Modèle d'Ising : Magnétisme et Algorithme de Monte-Carlo
Master 1 Physique

Vincent.Favre-Nicolin@ujf-grenoble.fr

2 février 2005

1 Introduction

Dans ce projet nous allons étudier les phénomènes de magnétisme en utilisant le modèle d'Ising. Nous étudierons pour cela un réseau bidimensionnel infini de spins. L'énergie d'interaction (couplage) entre spins dans un matériau peut (interaction entre deux moments dipolaires) :

$$E = \frac{\overrightarrow{\mu_1}.\overrightarrow{\mu_2}}{r^3} - 3\frac{(\overrightarrow{\mu_1}.\overrightarrow{r})\,.\,(\overrightarrow{\mu_2}.\overrightarrow{r})}{r^5}$$

où $\overrightarrow{\mu_1}$ et $\overrightarrow{\mu_2}$ sont les deux moments considérés, et $\overrightarrow{r}$ le vecteur liant les origines de ces moments. En supposant pour simplifier que (1) les spins sont verticaux et (2) en négligeant les interactions au-delà des premiers voisins, on écrira plus simplement (et plus généralement, J pouvant être négatif) l'énergie d'interaction sous la forme [1] :

$$E = -JS_iS_j$$

où J est la constante de couplage et S_i et S_j sont les projections des spins selon l'axe vertical.

Le but de ce modèle sera d'étudier le comportement d'un tel système en fonction de la température et des différents paramètres (valeur du couplage J, présence d'un champ magnétique).

2 Modélisation du réseau de Spins : MatriceSpin

2.1 Principe

Afin de modéliser un réseau de spins nous pouvons utiliser une matrice dont les éléments M(i,j) seront les valeurs de spins +1 ou -1. Au lieu de considérer un 'vrai' réseau infini, nous allons travailler sur un réseau périodique carré NxN, tel que $S(i,,j) = S(i + n_1 * N, j + n_2 * N)$. Pour modéliser cela on crée une classe MatriceSpin qui comprend un tableau carré (le tableau des spins, égaux à 1 ou -1), et ajouter à cette fonction un opérateur qui permet d'accéder à un spin [i,j] quelconque, même en dehors de l'intervalle [0 :N-1 , 0 :N-1].

2.2 Implémentation

Voici une partie du code permettant de commencer :

```
from ROOT import gROOT, TCanvas, TH2C, TGraph # pour les affichages graphiques
import Numeric # pour stocker la matrice de Spins
import RandomArray # pour la génération de nombres pseudo-aléatoires
```

