, les spins sont alignés dans la même direction et contribuent tous à l'aimantation. Dans le second cas, les spins sont régulièrement alternés, on parle d'anti-alignement, et l'aimantation globale est nulle. Deux températures caractérisent ces types de matériaux : la température de Curie pour les matériaux ferromagnétiques et la température de Néel pour les matériaux antiferromagnétiques. On se propose de mettre en avant ces caractéristiques dans la suite de ce document. Les questions du sujet sont traitées dans l'ordre, et le code utilisé dans cette modélisation est disponible en annexe (Notebook Jupyter). Les éléments principaux de codes sont intégrés au rapport.

1 Question 1

Un tableau de dimension 10×10 (100 cellules) rempli aléatoirement avec des valeurs de spins pouvant prendre les valeurs ± 1 est initialisé en utilisant le sous-module `random` du module `numpy` en Python :

```
1 import numpy as np
2 spin_table = np.random.choice([-1, 1], size=(10, 10))
```

2 Question 2

La figure 1 ci-dessous est une représentation graphique de cette table de spin. Les spins +1 sont représentés en blanc tandis que les spins -1 sont représentés en noir.

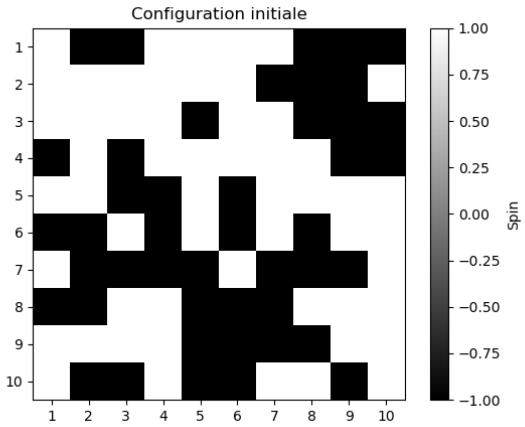


Figure 1: Représentation graphique de la table de spins

3 Question 3

L'énergie de la configuration initiale décrite par cette table de spin est calculée d'après la relation suivante :

$$H = J \sum S_i S_j \tag{1}$$

J traduit l'interaction ferromagnétique (J=1) ou antiferromagnétique (J=-1) des spins avec leurs plus proches voisins. Des conditions aux limites périodiques sont appliquées : les spins situés en bordure du tableau sont considérés comme ayant pour voisins les spins adjacents ainsi que ceux situés sur les bordures opposées. Le calcul de l'énergie se fait au moyen d'une fonction `calculate_energy()` prenant 2 arguments : une table de spin et une valeur de J.

```python
def calculate_energy(spin_table, J):
    # Initialisation
    energy = 0
    rows, cols = spin_table.shape
    for i in range(rows):
        for j in range(cols):
            # Conditions aux limites périodiques
            neighbor_up = spin_table[(i-1) % rows, j]
            neighbor_down = spin_table[(i+1) % rows, j]
            neighbor_left = spin_table[i, (j-1) % cols]
            neighbor_right = spin_table[i, (j+1) % cols]
            # Calcul de l'énergie
            energy += J * spin_table[i, j] * (
                neighbor_up + neighbor_down + neighbor_left + neighbor_right
            )
    # Diviser l'énergie par 2 pour ne considérer qu'une fois chaque couple
    return energy / 2
```

4 Question 4

Le cas ferromagnétique J = 1 est considéré dans un premier temps. L'énergie associée à la configuration initiale vaut E = 28 J.

5 Question 5

L'étude se poursuit par l'application de la méthode de Métropolis à la table de spins de la configuration initiale par l'intermédiaire d'une boucle **for** sur 100000 itérations avec le paramètre T fixé à 0,1 et la valeur du paramètre J fixé à 1. Le temps de calcul est d'environ 10 secondes

```
for _ in range(n_iter):
    # Configuration initiale
    energy_initial = calculate_energy(spin_table, J)
    P_A = np.exp(-energy_initial / T)

    # Choix aléatoire d'un spin
    rows, cols = spin_table.shape
    i = np.random.randint(0, rows)
    j = np.random.randint(0, cols)

    # Modification de la configuration
    random_spin_value = spin_table[i, j]
    spin_table_modified = spin_table.copy()
    spin_table_modified[i, j] = -random_spin_value

    # Calcul nouvelle energie
    new_energy = calculate_energy(spin_table_modified, J)
    P_B = np.exp(-new_energy / T)

    # Calcul de R (attention division par 0)
    if P_B == 0:
        R = np.inf
    else:
        R = P_A / P_B

    # Metropolis
    if R >= 1:
        final_table = spin_table_modified
    else:
        p = random.random()
        if p < R:
            final_table = spin_table_modified
        else:
            final_table = spin_table

    # Mise à jour de la table de spin
    spin_table = final_table

```

6 Question 6

Une comparaison des configurations initiale et finale, respectivement avant et après application de la méthode de Métropolis, est présentée en figure 2.

