Très bon travail. Les programmes étaient difficiles à mener à bout.
et tu as bien su régler les paramètres.

AHMED BACHIR

RAPPORT TPK

Algorithme de Metropolis

1 Principe de l'algorithme

Exercice 2.1 : Vérifions que (P,π) satisfait l'équation :

$$\forall x, y \in E \quad \pi(x)P(x,y)=\pi(y)P(y,x)$$

On sait que : $P(x,y)=Q(x,y)\,min(\frac{\pi(y)}{\pi(x)},1)$ (1)

$P(y,x)=Q(y,x)\,min(\frac{\pi(x)}{\pi(y)},1)$ (2)

TB

-Si $\frac{\pi(y)}{\pi(x)}>1$, on a :

(1) => $P(x,y)=Q(x,y)$ (*)

(2)=> $P(y,x)=Q(y,x)\frac{\pi(x)}{\pi(y)}$ => $\pi(y)P(y,x)=\pi(x)Q(y,x)$

$\pi(y)P(y,x)=\pi(x)Q(x,y)$ (car Q symétrique)

$\pi(y)P(y,x)=\pi(y)P(x,y)$ (d'aprés (*))

Finalement $\pi(x)P(x,y)=\pi(y)P(y,x)$

-Si $\frac{\pi(y)}{\pi(x)}<1$ On montre de manière analogue que $\pi(x)P(x,y)=\pi(y)P(y,x)$

Déduisons-en que $\pi P=\pi$

On a :

$$\sum_{x\in E}\pi(x)P(x,y)=\sum_{x\in E}\pi(y)P(y,x)=\pi(y)\sum_{x\in E}P(y,x)=\pi(y)$$

Finalement : $\pi P=\pi$

3 Un exemple simple

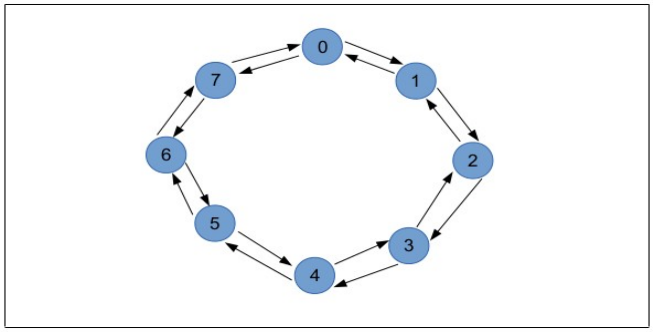
Décrivons la chaîne

E est un ensemble de 8 points disposés en cercle, que l'on identifie à $\mathbb{Z}/8\mathbb{Z}$. La loi à simuler est π = [π(0), ..., π(7)] définie par

$$\forall x \in \{0,\dots,7\} \quad \pi(x)=\frac{(x+1)^2}{Z} \quad \text{avec} \quad Z=\sum_{y\in E} y^2$$

Q est la matrice stochastique associée au graphe ci-dessous, ainsi :

$$Q(x,y)=\frac{1}{2}\times 1_{\{y=x+1 \mod 8\}}+\frac{1}{2}\times 1_{\{y=x-1 \mod 8\}}$$



Nous allons utiliser la classe Mcmc pour décrire cette chaîne :
-On ne définit aucun graphe, aucune matrice de transition. La chaîne de Markov (X t), est construite en utilisant la mécanique de transition liée à Q: je monte de 1 avec proba 1/2, je descends de 1 avec proba 1/2.

```
int aleat = 2 * rand.nextInt(2) - 1;
int xi=rand.nextInt(8);
int xi = ((etatCourant + aleat + nbEtat) % nbEtat);
```

- ***La constante Z n'a pas été calculée puisqu'il suffit de connaître*** π ***à une constante pré***

