TP4

BLANCHARD Simon

6 mars 2019

```
library(ggplot2)
library(gridExtra)
```

Exercice 1 : Simulation du modèle d'Ising

Question 1

La fonction suivante permet de générer une configuration initiale S correspondant à un maillage de taille $N \times N$ avec une proportion p de spins d'état -1 et une proportion $1-p$ de spins d'état 1 :

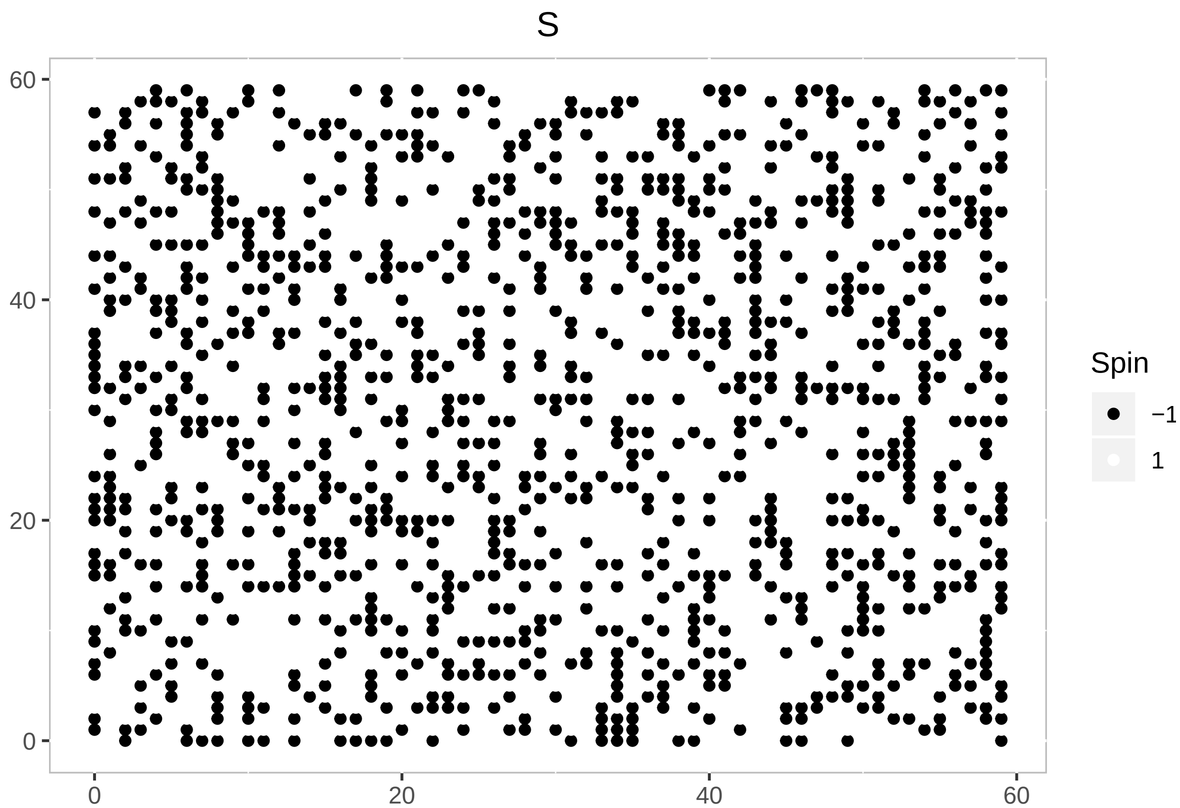
```
configuration_initiale_S <- function(N, p)
{
  I <- c()
  for(i in 0:(N - 1))
  {
    I <- c(I, rep(i, N))

  }
  J <- rep(0:(N - 1), N)
  S <- c()
  for(i in 1:(N * N))
  {
    u <- runif(1, 0, 1)
    S <- c(S, ifelse(u < p, -1, 1))
  }
  return(data.frame(I = I, J = J, S = S))
}
```

On génère alors une configuration initiale S :

```
S <- configuration_initiale_S(60, 0.3)
```

On affiche cette configuration initiale S :

```
S_plot <- ggplot(data = S, aes(x = I, y = J, color = factor(S), shape = factor(S))) + geom_point() + lal
S_plot
```