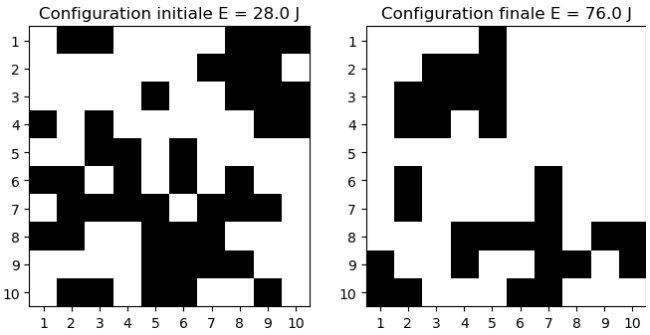


Figure 2: Représentation graphique de la table de spins avant et après la méthode de Métropolis

7 Question 7

La simulation est reconduite avec différentes valeurs de température (figure 3). Une organisation des spins s'opère entre T=0,1 et T=1 : après application de la méthode de Métropolis à basse température, les spins ont tendance à s'orienter dans la direction S=-1 jusqu'à être tous dans cet état à T=1. Lorsque la température continue d'augmnenter, les spins reprennent aléatoirement des valeurs ± 1 : l'ordre est perdu.

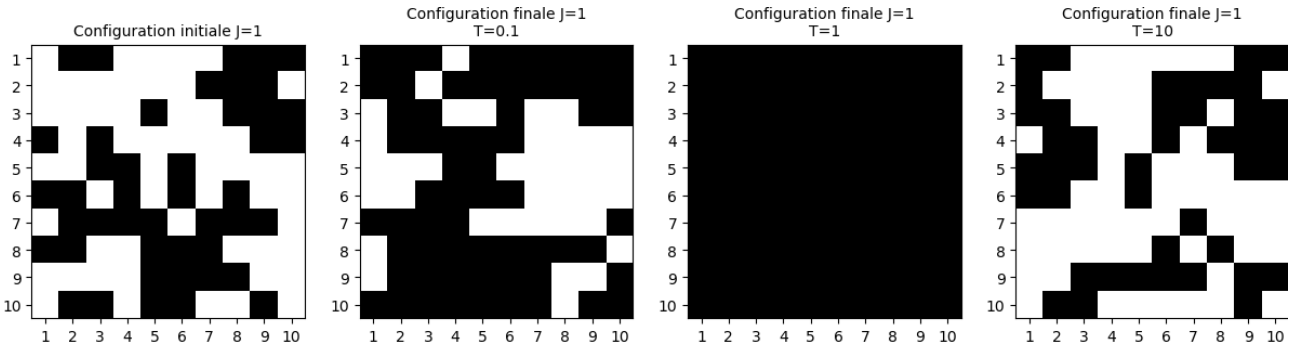


Figure 3: Représentation graphique de la table de spins pour différentes valeurs de température

8 Question 8

La figure 4 représente l'évolution de la magnétisation en fonction de la température ($T \in [0, 1, 10]$). La magnétisation est maximale lorsque les spins sont tous alignés puis elle chute lorsque la température augmente. Elle se stabilise autour de 0 lorsque la température continue d'augmenter. Entre T=1 et T=5, le réseau de spins subit une transition de phase faisant brutalement chuter sa magnétisation : cela se produit à la température de Curie. Au delà de cette température, le réseau de spin perd son caractère ferromagnétique : il devient paramagnétique.