exact

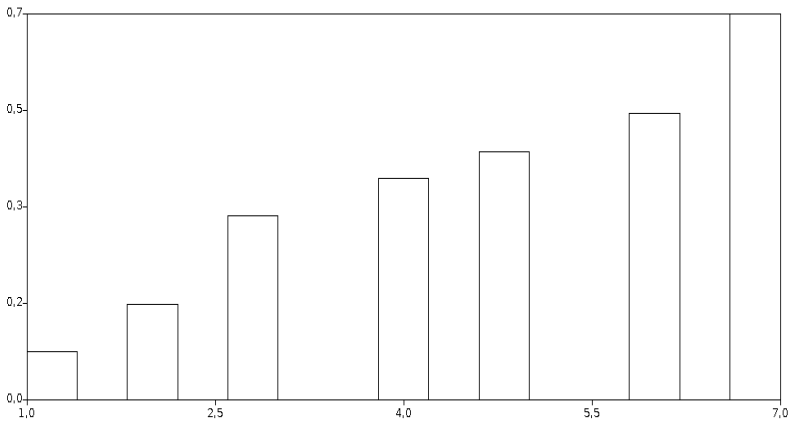
Généralisons au cas où π s'annule

Toujours avec la mécanique d'exploration ,testons le programme avec **:**

$$\forall\ x\in\{0,1,\ldots,7\}:\ \pi(x)=x$$

```
public McmcEnCercle() {
for (int i = 0; i < nbEtat; i++) {
  pi.add((double) (i ));
 }
}
```

Nous avons en sortie :

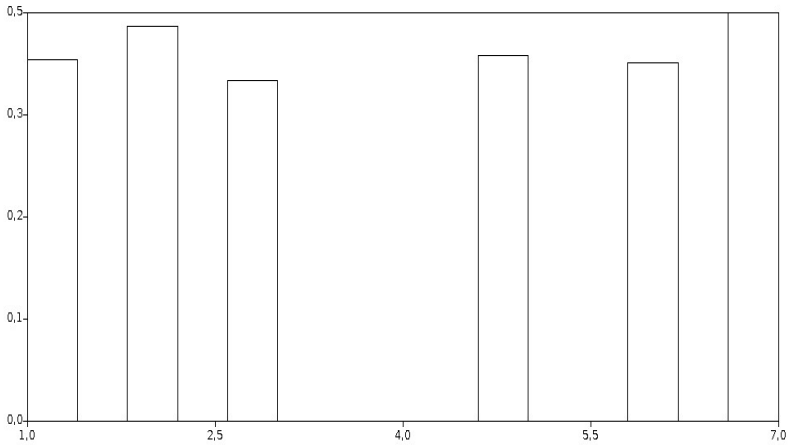


On voit bien que ça marche , $\pi(0)$ étant nulle l'état 0 n'a pas été visité

Testons maintenant le programme avec : $\pi(x)=1_{\{}x\in 1,2,3\}+2\times 1_{\{}x\in 5,6,7\}$

```
public McmcEnCercle() {
for (int i = 0; i < nbEtat; i++) {
  if (i == 1 || i == 2 || i == 3) {pi.add(1.);}
  else if  (i == 5 || i == 6 || i == 7){pi.add(2.);}
   else {pi.add(0.);}
 }
}
```

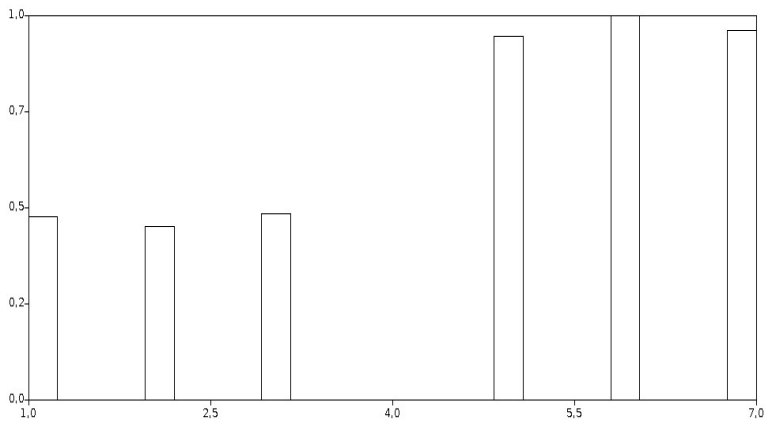
Nous obtenons en sortie :



Nous observons que cette chaîne visite autant de fois les états {1,2,3} que les états {5,6,7} ,alors qu'elle devait visiter deux fois plus les états {5,6,7} que les états {1,2,3} . bien compris

