Ising 2D

Rapport du projet d'Informatique n°2

Arveiler Maël, Duval Hugo

INTRODUCTION, GENERALITES.

Le modèle d'Ising, dénommé d'après le physicien Ernst Ising, est un modèle de physique statistique. Il a été utilisé pour modéliser différents phénomènes dans lesquels des effets collectifs sont produits par des interactions locales entre particules à deux états.

Le fondement du modèle d'Ising 2D consiste en un réseau carré NxN, dont les nœuds sont occupés par des particules. Chaque particule possède une information : son spin, égal à +1 (up) ou -1 (down). Ces spins interagissent entre eux, et tendent à ordonner le système. La température joue également un rôle : plus elle est élevée, moins le système est ordonné. Au bout d'un certain temps, une configuration d'équilibre est atteinte. Une fois cet équilibre atteint, on définit alors deux grandeurs physiques : l'aimantation et l'énergie.

L'algorithme de Métropolis est une méthode permettant d'obtenir des configurations d'équilibre dans le but de calculer ces grandeurs. Cet algorithme consiste à modifier un spin aléatoire d'une configuration préalablement établie, et d'observer la variation d'énergie engendrée. Selon sa valeur, on décide si le changement de spin initial est accepté ou pas. En règle générale, un changement de spin n'est accepté que s'il abaisse l'énergie du système. Cependant, il existe une probabilité que cette modification soit acceptée si elle engendre une augmentation d'énergie, probabilité qui augmente avec la température imposée : ce qui explique qu'à haute température, l'équilibre adopté est moins ordonné. C'est donc en répétant cette étape un très grand nombre de fois, i.e en faisant évoluer le réseau vers une structure d'équilibre, qu'il est possible de déterminer l'aimantation et l'énergie moyennes par spin.

Ainsi, dans le cadre de ce projet, notre objectif est, par l'intermédiaire de l'algorithme de Métropolis, de calculer ces grandeurs, et de simuler les fluctuations d'une configuration d'équilibre à travers une animation.

Analyse des solutions a mettre en œuvre.

Le choix que nous avons fait pour modéliser le réseau est de créer une matrice carrée, initialement remplie de +1, qui représentent les spins. La première étape du programme est la génération d'une configuration stable à l'aide de l'algorithme de Métropolis (fonction *metropolis*). Pour mettre en œuvre ce dernier, il faut inverser un spin aléatoire et calculer la variation d'énergie résultante qui dépend des spins de ses 4 voisins. Si le spin modifié est en bordure de grille, c'est le spin à l'autre extrémité qui est pris en compte (fonction *neighbor*). En réalité, on ne modifie pas la configuration de base avant de savoir si la modification est acceptée ou non. Cette dernière est validée si la variation d'énergie est positive, sinon, il faut alors calculer une probabilité dépendante de la température qui déterminera si le changement est quand même accepté ! Ainsi, à l'aide des fonctions *initialise* et *run_metro*, on génère une configuration stable en répétant l'algorithme 50*N*N fois l'algorithme. Le nombre d'itérations est limité si celle-ci dépasse une certaine taille, dans le but d'optimiser la vitesse d'exécution.