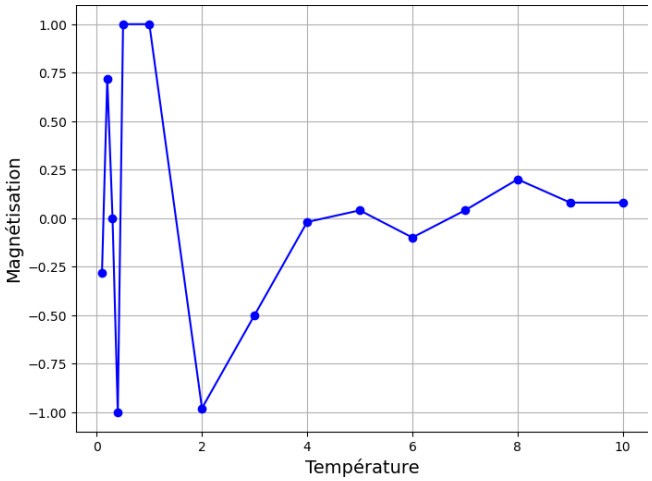
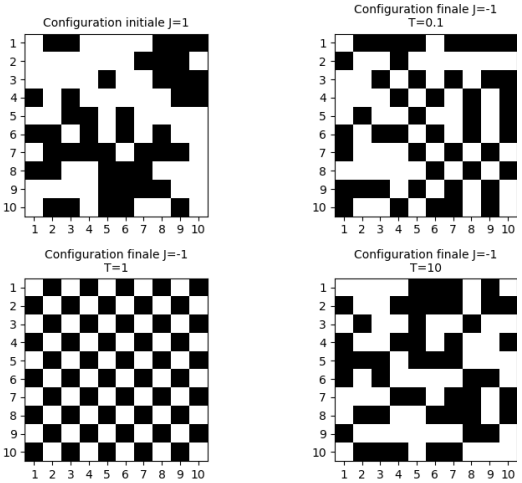


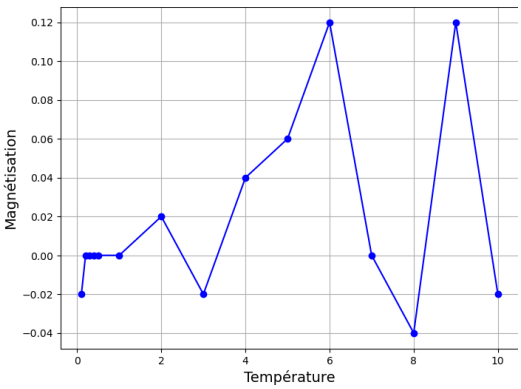
Figure 4: Représentation graphique de la magnétisation pour différentes valeurs de température

9 Question 9

La simulation est reconduite dans le cas antiferromagnétique J=-1. Les résultats sont présentés en figure 5. Les spins adoptent un comportement différents : ils tendent à s'orienter alternativement à basse température jusqu'à être parfaitement alternés lorsque le paramètre T vaut 1. Lorsque la température augmente, l'ordre est de nouveau perdu et les spins sont aléatoirement orientés. La magnétisation est nulle lorsque les spins sont ordonnés et elle reste par la suite proche de 0 lorsque la température augmente ($M_{max} = 0,12 m^{-1} A$). Le réseau adopte un comportement antiferromagnétique jusqu'à T=1. Au dela, l'ordre est perdu et le matériau adopte un comportement paramagnétique : c'est la température de Néel.



(a) Représentation graphique de la table de spins pour différentes valeurs de température



(b) Représentation graphique de la magnétisation pour différentes valeurs de température

Figure 5: Comparaison des configurations et de la magnétisation pour différentes températures. (a) Configuration des spins. (b) Magnétisation.

A Annexe: Notebook Jupyter

Cette annexe contient le code utilisé pour réaliser cet exercice. Il s'agit de l'exportation .pdf du Notebook Jupyter.