Ce résultat est assez logique vu la mécanique d'exploration qui a été choisie pour cette chaîne . En effet les probabilités de rester dans l'état 0 ou dans l'état 4 étant toutes deux nulles . Deux groupe d'états sont visitables dans cette chaîne à savoir {1,2,3} et {5,6,7} . Dés que l'on tombe dans un de ces groupes, la transition ne s'effectue qu'entre état avec ses plus proches voisins du même groupe ,c'est qui fait qu'ici notre probabilité invariante ne peut pas avoir l'effet escompté sur cette chaîne.
Pour remédier à cela nous devons changer la mécanique d'exploration . L'idée est de choisir une mécanique telle qu'il puisse y avoir transition entre les éléments du groupe {1,2,3} et {5,6,7} .On pose alors $Q(x,y)=\frac{1}{8}$. Dans ce cas on a: `xi=rand.nextInt(8);`

Et on a en sortie :



On vient bien maintenant que la chaîne le groupe {5,6,7} est deux fois plus visité que {1,2,3}

4 .Exemple complexe

- Considérons une grille de taille d^2 ,c-à-d contenant $d \times d$ sites
- Chaque site accueille une particule avec une charge $x_i = +1 \text{ ou } -1$
- L'ensemble des "configuration" (=état) est donc $E = \{-1, +1\}^G$
- On définit un Hamiltonien : $H(x) = \sum_{(i,j):i\sim j} x_i x_j$ avec $(i \sim j = x_i \text{ et } x_j \text{ sont voisins})$
- La probabilité d'observer une configuration particulière *x* est :

$$\pi(x) = \frac{1}{Z} e^{-\beta H(x)} \quad \text{avec} \quad Z = \sum_{x \in E} e^{-\beta H(x)}$$

-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

Grille avec la particules de charge -1 (enn rouge) et ses voisins (en bleu)

Simulation

Afin de réaliser cette simulation j'ai eu à créer deux classes : **Grille** et **DessineGrille** .

La classe Grille

Variables globales:

```
private int dim;
private int [][] content ;
Random rand = new Random();
```

- La variable dim est la dimension de la grille
-Content est la matrice de taille *dim* x *dim* qui contient toutes les valeurs des charges de notre grille

Constructeur :Ce constructeur crée la matrice *content* remplie aléatoirement de +1 et -1 avec proba=1/2

```
public Grille(int dim) {
this.dim=dim;
  this.content= new int [dim][dim];
  for (int i = 0; i < dim; i++)
    for (int j = 0; j < dim; j++)
      this.content[i][j] = 2 * rand.nextInt(2) - 1;
}
```

bonne présentation des programmes

Accesseur et mutateur

```
public int getDim(){return dim ;}
public int [][] getContent(){return content;}
```

Les méthodes

Hamiltonien

```
public int Hamiltonien (int a ,int b){
 int Hamiltonien [] [] = new int [dim][dim];
 for (int i = 0; i < dim; i++) {
   for (int j = 0; j < dim; j++) {
     int voisin1 = content[(i - 1+dim) % dim][j];
     int voisin2 = content[i][(j - 1+dim) % dim];
     int voisin3 = content[(i + 1+dim) % dim][j];
     int voisin4 = content[i][(j + 1+dim) % dim];
     Hamiltonien[i][j] = content[i][j] * (voisin1 + voisin2 + voisin3 + voisin4);
   }
 }
 return Hamiltonien[a][b];
}
```

Cette fonction sert à calculer l'hamiltonien .Elle prend en argument deux entiers qui sont l'abscisse et l'ordonnée d'une case de la grille autour de laquelle on veut calculer l'hamiltonien.A chaque simulation ce dernier est calculé sur toute la grille .Ce procédé n'est pas très optimal certes , mais le plus important ici c'est que cela marche .

Voisin1 est le voisin *nord* de la particule
Voisin 2 est le voisin *oues*t
Voisin 3 est le voisin *sud*
Voisin4 est le voisin *est*

tu as choisi de travailler sur un tore, bonne idée de simplification

Au niveau des conditions de bord à la limite de la grille nous allons considérer que la grille est une surface «fermée» (au sens mathématique du terme) .Ainsi notre première particule tout en haut à gauche a comme voisins habituels celui du dessous et celui de droite ,mais il a aussi comme voisin à gauche la dernière particule de sa ligne et comme voisin du dessus la dernière particule de sa colonne.