[1] En fait l'interaction est plus compliquée et il peut exister une compétition entre l'*interaction d'échange* et l'*interaction dipolaire*, mais nous n'entrerons pas dans le détail.

```
from time import sleep

# initialise la racine de la génération de nombres pseudo-aléatoires
RandomArray.seed()
RandomArray.random()     # Renvoie un nombre aléatoire entre 0.0 et 1.0


# Cette classe ouvre une fenêtre pour un graphe 1D
# Elle permet d'afficher un graphique y=f(x), les listes de valeurs
# sont stockées dans les tableaux 1D listex et listey.
class FenetreGraphe1D :
   def __init__(self,listex,listey,nom="Graphe 1D",largeur=400,hauteur=400,x0=10,y
      self.fenetre=TCanvas(nom,nom, x0, y0, largeur, hauteur )
      self.graphique=TGraph(len(listex))
      self.graphique.SetLineColor( 2 )
      self.graphique.SetLineWidth( 4 )
      self.graphique.SetMarkerColor( 4 )
      self.graphique.SetMarkerStyle( 21 )
      self.graphique.SetTitle( "nom" )
      self.graphique.GetXaxis().SetTitle( 'X' )
      self.graphique.GetYaxis().SetTitle( 'Y' )
      self.Dessin(listex,listey)
   def Dessin(self,listex,listey) :     # Affiche de nouvelles valeurs
      self.fenetre.Clear()
      self.graphique.Set(len(listex))
      for i in range(len(listex)) :
         self.graphique.SetPoint(i,listex[i],listey[i])
      self.graphique.Draw( 'ACP' )
      self.fenetre.Update()


# Cette classe ouvre une fenêtre pour un graphe 2D z=f(i,j)
# Les valeurs sont stockées dans la variable tableau.
class FenetreGraphe2D :
   def __init__(self,tableau,nom="Graphe 2D",largeur=400,hauteur=400,x0=10,y0=10)
      self.fenetre=TCanvas(nom,nom, x0, y0, largeur, hauteur )
      self.graphique=TH2C(nom,nom,tableau.shape[0],1,tableau.shape[0],tableau.shap
      self.graphique.SetXTitle('X');
      self.graphique.SetYTitle('Y');
      self.Dessin(tableau)
   def Dessin(self,tableau) :    # Affiche un nouveau tableau
      self.graphique.SetBinsLength(tableau.shape[0]*tableau.shape[1])
      for i in range(tableau.shape[0]) :
         for j in range(tableau.shape[1]) :
            self.graphique.SetBinContent(i,j,tableau[i,j])
      self.fenetre.Clear()
      self.graphique.SetStats(False)
      self.graphique.Draw("col")
      self.fenetre.Update(


class MatriceSpin :
   def __init__(self,taille=20,J=1.0) :
```

```
        self.matrice=Numeric.ones((taille,taille)) # Nouvelle matrice remplie
de 1
        self.taille=taille # on stocke la taille de la matrice
        self.fenetre2D=FenetreGraphe2D(self.matrice,'Matrice de Spins')
#fenêtre d'affichage
        self.J=J # on stocke la constante de couplage
     def InitAleatoire(self) :# Initialisation aléatoire de la matrice des
spins
       ??????????????????
       ??????????????????
     def __getitem__(self,index) :
     # on écrit un opérateur [], on attend 2 valeurs dans l'index (i.e.
[2,5])
     # il faut "ramener" les indices i,j dans [0;N-1] par périodicité,
     # pour renvoyer le bon spin
       ??????????????????????
        return?????????????
     def __setitem__(self,index,valeur) : # opérateur [] , exemple : maMatriceSpin[5
     # on écrit un opérateur [], on attend 2 valeurs dans l'index (i.e.
[2,5])
     # il faut "ramener" les indices i,j dans [0;N-1] par périodicité,
     # pour changer la valeur du spin à cette coordonnée
       ??????????????????????
       ??????????????????????
     def Affiche(self) :
        self.fenetre2D.Dessin(self.matrice)
```

Dans le code ci-dessus, ajouter le code correspondant aux fonctions `InitAleatoire`() (initialisation de tous les spins à une valeur aléatoire +1 ou -1), `__getitem__`() et `__setitem__`() (qui permettent d'accéder à la valeur d'un spin dans le plan pour des valeurs quelconques de [i,j], en utilisant la périodicité) de la classe MatriceSpin.

Après avoir écrit cela, tester en créant une matrice 20x20, en l'affichant, modifiant quelques spins, et en utilisant l'initialisation à des spins aléatoires...

3 Méthode de Monte-Carlo

3.1 Algorithme

L'algorithme de Monte-Carlo est très général : il peut s'appliquer à tout modèle pour lequel il est possible (i) de faire des ***modifications aléatoires du modèle*** (en général à partir d'un nombre fini de variables continues ou discrètes qui sont changées de manière aléatoire) et (ii) d'associer une ***variation d'énergie*** à chacune de ces modifications. L'algorithme procède alors de la manière suivante :

- soit une modification aléatoire qui transforme le modèle M_i en modèle M_{i+1}, associé à une variation d'énergie ΔE.
- Si $\Delta E < 0$, la modification est acceptée.
- Si $\Delta E > 0$, la modification est acceptée avec la probabilité $P = e^{\frac{-\Delta E}{kT}}$, où T est la température de l'algorithme et k la constante de Boltzmann (on prendra k=1 pour simplifier).