DM1

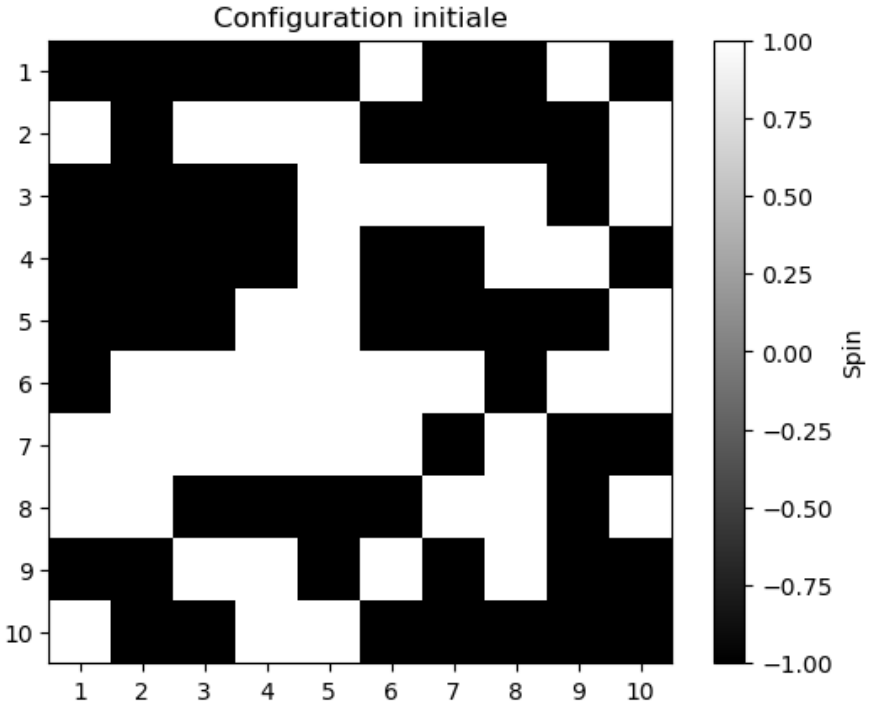
January 9, 2025

[1]:
```
# Packages importation

import numpy as np
import matplotlib.pyplot as plt
import random
```

[2]:
```
# 1. Créer un tableau rempli aléatoirement avec des valeurs de spin.

# np.random.seed(42)
spin_table_initial = np.random.choice([-1, 1], size=(10, 10))
spin_table_initial
```

[2]:
```
array([[-1, -1, -1, -1, -1,  1, -1, -1,  1, -1],
       [ 1, -1,  1,  1,  1, -1, -1, -1, -1,  1],
       [-1, -1, -1, -1,  1,  1,  1,  1, -1,  1],
       [-1, -1, -1, -1,  1, -1, -1,  1,  1, -1],
       [-1, -1, -1,  1,  1, -1, -1, -1, -1,  1],
       [-1,  1,  1,  1,  1,  1,  1, -1,  1,  1],
       [ 1,  1,  1,  1,  1,  1, -1,  1, -1, -1],
       [ 1,  1, -1, -1, -1, -1,  1,  1, -1,  1],
       [-1, -1,  1,  1, -1,  1, -1,  1, -1, -1],
       [ 1, -1, -1,  1,  1, -1, -1, -1, -1, -1]])
```

[3]:
```
# 2. Afficher le tableau sous forme d'image

plt.imshow(spin_table_initial, cmap='gray', interpolation='nearest')
plt.colorbar(label='Spin')
plt.title('Configuration initiale')
plt.xticks(np.arange(spin_table_initial.shape[1]), labels=np.arange(1,␣
 ↪spin_table_initial.shape[1]+1))
plt.yticks(np.arange(spin_table_initial.shape[0]), labels=np.arange(1,␣
 ↪spin_table_initial.shape[0]+1))

plt.savefig('ising_model/table_de_spin_initiale.png', format='png')

plt.show()
```



[4]:
```python
# 3. Calculer l'énergie de la configuration initiale.

def calculate_energy(spin_table, J):
    energy = 0
    rows, cols = spin_table.shape
    for i in range(rows):
        for j in range(cols):
            # Conditions aux bords périodiques
            neighbor_up = spin_table[(i-1) % rows, j]
            neighbor_down = spin_table[(i+1) % rows, j]
            neighbor_left = spin_table[i, (j-1) % cols]
            neighbor_right = spin_table[i, (j+1) % cols]

            # Calcul de l'énergie
            energy += J * spin_table[i, j] * (neighbor_up + neighbor_down +␣
↪neighbor_left + neighbor_right)

    return energy / 2
```