La fonction suivante génère une réalisation d'une loi uniforme sur $\{0, ..., N-1\}$:

```
realisation_loi_uniforme_gamma <- function(N)
{
  u <- runif(1, 0, 1)
  p <- 1 / N
  i <- 0
  while(u > ((i + 1) * p) && i < (N - 1)) i <- i + 1
  return(ifelse(i == (N - 1), N - 1, i))
}
```

La fonction suivante calcule $\Delta H(S, S_{(x,y)})$ pour une configuration initiale S correspondant à un maillage de taille $N \times N$:

```
delta_H <- function(S, x, y, N)
{
  s <- 0

  # en (0, 0)
  if(x == 0 && y == 0) s <- S$S[2] + S$S[N + 1]

  # en (N-1, N-1)
  else if(x == (N - 1) && y == (N - 1)) s <- S$S[N * (N - 1)] + S$S[N * (N - 1) + N - 1]

  # en (0, i) pour i dans 1,...,N-2
  else if(x == 0 && y >= 1 && y <= (N - 2)) s <- S$S[y] + S$S[y + 2] + S$S[N + y + 1]

  # en (0, N-1)
  else if(x == 0 && y == (N - 1)) s <- S$S[y] + S$S[N + y + 1]
```

```
  # en (i, 0) pour i dans 1,...,N-2
  else if(x >= 1 && x <= (N - 2) && y == 0) s <- S$S[1] + S$S[(x + 1) * N + 1] + S$S[x * N + 2]

  # en (N-1, 0)
  else if(x == (N - 1) && y == 0) s <- S$S[(N - 2) * N + 1] + S$S[N * (N - 1) + 2]

  # en (i, N-1) pour i dans 1,...,N-2
  else if(x >= 1 && x <= (N - 2) && y == (N - 1)) s <- S$S[N * x] + S$S[N * (x + 2)] + S$S[N * (x + 1)

  # en (N-1, i) pour i dans 1,...,N-2
  else if(x == (N - 1) && y >= 1 && y <= (N - 2)) s <- S$S[N * (N - 1) + y] + S$S[N * (N - 1) + y + 2]

  # en (i, j) pour i,j dans 1,...,N-2
  else s <- S$S[N * x + y] + S$S[N * x + y + 2] + S$S[N * (x - 1) + y + 1] + S$S[N * (x + 1) + y + 1]

  return(2 * s * S$S[N * x + y + 1])
}
```

La fonction suivante implémente l'algorithme de Hastings-Metropolis pour une configuration initiale S correspondant à un maillage de taille $N \times N$, $\beta = B$ et M itérations :

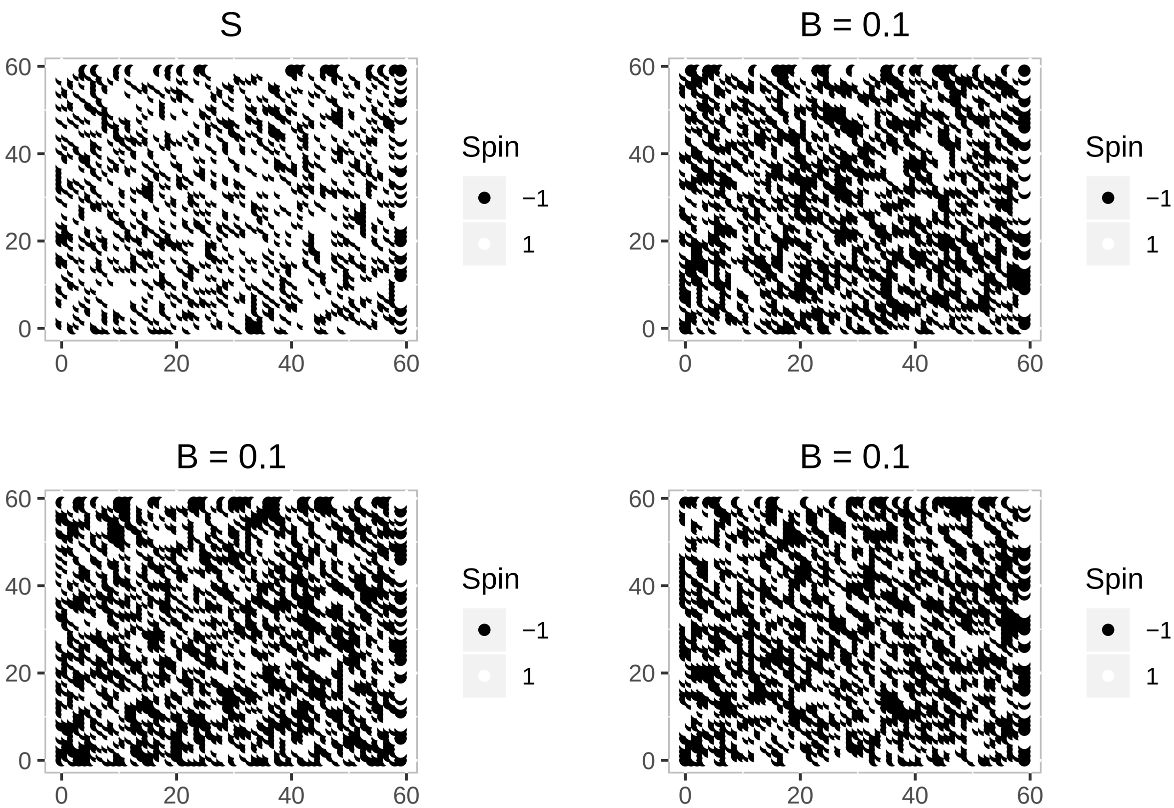
```
hastings_metropolis <- function(S, B, N, M)
{
  for(i in 1:M)
  {
    x <- realisation_loi_uniforme_gamma(N)
    y <- realisation_loi_uniforme_gamma(N)
    u <- runif(1, 0, 1)
    delta <- delta_H(S, x, y, N)
    S$S[N * x + y + 1] <- ifelse(u <= exp(-B * delta), - S$S[N * x + y + 1], S$S[N * x + y + 1])
  }
  return(S)
}
```