La méthode pour simuler par mcmc

```
public void simulGrilleParMcmc (double beta,int nbSimulation){
for (int i=0;i<nbSimulation;i++) {
  int ia = rand.nextInt(dim);
  int ib = rand.nextInt(dim);
  double pi = (double) Math.exp(-beta * Hamiltonien(ia, ib));
  double choix = rand.nextDouble();
  if (choix < pi) {
    content[ia][ib] = -content[ia][ib];
  }
 }
}
```

Cette méthode prend en paramètre un réel *beta* (= β) et un entier *nbSimulation* qui correspond au nombre de configuration que l'on veut simuler .
Sa principale fonction c'est en premier de tirer manière aléatoire l'abscisse i_a et l'ordonnée i_b d'une case contenant un particule x_i de la grille .Puis inverse la charge cette particule avec un probabilité $\pi(x_i)$

La classe DessineGrille

C'est cette classe qui va nous permettre d'obtenir les coordonnées de chaque particule pour réaliser le dessin de la grille via le multiplot . Elle implémente donc , naturellement les interfaces PlotablePoint et PlotableColored . Ces dernières sont indispensables pour la création de boules de rayons et couleurs variables.Dans le dessin qui s'affichera à la fin de la simulation , chaque boule représente une particule .Les coordonnées d'une particule dans le dessin sont exactement les mêmes que les siennes dans la matrice .En d'autres termes ,pour réaliser le dessin on effectué une «projection» de la matrice sur le plan 2d .
Voici le coeur de cette classe :

```
public DessineGrille(Grille P){
Random rand =new Random();
 for (int i=0;i<P.getDim();i++){
  for (int j=0;j<P.getDim();j++)
  {Xs.add(i*1.);}
 }
 for (int i=0;i<P.getDim();i++){
  for (int j=0;j<P.getDim();j++)
  {Ys.add(j*1.);}
 }
 for (int i=0;i<P.getDim();i++){
  for (int j=0;j<P.getDim();j++){
   colors.add((double)P.getContent()[i][j]);
   radii.add(8.);
   //radii.add((double)P.content[i][j]); //des boules dont le rayon dépend de la charge de la particule
  }
 }
}
```

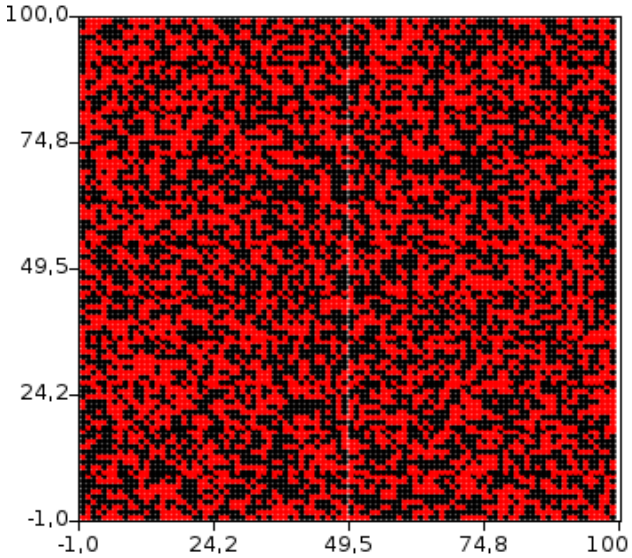
Maintenant que le programme est réalisé , on va maintenant le tester .

```
int dim=100;
Grille G =new Grille(dim);// Création de la grille
G.simulGrilleParMcmc(10, 50000);//simulation des configurations
multi.addPlotable("identité pointé", new DessineGrille(G));
multi.imposeRectangle(-1, -1, dim, dim);
multi.plotNow();
```