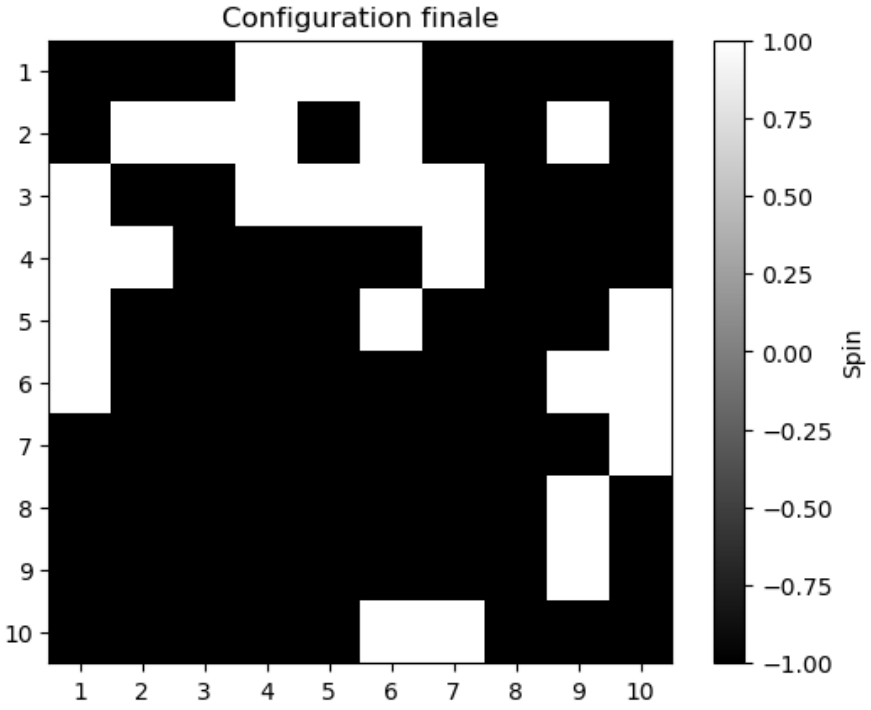
[5]:
```
# 4. On prendra J=1.

energy_initial = calculate_energy(spin_table_initial, J=1)
print("Énergie de la configuration initiale :", energy_initial)
```

```
Énergie de la configuration initiale : 8.0
```

[6]:
```
# 5. Choisir un spin au hasard, et appliquer la méthode de Métropolis. Modifier
 ↪le tableau si
# nécessaire. Réitérer l'opération un grand nombre de fois (100 000 fois).

# Paramètres initiaux
T = 0.1
J = 1
n_iter = 100000

# Méthode de Métropolis
spin_table = spin_table_initial.copy()

for _ in range(n_iter):

    # Configuration initiale
    energy_initial = calculate_energy(spin_table, J)
    P_A = np.exp(-energy_initial / T)

    # Choix aléatoire d'un spin
    rows, cols = spin_table.shape
    i = np.random.randint(0, rows)
    j = np.random.randint(0, cols)

    # Modification de la configuration
    random_spin_value = spin_table[i, j]
    spin_table_modified = spin_table.copy()
    spin_table_modified[i, j] = -random_spin_value

    new_energy = calculate_energy(spin_table_modified, J)
    P_B = np.exp(-new_energy / T)

    # Calcul de R
    if P_B == 0:
        R = np.inf
    else:
        R = P_A / P_B

    # Metropolis
    if R >= 1:
        final_table = spin_table_modified
```

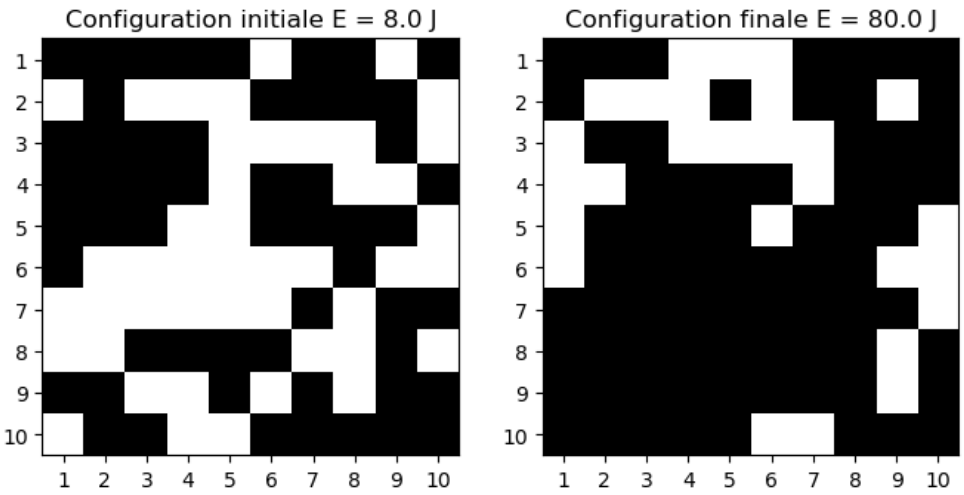
```
    else:
        p = random.random()
        if p < R:
            final_table = spin_table_modified
        else:
            final_table = spin_table

    # Mise à jour de la table de spin
    spin_table = final_table
```