Il est facile de voir qu'une fois un état stationnaire atteint, la distribution des états générés $\{M_i\}$correspnd à une distribution de Boltzmann.

3.2 Implémentation

3.2.1 Modifications de la classe MatriceSpin

Il faut tout d'abord rajouter une fonction membre `NouvelleConfiguration()` à la classe Matrice Spin : cette fonction choisit au hasard un spin, l'inverse, et renvoie le changement d'énergie associé à ce changement de configuration. En outre, cette fonction stocke les coordonnées du spin inversé de manière à pouvoir annuler le changement si nécessaire.

Il est également nécessaire d'ajouter une fonction membre `AnnuleChangement()` pour annuler le changement si nécessaire.

Bien sûr, tester ces deux fonctions en générant une série de nouvelles configurations, en les dessinant avec la fonction `Affiche()`.

3.2.2 Classe MonteCarlo

La classe MonteCarlo implémente l'algorithme qui va tester des nouvelles configurations, en fonction de la température où se fait l'évolution. Elle se présente ainsi :

```
class MonteCarlo :
   def __init__(self,matrice,temperature=1.0) :
      # Stocke la matrice et la température comme variables membres
     ??????????????????????
   def Essai(self) :
      # Demande une nouvelle configuration aléatoire de la matrice .
      # En fonction de la variation d'énergie renvoyée (et de la température),

      # décide si la nouvelle configuration doit être annulée ou non
     ??????????????????????
   def Evolution(self,nb) :
      # Réalise nb essais, et affiche de temps en temps la matrice
     ???????????????????????????
```

Implémenter les 3 fonctions, et tester la fonction `Evolution()` sur un objet matrice.

4 Application

4.1 Evolution qualitative

Insérer une boucle qui fait varier la température entre 0.1 et 5, avec quelques points, et pour chaque température faire évoluer le système (en prenant une matrice de 20x20 spins) pour environ 50 000 essais et dessiner la matrice de spins.

Que se passe-t-il si on choisit J = 1 ? J = -1 ? Quels sont les différents régimes en fonction de la température ?

Pour J=1, essayer de faire évoluer un système à température fixe T = 0.5 pour une "grande" dimension (50x50 ou 100x100) de la matrice. Observer la formation de domaines magnétiques (au besoin réessayer plusieurs fois), et expliquer (qualitativement) la (méta)stabilité des domaines observés à partir de la forme des parois et de la taille des domaines.

4.2 Mesure de l'aimantation moyenne <M> et de l'énergie totale E

Rajouter une fonction membre `AimantationMoyenne()` à la classe `MatriceSpin` qui calcule la valeur moyenne des spins sur l'ensemble de la matrice, ainsi qu'une fonction `Energie()` qui calcule l'énergie moyenne associée à l'ensemble des spins (attention à ne pas compter deux fois le même terme dans le calcul de l'énergie...).

4.3 Vitesse de convergence

Lorsqu'un paramètre (i.e. la température) est modifiée il faut attendre un certain nombre d'essai pour que le système se stabilise (variation de la configuration microscopique mais pas de variation des grandeurs moyennes). Pour évaluer qualitativement la "convergence", on pourra tracer l'évolution de l'aimantation moyenne et de l'énergie totale en fonction du nombre d'essais. Tester pour différentes tailles de matrice, de 20x20 à 50x50.

4.4 Mesure de la longueur de corrélation (plus difficile)

La fonction de corrélation de spin peut s'écrire :

$$G(\overrightarrow{r}) = G(\overrightarrow{r_i} - \overrightarrow{r_j}) =< \overrightarrow{S_i}\overrightarrow{S_j} >$$

(valeur moyenne de$\overrightarrow{S_i}\overrightarrow{S_j}$ pour des spins situés à une distance $\overrightarrow{r}$)

A longue distance, il est possible d'approximer $G(\overrightarrow{r})$ sous la forme :

$$G(\overrightarrow{r}) \approx e^{\frac{-r}{\xi}}$$

où ξ est la longueur de corrélation du système, qui peut donc s'écrire pour r grand :

$$\xi \approx \frac{r}{< \overrightarrow{S_i}\overrightarrow{S_j} >}$$

où r est la distance sépérant les deux spins Si et Sj.