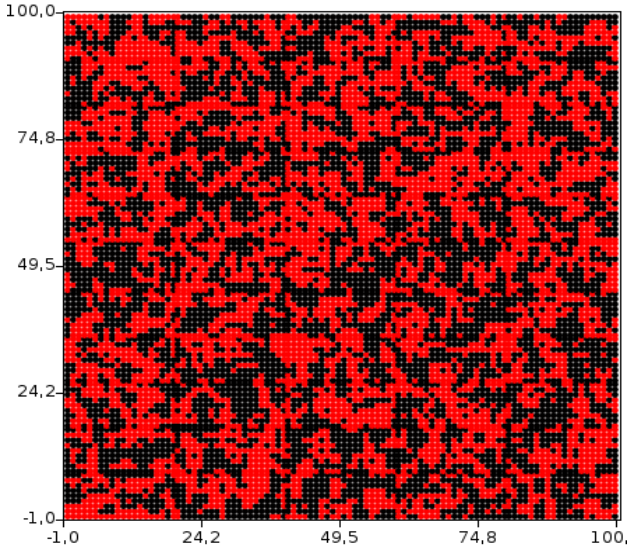
plutot noir

Les particules positives sont en bleues et les négatives en rouge

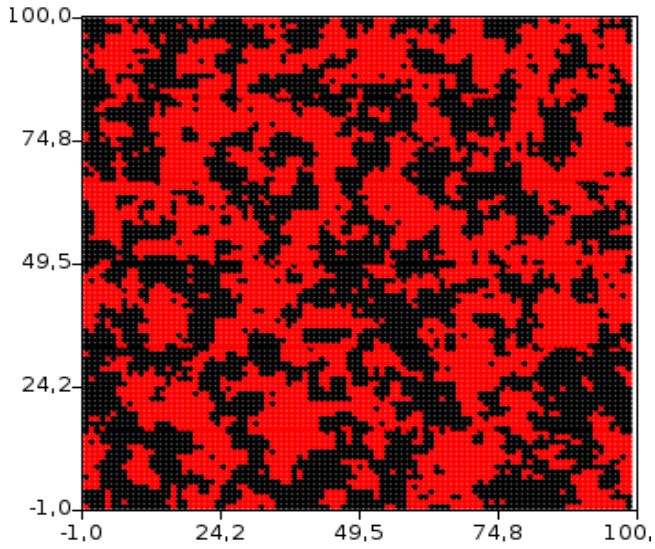
$\beta=0.1$



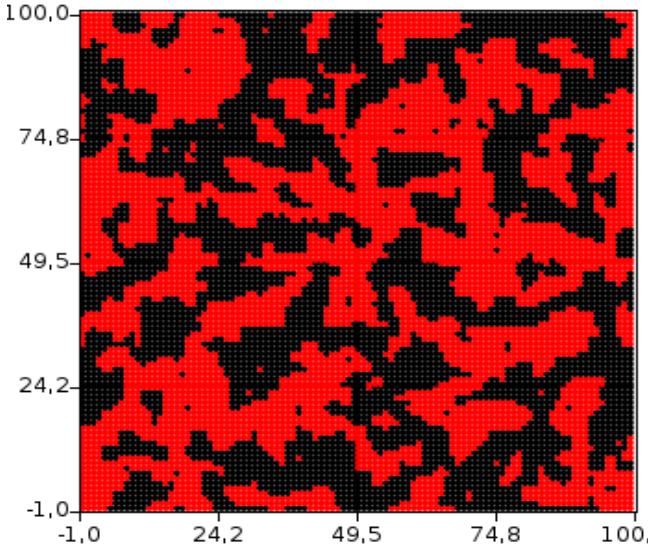
$\beta=0.5$



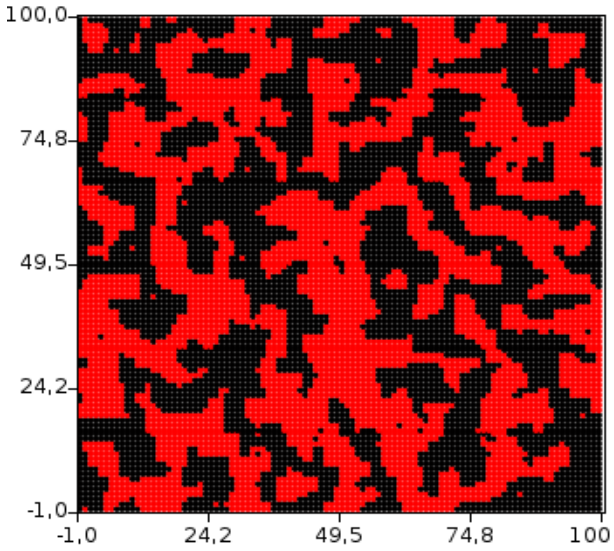
β=1



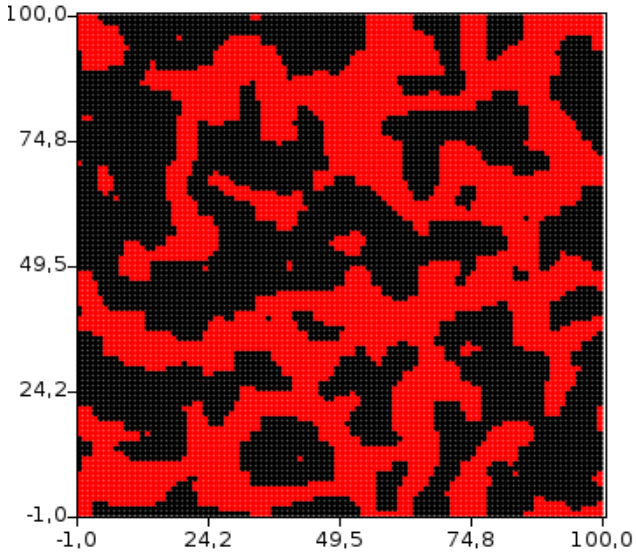
β=2



β=5



β=10



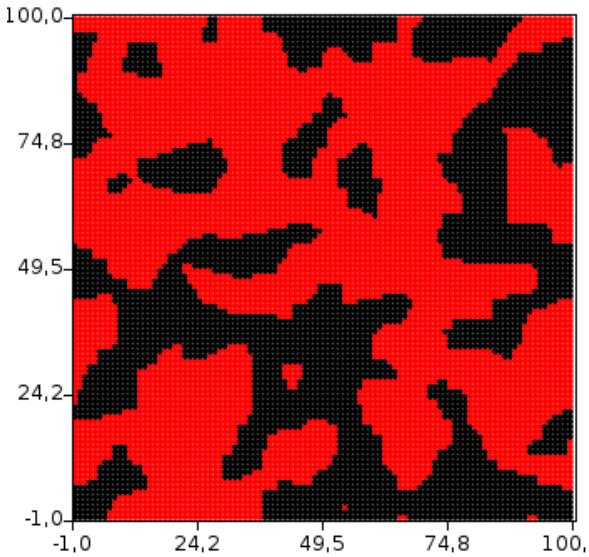
parfait mais je pense que tu es déjà sur le second exemple ici : les particules se rassemblent.

Conclusion: On remarque que plus β augmente plus la configuration devient stable ,c'est à dire les particules de même charge ont tendance à se regrouper ensemble .Ainsi il est bien cohérent que β représente l'inverse de la température .

Second exemple compliqué