[7]:
```
# 6. Afficher la configuration finale sous fore d'image.

plt.imshow(spin_table, cmap='gray', interpolation='nearest')
plt.colorbar(label='Spin')
plt.title('Configuration finale')
plt.xticks(np.arange(spin_table.shape[1]), labels=np.arange(1, spin_table.
 ↪shape[1]+1))
plt.yticks(np.arange(spin_table.shape[0]), labels=np.arange(1, spin_table.
 ↪shape[0]+1))

plt.savefig('ising_model/table_de_spin_finale.png', format='png',␣
 ↪bbox_inches='tight', pad_inches=0)

plt.show()
```



[8]:
```
# Subplot configuration initiale et configuration finale

plt.figure(figsize=(8,4))

plt.subplot(121)
plt.imshow(spin_table_initial, cmap='gray', interpolation='nearest')
plt.title(f'Configuration initiale E = {calculate_energy(spin_table_initial,␣
 ↪J=1)} J')
plt.xticks(np.arange(spin_table_initial.shape[1]), labels=np.arange(1,␣
 ↪spin_table_initial.shape[1]+1))
plt.yticks(np.arange(spin_table_initial.shape[0]), labels=np.arange(1,␣
 ↪spin_table_initial.shape[0]+1))

plt.subplot(122)
plt.imshow(spin_table, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale E = {calculate_energy(spin_table, J=1)} J')
plt.xticks(np.arange(spin_table.shape[1]), labels=np.arange(1, spin_table.
 ↪shape[1]+1))
plt.yticks(np.arange(spin_table.shape[0]), labels=np.arange(1, spin_table.
 ↪shape[0]+1))
```

```
plt.savefig('ising_model/table_de_spin_initiale_vs_finale.png', format='png',␣
 ↪bbox_inches='tight', pad_inches=0)

plt.show()
```



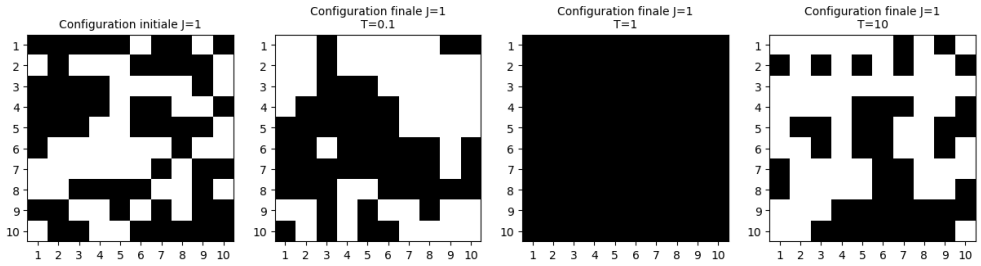
[9]:
```
# 7. On testera différentes valeurs de température : 0.1,1,10. Interprétation.

def metropolis_function(T, J, spin_table):

    # Méthode de Métropolis sur 100000 itérations
    for _ in range(100000):

        # Configuration initiale
        energy_initial = calculate_energy(spin_table, J)
        P_A = np.exp(-energy_initial / T)

        # Choix aléatoire d'un spin
        rows, cols = spin_table.shape
        i = np.random.randint(0, rows)
        j = np.random.randint(0, cols)

        # Modification de la configuration
        random_spin_value = spin_table[i, j]
        spin_table_modified = spin_table.copy()
        spin_table_modified[i, j] = -random_spin_value
```

```
        new_energy = calculate_energy(spin_table_modified, J)
        P_B = np.exp(-new_energy / T)

        # Calcul de R
        if P_B == 0:
            R = np.inf
        else:
            R = P_A / P_B

        # Metropolis
        if R >= 1:
            final_table = spin_table_modified
        else:
            p = random.random()
            if p < R:
                final_table = spin_table_modified
            else:
                final_table = spin_table

        # Mise à jour de la table de spin
        spin_table = final_table

    return spin_table
```

[10]:
```
J = 1
spin_table_ferro = spin_table_initial.copy()
spin_table_ferro_0_1 = metropolis_function(0.1, J, spin_table_ferro)
spin_table_ferro_1 = metropolis_function(1, J, spin_table_ferro)
spin_table_ferro_10 = metropolis_function(10, J, spin_table_ferro)
```