On génère alors trois modèles d'Ising de configuration initiale S avec $\beta = 0.1$ et $N = 60$:

```
Ising_01_1 <- ggplot(data = hastings_metropolis(S, 0.1, 60, 100000), aes(x = I, y = J, color = factor(S
Ising_01_2 <- ggplot(data = hastings_metropolis(S, 0.1, 60, 100000), aes(x = I, y = J, color = factor(S
Ising_01_3 <- ggplot(data = hastings_metropolis(S, 0.1, 60, 100000), aes(x = I, y = J, color = factor(S
```

On affiche ces trois modèles d'Ising ainsi que la configuration initiale S :

```
grid.arrange(S_plot, Ising_01_1, Ising_01_2, Ising_01_3)
```



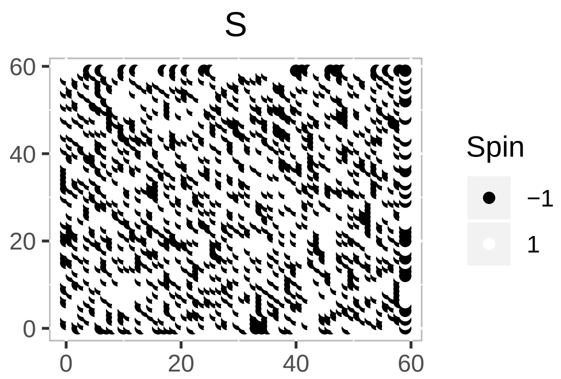
Question 2

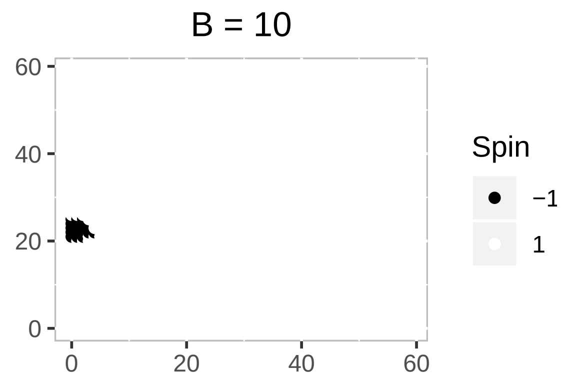
On génère trois modèles d'Ising de configuration initiale S avec $\beta = 10$ et $N = 60$:

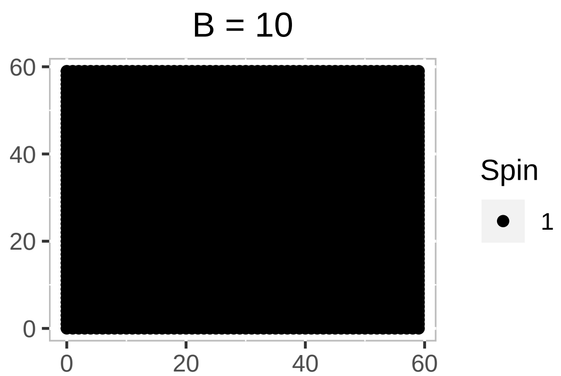
```
Ising_10_1 <- ggplot(data = hastings_metropolis(S, 10, 60, 100000), aes(x = I, y = J, color = factor(S)
Ising_10_2 <- ggplot(data = hastings_metropolis(S, 10, 60, 100000), aes(x = I, y = J, color = factor(S)
Ising_10_3 <- ggplot(data = hastings_metropolis(S, 10, 60, 100000), aes(x = I, y = J, color = factor(S)
```

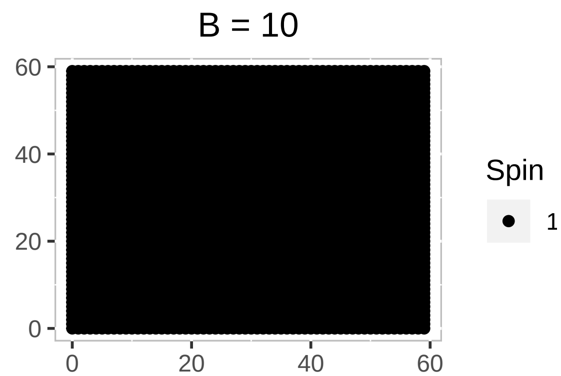
On affiche ces trois modèles d'Ising ainsi que la configuration initiale S :

```
grid.arrange(S_plot, Ising_10_1, Ising_10_2, Ising_10_3)
```









Question 3

On a :

$$\beta = \frac{1}{T}$$

On constate alors bien que lorsque β est faible (donc T élevée), les fluctuations thermiques dominent rendant le système désordonné. Au contraire, lorsque β est élevé (donc T faible), le système privilégie les configurations de basse énergie tendant à aligner les spins.