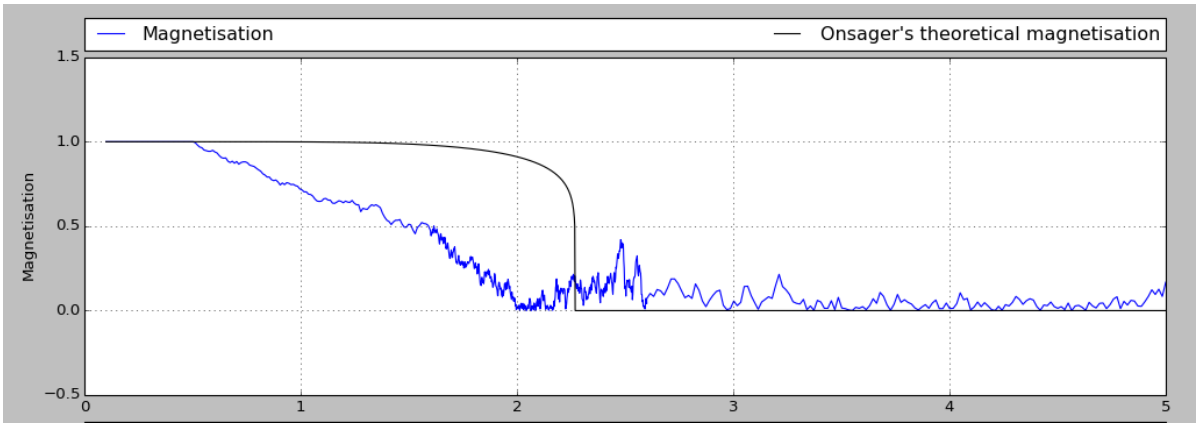
Il est à présent aisé de calculer nos valeurs moyennes grâce aux formules proposées, pour une température donnée (fonctions *energy* et *magnetisation*). Cependant, les résultats à obtenir sont les graphiques de l'aimantation et de l'énergie en fonction de la température. Pour cela, nous faisons appel à la fonction *graphs*. Elle permet d'exécuter l'algorithme de Métropolis un très grand nombre de fois, et de calculer les valeurs de l'aimantation et de l'énergie toutes les N*N exécutions, tout en faisant varier la température. La fonction *M_Onsager*, est également appelée pour confronter les valeurs expérimentales obtenues à la référence théorique. Le nombre de points Curie augmente afin d'observer de manière plus précise la transition de phase. En outre, les graphiques exigés sont tracés, tout en informant l'utilisateur de l'avancée du calcul lors de l'exécution du programme.

Enfin, il s'agit pour finir de générer l'animation de la configuration qui évoluera en temps réel, selon une température choisie arbitrairement par l'utilisateur. Cette fonctionnalité du programme est remplie par *animated_lattice,* qui affiche le réseau de spins up et down représentés par des pixels noirs et blancs ; applique l'algorithme de Metropolis ; et met à jour la configuration à chaque fois qu'un spin est modifié. L'utilisateur voit alors le système se stabiliser vers un état d'équilibre. Il est également mis à sa disposition un curseur à l'aide duquel il peut changer la température à tout instant. Ces changements de température sont pris en compte dans l'algorithme, et comme expliqué précédemment, cela aura pour effet de donner une impression de système plus ou moins ordonné. C'est la fonction pré-intégrée à Python : *FuncAnimation*, qui gère l'affichage du réseau et le met à jour.

En outre, nous avons fait le choix d'intégrer au programme une interface graphique afin de rendre ce dernier plus convivial. L'utilisateur y rentre la taille de la matrice qu'il désire, son choix est guidé selon s'il souhaite voir l'animation ou les courbes.

LIMITES DU PROGRAMME ET CONCLUSION.

Le principal défaut de l'algorithme est la non-concordance des courbes d'aimantation par rapport à la théorie. En effet, la plupart du temps, l'aimantation chute bien avant la température de Curie, comme montré ci-dessous :



C'est un problème que nous n'avons pas réussi à corriger malgré les tentatives, nous avons pourtant essayé d'y remédier en modifiant le nombre de points tracés et en augmentant le nombre d'exécutions de l'algorithme de Métropolis, mais sans succès.
A cela, on peut ajouter que le temps de calcul du programme général est relativement élevé, surtout lorsqu'il s'agit de tracer les courbes et animer de grandes matrices. Justement, la fluidité de l'animation serait également à revoir, même si elle fonctionne assez bien d'une manière générale. Enfin, on peut remarquer une erreur dans la console lors du lancement de l'animation, due au fait que l'on est obligé de quitter deux fois la fenêtre Tkinter pour que le bon fonctionnement du programme. C'est un bug mineur qui mériterait d'être corrigé mais qui est sans impact sur le reste du codage.

Pour conclure, nous avons effectivement pu mener le projet à terme en répondant aux objectifs à atteindre, ceci en essayant de rendre le programme le plus clair et compréhensible possible pour l'utilisateur.