[11]:
```
plt.figure(figsize=(15,10))
plt.subplot(141)
plt.imshow(spin_table_ferro, cmap='gray', interpolation='nearest')
plt.title(f'Configuration initiale J=1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))

plt.subplot(142)
plt.imshow(spin_table_ferro_0_1, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=0.1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))
```

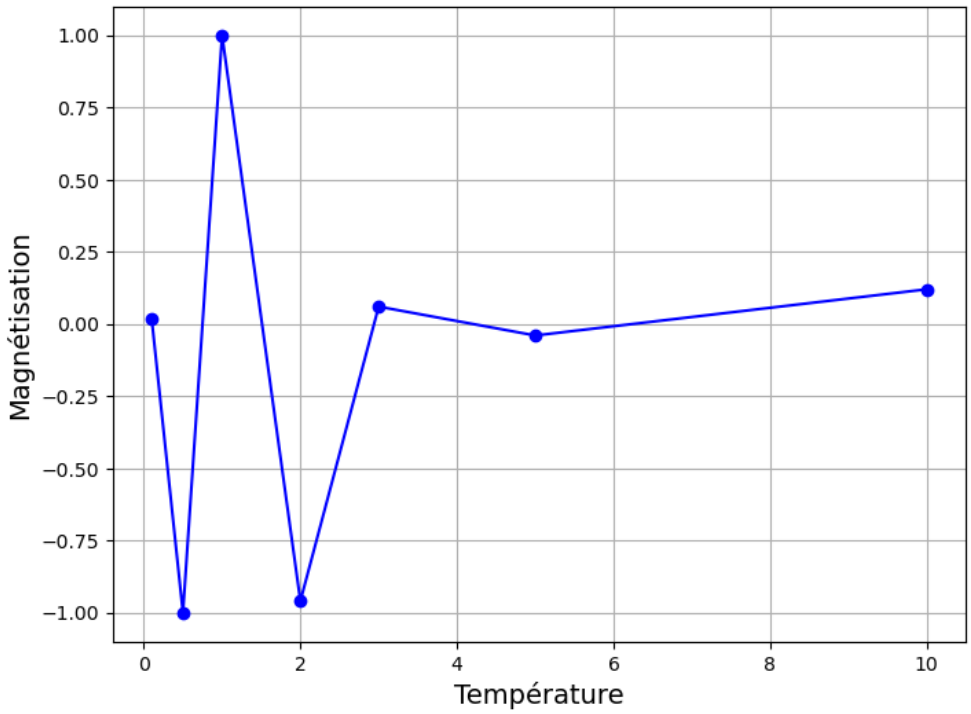
```
plt.subplot(143)
plt.imshow(spin_table_ferro_1, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
 ↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
 ↪spin_table_ferro.shape[0]+1))

plt.subplot(144)
plt.imshow(spin_table_ferro_10, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=10', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
 ↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
 ↪spin_table_ferro.shape[0]+1))

plt.savefig('ising_model/ferromagnetisme_a_differente_T.png', format='png',␣
 ↪bbox_inches='tight', pad_inches=0)

plt.show()
```



[12]:
```
# 8. On calculera aussi la magnétisation

def calculate_magnetisation(spin_table):
    N = spin_table.size
    return np.sum(spin_table) / N

# Plage de températures
temperatures = [0.1, 0.5, 1, 2, 3, 5, 10]

# Calcul de la magnetisation
J = 1
magnetisations_ferro = []

for T in temperatures:
```

```
    spin_table_T = metropolis_function(T, J, spin_table_initial.copy())
    magnetisation = calculate_magnetisation(spin_table_T)
    magnetisations_ferro.append(magnetisation)
```

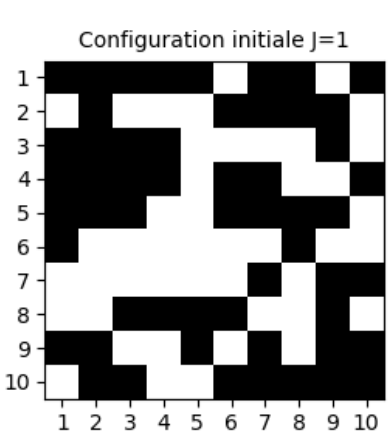
[13]:
```
plt.figure(figsize=(8, 6))
plt.plot(temperatures, magnetisations_ferro, marker='o', linestyle='-',␣
 ↪color='b')
plt.xlabel('Température', fontsize=14)
plt.ylabel('Magnétisation', fontsize=14)
# plt.title('Magnétisation en fonction de la température')
plt.grid(True)

plt.savefig('ising_model/magnetisation_ferro.png', format='png',␣
 ↪bbox_inches='tight', pad_inches=0)

plt.show()
```