Maintenant modélisons des particules d'huiles et d'eau mélangées sur une fine surface. Les particules d'un même type ayant tendance à se regrouper.

$\beta = 10$



6 Un troisième exemple compliqué

Modélisons de grosses particules sur une grille. Elles sont toutes de même type, mais, à cause de leur taille, elles ne peuvent pas occuper 2 sites contiguës. Remarquons qu'il s'agit d'un cas où π s'annule. En notant $x_i = 1$ la présente d'une particule sur le site i, et $x_i = 0$ l'absence de particule en i, on peut écrire:

$\pi(x) = \frac{1}{Z} \quad \forall\ i, j \in G \quad i \sim j \ \ on\ a \ \ x_i x_j = 0$

$\pi(x) = 0 \qquad\qquad\qquad\qquad\qquad\qquad sinon$

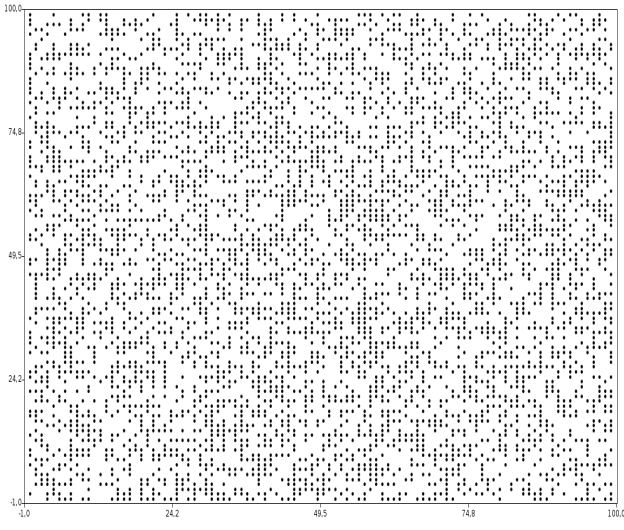
ou $Z = 2d$

Cet exemple est équivalent à l'exemple avec particules chargé de +1 ou -1 ,il suffit simplement de faire un changement de variable en posant :