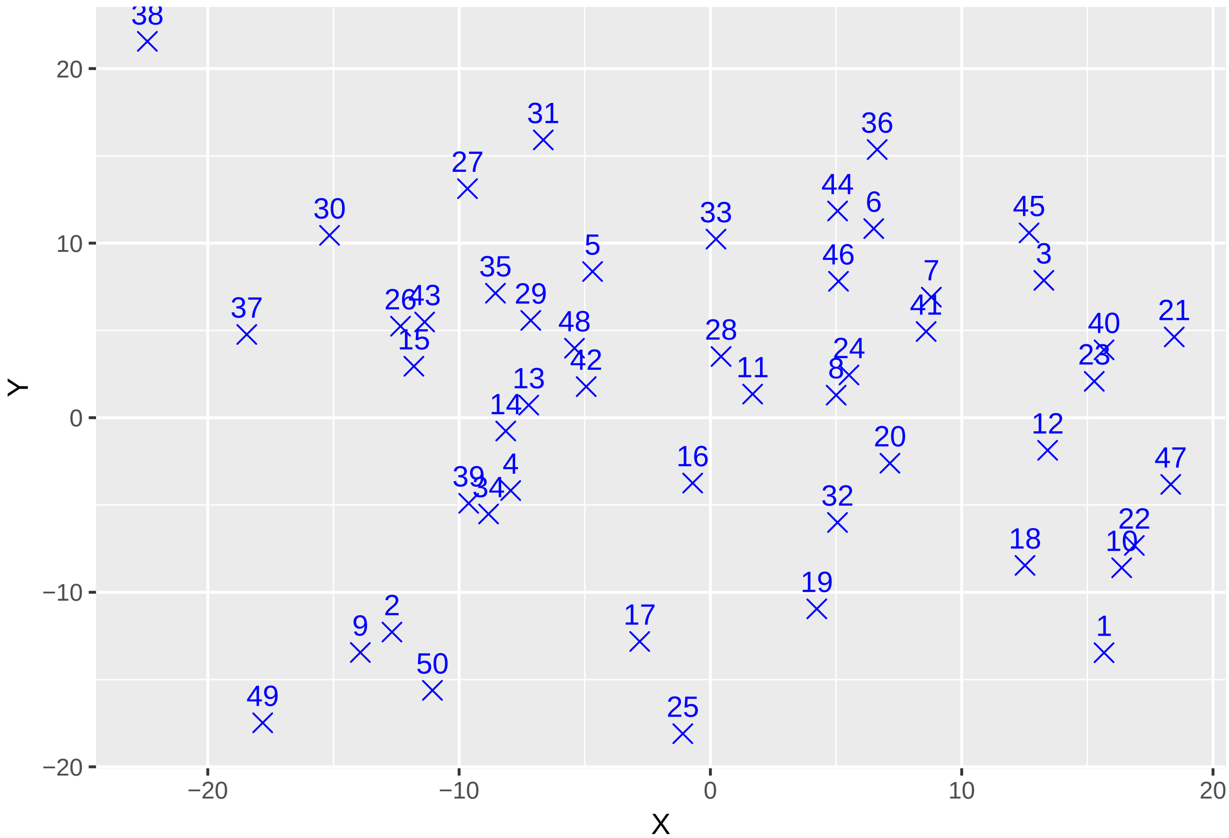
Exercice 2 : Voyageur de commerce

On lit le fichier contenant les coordonnées des villes :

```
coordonnees <- read.csv("Coordonnees-Villes.csv")
```

On affiche alors la disposition des villes :

```
ggplot(data = coordonnees, aes(x = V1, y = V2, label = X)) + geom_point(size = 3, shape = 4, color = "bl
```



La fonction suivante permet de calculer $H(\sigma)$ pour $\sigma = permutation$ à partir des *coordonnees* des villes :

```
distance <- function(permutation, coordonnees)
{
  N <- length(permutation)
  ret <- 0
  for(i in 1:(N - 1)) ret <- ret + sqrt((coordonnees$V1[permutation[i]] - coordonnees$V1[permutation[i +
  ret <- ret + sqrt((coordonnees$V1[permutation[N]] - coordonnees$V1[permutation[1]])^2 + (coordonnees$V
  return(ret)
}
```

La fonction suivante génère une réalisation d'une loi uniforme sur $\{1, ..., N\}$:

```
realisation_loi_uniforme_N <- function(N)
{
  u <- runif(1, 0, 1)
  p <- 1 / N
  i <- 1
  while(u > (i * p) && i < N) i <- i + 1
  return(ifelse(i == N, N, i))
}
```

La fonction suivante génère une permutation ω telle que $\omega \sim \sigma$ avec $\sigma = permutation$:

```
voisin <- function(permutation)
{
  ret <- permutation
  N <- length(permutation)
  i <- realisation_loi_uniforme_N(N)
```

```
  j <- realisation_loi_uniforme_N(N)
  while(i == j)
  {
    i <- realisation_loi_uniforme_N(N)
    j <- realisation_loi_uniforme_N(N)
  }
  x <- which(ret == i)
  y <- which(ret == j)
  ret[x] <- j
  ret[y] <- i
  return(ret)
}
```

La fonction suivante implémente l'algorithme du récuit avec M itérations à partir de *coordonnes* et c :

```
recuit_simule <- function(coordonnees, M, c)
{
  permutation <- coordonnees$X
  for(i in 1:M)
  {
    w <- voisin(permutation)
    u <- runif(1, 0, 1)
    if(u <= exp(-c * log(i + 1) * (distance(w, coordonnees) - distance(permutation, coordonnees)))) perm
  }
  return(permutation)
}
```

Pour $N = 60$, $M = 10000$ et $c = 10$, on trouve la permutation *solution* et sa distance associée suivante :

```
solution <- recuit_simule(coordonnees, 10000, 10)
solution
```

```
##  [1] 14 48 45  3 21 40 18 19 25 17  4 13 42 28 24  8 32 20 12 23 47 22 10
## [24]  1 41  7 36 31 38 30 27 35 29 15 43 26 37  2  9 49 50  5 33 44  6 46
## [47] 11 16 34 39
```

```
distance(solution, coordonnees)
```

```
## [1] 342.9801
```

La fonction suivante permet de générer le chemin indiquant l'ordre dans lequel le voyageur de commerce doit visiter les villes et leurs coordonnées respectives à partir d'une *solution* :

```
chemin <- function(solution, coordonnees)
{
  N <- nrow(coordonnees)
  V1 <- numeric(N)
  V2 <- numeric(N)
  for(i in 1:N)
  {
    V1[i] <- coordonnees$V1[solution[i]]
    V2[i] <- coordonnees$V2[solution[i]]
  }
  return(data.frame(X = solution, V1 = V1, V2 = V2))
}
```

On affiche alors la *solution* :

```
ggplot(data = chemin(solution, coordonnees), aes(x = V1, y = V2, label = X)) + geom_point(size = 3, shap
```