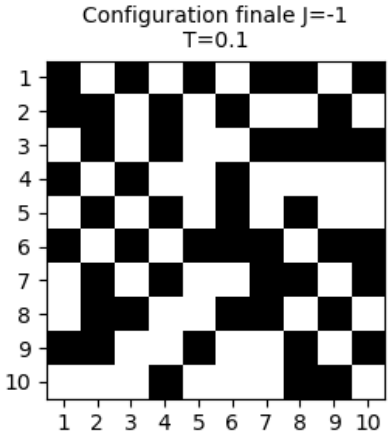


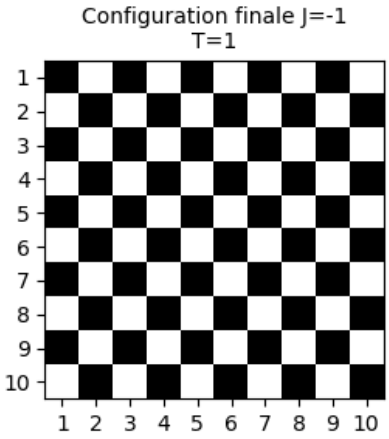
[14]:
```
# 9. On recommencera l'exercice pour J=-1

J = -1
```

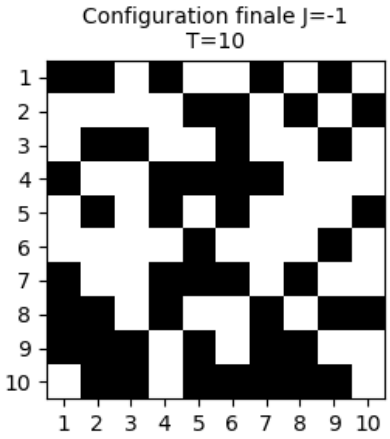
```
spin_table_antiferro = spin_table_initial.copy()
spin_table_antiferro_0_1 = metropolis_function(0.1, J, spin_table_antiferro)
spin_table_antiferro_1 = metropolis_function(1, J, spin_table_antiferro)
spin_table_antiferro_10 = metropolis_function(10, J, spin_table_antiferro)
```

[15]:
```
plt.figure(figsize=(8,6))
plt.subplot(221)
plt.imshow(spin_table_antiferro, cmap='gray', interpolation='nearest')
plt.title(f'Configuration initiale J=1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))

plt.subplot(222)
plt.imshow(spin_table_antiferro_0_1, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=0.1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))

plt.subplot(223)
plt.imshow(spin_table_antiferro_1, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=1', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))

plt.subplot(224)
plt.imshow(spin_table_antiferro_10, cmap='gray', interpolation='nearest')
plt.title(f'Configuration finale J={J}\nT=10', fontsize=10)
plt.xticks(np.arange(spin_table_ferro.shape[1]), labels=np.arange(1,␣
↪spin_table_ferro.shape[1]+1))
plt.yticks(np.arange(spin_table_ferro.shape[0]), labels=np.arange(1,␣
↪spin_table_ferro.shape[0]+1))

plt.tight_layout()
plt.savefig('ising_model/antiferromagnetisme_a_differente_T.png', format='png',␣
↪bbox_inches='tight', pad_inches=0)

plt.show()
```









[16]:
```
# Range of temperatures to test
temperatures = [0.1, 0.5, 1, 2, 3, 5, 10]

# Calculate magnetisation for each temperature
J = -1
magnetisations_antiferro = []

for T in temperatures:
    spin_table_T = metropolis_function(T, J, spin_table_initial.copy())
    magnetisation = calculate_magnetisation(spin_table_T)
    magnetisations_antiferro.append(magnetisation)
```

[17]:
```
# Plot the results
plt.figure(figsize=(8, 6))
```

```
plt.plot(temperatures, magnetisations_antiferro, marker='o', linestyle='-',␣
 ↪color='b')
plt.xlabel('Température', fontsize=14)
plt.ylabel('Magnétisation', fontsize=14)
# plt.title('Magnétisation en fonction de la température')
plt.grid(True)

plt.savefig('ising_model/magnetisation_antiferro.png', format='png',␣
 ↪bbox_inches='tight', pad_inches=0)

plt.show()
```