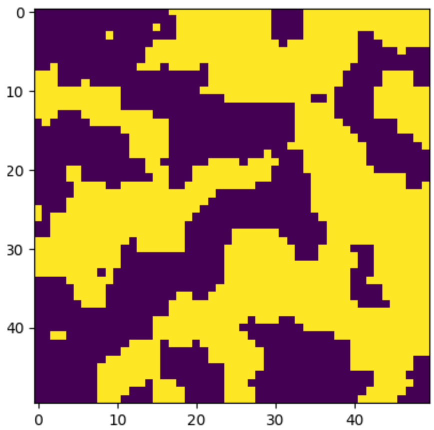
TP : modèle d'Ising
Informatique pour tous



Le modèle d'Ising sert à modéliser le ferromagnétisme (aimants...) en considérant des particules ayant deux états magnétiques (positif ou négatif).
Ces particules seront stockées dans une matrice M remplies de 1 et -1. Chaque élément $M_{i,j}$ de M correspond donc à l'état d'une particule. Deux particules sont voisines si elles correspondent à deux cases adjacentes de M (la particule (i, j) est voisine avec les particules $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$, si celles-ci existent bien dans M).

I Généralités

1. Écrire une fonction `voisin` telle que `voisin(M, i, j)` renvoie la somme des cases voisines de la case `i`, `j` (attention à de pas dépasser de `M`).
   On note $s_{i,j}$ cette valeur dans la suite.
   On pourra commencer le code de la façon suivante:
   ```
   def voisin(M, i, j):
       res = 0
       if i > 0: ...
   ```
2. On définit l'**énergie** du système correspondant à M par $-\frac{1}{2}\sum_{i,j} M_{i,j}s_{i,j}$.

   Écrire une fonction `energie` telle que `energie(M)` calcule cette valeur.
   Vérifier que l'énergie de `M = [[1, -1, 1], [-1, 1, 1]]` est -3.

II Simulation avec la méthode de Monte-Carlo

Écrire `import matplotlib.pyplot as plt` et `import numpy as np`.
On pourra utiliser les fonctions suivantes:

- `np.random.randint(n)`: renvoie un entier entre 0 et $n-1$ uniformément au hasard
- `np.random.random()`: renvoie un flottant uniformément au hasard entre 0 et 1
- `np.random.rand(n, p)` crée une matrice de taille $n \times p$ avec des flottants uniformément au hasard entre 0 et 1

- `np.round(x)`: arrondit un flottant `x` à l'entier le plus proche

1. Définir une matrice `M` de taille 50×50 contenant des 1 et des -1 pris au hasard.
   Vous pouvez afficher graphiquement votre matrice avec `plt.imshow(M)` puis `plt.show()`.

Soit $\beta = \dfrac{1}{k_B T}$, où T est la température et k_B la constante de Boltzmann.
Pour simuler l'évolution d'une configuration de particules, on peut répéter le processus suivant:

- Soit (i, j) une particule au hasard.
- Soit $\Delta E = 2M_{i,j}s_{i,j}$ (c'est la variation d'énergie engendrée par le changement de signe de $M_{i,j}$)
- Soit p un flottant choisit uniformément au hasard entre 0 et 1.
- Si $p < \exp(-\beta \Delta E)$: changer le signe de $M_{i,j}$

2. Écrire une fonction `step(M, B)` effectuant une fois ce processus, où `B` est la valeur de β.
3. Afficher d'abord la matrice obtenue aléatoire avec la question 1, puis (sur une nouvelle figure en écrivant `plt.figure()` puis `plt.imshow(M)`) la même matrice obtenue après 10000 appels à `step`.
   Essayer avec $\beta = 0.8$ (faible température) puis $\beta = 0.2$ (haute température).
4. En utilisant `plt.clf()` (pour effacer la figure) et `plt.pause(0.1)` (pour avoir le temps de voir chaque image), faire une animation de la simulation. On pourra faire 100 appels à `step` à chaque rafraîchissement de la figure, pour éviter que la simulation ne soit trop longue.

III Énumération des niveaux d'énergie

Soit n et p fixés. On voudrait connaître toutes les énergies possibles associées à des matrices de taille $n \times p$.
Pour cela on peut énumérer toutes les matrices $n \times p$ avec que des -1 et 1.
En remplaçant -1 par 0 et en juxtaposant tous les éléments d'une matrice, on voit qu'on peut coder chacune de ces matrices par une suite de 0 et de 1, qui correspond à l'écriture en base 2 d'un certain entier. On va en fait énumérer chaque entier (ce qui est facile avec une boucle `for`) que l'on va ensuite convertir en matrice.

1. Écrire une fonction `list_to_mat` telle que, si `L` est une liste de taille np, `list_to_mat(L, n, p)` renvoie une matrice à n lignes, p colonnes et contenant les mêmes éléments que `L`.
   Par exemple, `list_to_mat([1, 2, 3, 4], 2, 2)` doit renvoyer la matrice `np.array([[1, 2], [3, 4]])`.
2. Écrire une fonction `to_base2(k, p)` renvoyant une liste de taille `p` contenant l'écriture en base 2 de `k` (avec éventuellement des 0 à gauche pour compléter).
3. Écrire une fonction `int_to_mat(k, n, p)` renvoyant la matrice de taille `n`×`p` codé par l'entier `k`.
   Vérifier qu'en écrivant la ligne suivante vous obtenez toutes les matrices de taille 2×2:
   `for i in range(2**4): print(int_to_mat(i, 2, 2))`
4. Écrire une fonction `energies(n, p)` renvoyant la liste des énergies possibles pour une matrice de taille `n`×`p`.
5. Afficher ce résultat sous forme d'histogramme en écrivant, par exemple: `plt.hist(energies(3, 3))`

Remarque: il est possible de faire plus efficace en utilisant une autre énumération (code de Gray) pour ne pas avoir besoin de recalculer l'énergie entièrement pour chaque matrice.