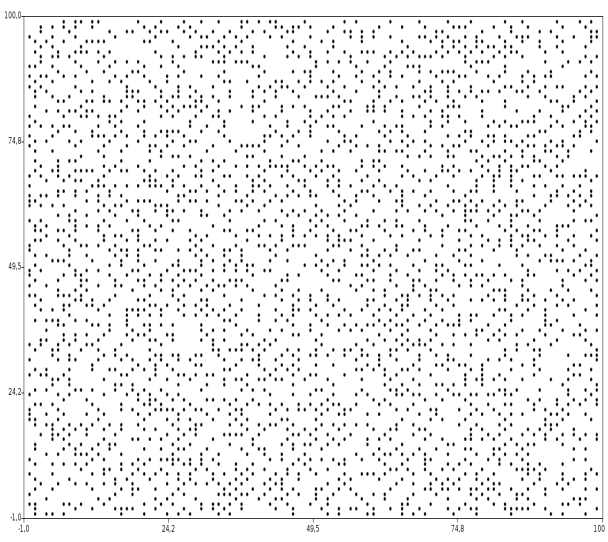
$x_i = \frac{1 - y_i}{2}$ avec $y_i = +1\, ou - 1$

On doit donc recalculer l'Hamiltonien : $H(x) = \sum_{(i,j):i\sim j} x_i x_j = \sum_{(i,j):i\sim j} (\frac{1-y_i}{2})(\frac{1-y_j}{2}) = \frac{1}{4} \sum_{(i,j):i\sim j} (1-y_i)(1-y_j)$

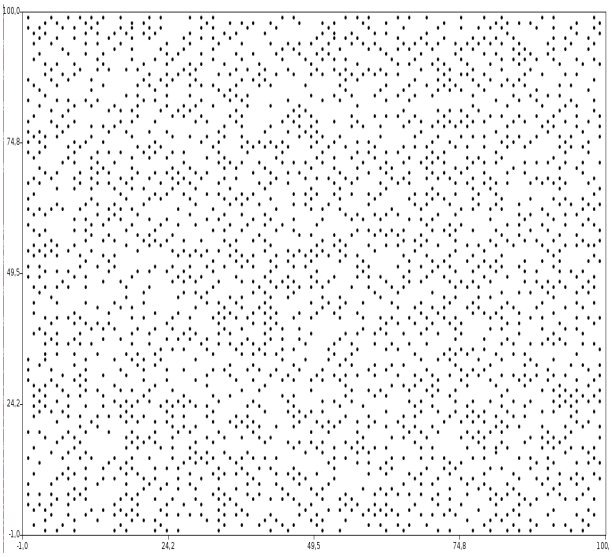
$\beta = 0.1$



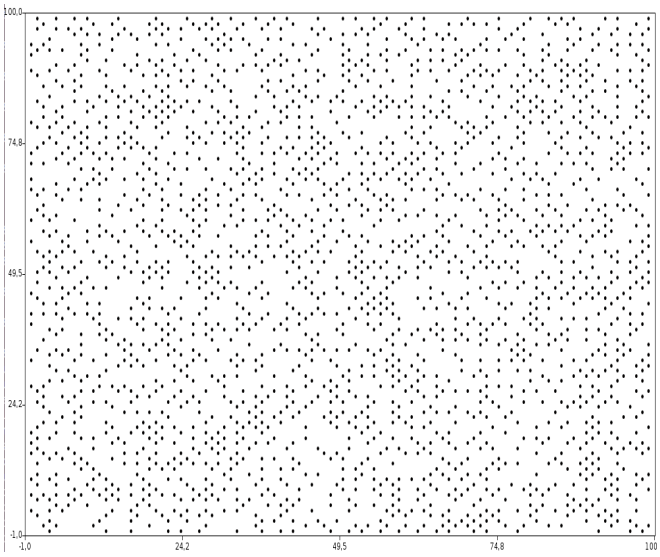
$\beta = 1$



β=5



β=10



Note: Pour mener à bien ces simulations , j'ai eu effectuer quelques modifications dans le Multiplot . Celles-ci ont été principalement faites dans les classes *PlotablePanel* et *PlotablePlus* de telle sorte que l'on puisse selon notre convenance ,générer des boules de couleurs ou de rayon nul pour représenter les particules et leurs états . .. Cependant ce que j'ai fait n'est utile que pour les simulations de ce Tp et dénature un peu le *PlotablePlus*.

c'est bien de disposer des sources d'un outils pour pouvoir l'adapter à sa sauce

Ce Tp a été très intéressant .Il m'a permit d'apprécier un peu plus l'utilité de chaînes de Markov . Et grâce aux explications bien détaillées des concepts évoqués dans l'énoncé ,ma compréhension des chaînes de Markov s'est plus enrichie.