Rajouter une fonction `LongueurCorrelation()` à la classe MatriceSpin qui calcule la longueur de corrélation. Pour ce faire, on choisira un petit nombre (10% du nombre total) de spins au hasard et pour chacun on calculera leur corrélation avec 10% du nombre total de spins pris à une distance maximale (i.e. la moitié de la taille de la matrice, dans une direction aléatoire).

Nota bene : ceci n'est applicable qu'au voisinage de la température critique , car sinon l'exponentielle décroit trop vite (T<<Tc), ou la longueur de corrélation est trop grande (T>>Tc). Optionnellement, il est possible d'écrire la fonction de manière à *affiner* la fonction lnG=f(r) pour déterminer la pente $\frac{-1}{\xi}$, mais cela est plus complexe... Il faut de manière générale faire attention aux valeurs particulières ($< \overrightarrow{S_i}\overrightarrow{S_j} >= 0$ ou 1)... Il pourra être intéressant d'utiliser les fonctiàns d'affinement du module SciPy.

4.5 Mesure de la Température critique T_c

Ajouter dans le programme principal une boucle en fonction de la température, en laissant le système évoluer suffisament longtemps à chaque température, et stocker dans des tableaux 1D les valeurs de l'aimantation moyenne, de l'énergie et de la longueur de corrélation (on prendra J=1). A la fin de la boucle, afficher les différents graphes (aimantation moyenne, énergie, longueur de corrélation) à l'aide de la classe `FenetreGraphe1D` et interpréter.

Trouver la valeur de la température critique. On doit vérifier :

$$T_c = \frac{2J}{k \ln(1+\sqrt{2})}$$

Note :

Il est normalement possible de mesurer également la ***chaleur spécifique*** :

$$C_p = \frac{\partial E}{\partial T}$$

ainsi que la ***susceptibilité magnétique*** :

$$\chi = \frac{\partial < M >}{\partial T}$$

...mais dans la pratique les simulations que nous faisons sont trop rapides pour bien mettre en évidence les propriétés de ces fonctions, particulièrement au voisinage de T_c (la taille du système et le nombre d'itérarions n'est pas assez grand pour obtenir une courbe suffisament lisse pour effectuer la dérivation).

Comment varie la convergence (tracer E et <M> en fonction du nombre d'essais) au voisinage de T_c ?

5 Etude en présence d'un champ magnétique

5.1 Introduction du terme énergétique liés au champ magnétique

En présence d'un champ magnétique H vertical (parallèle à la direction des spin), l'énergie du système s'écrit sous la forme :

$$E = -J \sum S_i S_j - 2\mu_B H \sum S_i$$

Programmation

Ajouter à la classe `MatriceSpin` une variable membre H (amplitude du champ magnétique).

Modifier ensuite les fonctions `NouvelleConfiguration()` et `Energie()` pour tenir compte du nouveau terme dans le calcul de l'énergie (on prendra $\mu_B = 1$ pour simplifier).

5.2 Etude du phénomène d'hystérésis : <M>(H)

En prenant J=1 et une taille "raisonnable" de la matrice de spin (20x20 à 50x50 suivant la puissance de l'ordinateur), effectuer une boucle faisant varier le champ magnétique de -1 à +1, en laissant évoluer le système après chaque variation du champ magnétique. Là encore, stocker les valeurs du champ magnétique et de l'aimantation moyenne (et accessoirement de l'énergie et de la longueur de corrélation) dans des vecteurs et les afficher en fin de programme.

Constater les différents effets en fonction de :

- T < Tc ou T>Tc
- sens de variation de H, de -1 à +1 ou de +1 à -1. Observer l'effet d'hystérésis. Comment varie-t-il en fonction de la position de T par rapport à Tc ?