

# TP3 : Chaînes de Markov

Li-Thiao-Té Sébastien

## 1 Simulation exacte

**Introduction** Pour une chaîne ergodique, on a convergence vers la loi stationnaire, mais on est jamais exactement sous la loi invariante. Par une méthode de couplage, Propp et Wilson ont proposé un algorithme pour simuler exactement selon la loi invariante en prenant des trajectoires infinies dans le passé.

**Objets matlab** Étant donné une chaîne de Markov sur  $[1, n]$ , on simule  $n$  trajectoires couplées entre  $-t$  et 0. On suppose que la chaîne  $i$  part de l'état  $i$  au temps  $-t$ . On ne garde que les états finaux des  $n$  chaînes de Markov ; il suffit donc d'un vecteur de taille  $n$  que l'on note  $\mathbf{r}$ .

**Préliminaires** Programmer une fonction `rmarkov(m,i,n)` qui calcule une trajectoire par la chaîne  $m$  en partant de l'état  $i$  avec  $n$  transitions. Pour les simulations, on prendra par exemple  $\mathbf{m} = [0.9 \ 0.05 \ 0.05; 0.1 \ 0.4 \ 0.5; 0.1 \ 0.5 \ 0.4]$ .

**Remonter le temps** A chaque étape  $-t$ ,  $\mathbf{r}$  résume  $n$  trajectoires entre  $-t$  et 0. On construit  $r$  en faisant remonter  $-t$  :

- **Passage de -1 à 0** La chaîne  $i$  est initialement en l'état  $i$  et fait une transition de Markov. On tire `rmarkov(i)` et l'on met à jour  $\mathbf{r}(i)$ . Répéter pour les autres chaînes.
- **Passage de -2 à 0** Pour avoir les trajectoires de  $-2$  jusqu'à 0, on suppose que la chaîne  $i$  part de  $i$  puis fait une transition par `rmarkov(i)` et arrive en  $j$ . Le reste du chemin consiste à passer du temps  $-1$  au temps 0, ce qui est résumé par  $\mathbf{r}(i)$ .
- **Passage de -t à 0** La chaîne  $i$  part de  $i$  et effectue une transition de Markov. On recolle ensuite avec les trajectoires entre  $-t + 1$  et 0.

Implémenter une fonction `rpropp(m, t)` qui simule en remontant jusqu'au temps fixe  $-t$ .

**Coalescence** On ne remonte pas le temps jusqu'à  $-\infty$ . En effet, il existe un temps  $t$  pour lequel les  $n$  chaînes aboutissent au même état  $r$ , on dit que l'on a coalescence des chaînes. Pour tout temps antérieur, les chaînes aboutissent encore au même état  $r$ .  $r$  est distribué selon la loi invariante. Modifier la fonction `rpropp` afin de continuer les simulations jusqu'à coalescence.

### Vérifications et Comparaisons

- calculer la loi invariante (exactement)
- après 15 transitions en partant de l'état 1, est-on sous la loi invariante ? Calculer la loi (exactement).
- simuler avec `rpropp` un grand nombre d'échantillons et en déduire une approximation de la loi de  $r$

Comparer.

## 2 Voyageur de commerce

**Introduction** Un vrp souhaite visiter un nombre connu  $n$  de clients en parcourant le moins de distance possible. On suppose qu'il part de chez lui, effectue son trajet, puis revient chez lui. Il a  $n + 1$  destinations et connaît les  $(n + 1)^2$  distances entre elles. Un chemin possible correspond à la donnée d'une liste de longueur  $n$  indiquant l'ordre des clients dans lequel s'effectue la visite. L'espace des chemins contient  $n!$  éléments, et le problème est NP complet.

**Représentation des objets en matlab** Un chemin est représenté par un vecteur de longueur  $n$  contenant des entiers entre 1 et  $n$ . Les distances entre les maisons sont fournies sous la forme d'une matrice carrée `distances` de taille  $(n + 1)^2$  telle que `distances(i,j)` contient la distance entre la maison  $i$  et la maison  $j$ . La maison  $n + 1$  est la maison du vrp.

**Tirage d'un chemin aléatoire** Écrire une fonction `rchemin(n)` qui tire un chemin uniformément sur l'ensemble des permutations de  $[1, n]$ . Écrire d'abord une fonction `retourne(tab)` qui tire deux indices  $i$  et  $j$  au hasard et retourne le contenu des cases d'indice  $i$  à  $j$  du tableau. Question : pourquoi est-ce une marche aléatoire ?

**Tirage d'un problème au hasard** Il s'agit de tirer un ensemble de destinations et une matrice de distances. Par exemple :

- tirer  $n$  points du plan au hasard dans un carré, puis considérer les distances euclidiennes (conseillé)
- si les rues se croisent à angle droit, tirer des points de coordonnées entières, et considérer les distances  $L_1$

La fonction `rprobleme(n)` renverra une matrice de taille  $(n + 1) * 2$  contenant les coordonnées de chacune des maisons, et une matrice carrée de taille  $(n + 1)^2$  contenant les distances.

### Visualisations

- Écrire une fonction `longueur_chemin(c,distances)` qui calcule la longueur du trajet  $c$ .
- Écrire une fonction `affiche_chemin(c)` qui dessine un chemin avec `plot`.

**Optimisation 1 : Marche aléatoire dirigée** Reprendre le code de la fonction `rchemin`. Après chaque modification, comparer la nouvelle longueur avec l'ancienne longueur. On ne conservera que les modifications bénéfiques. On écrira une fonction `optim_marche(c,N)` en stoppant la simulation après  $N$  appels à `retourne`.

### Optimisation 2 : Hastings-Metropolis

On souhaite simuler une chaîne de Markov de probabilité invariante  $\pi(x) = 1/Z e^{-l(x)/\beta}$ . Quels sont les chemins chargés par  $\pi$  pour différentes valeurs de  $\beta$ ?  $l$  est interprété comme une énergie, et  $\beta$  comme une température.

La fonction `retourne(tab)` définit une marche aléatoire qui est une chaîne de Markov de matrice de transition  $q(x, y)$ . Montrer que  $q(x, y) = q(y, x)$ .

Une chaîne de Markov de loi invariante  $\pi$  est donnée par les transitions :

$$\begin{cases} Q(x, y) &= q(x, y) \left( \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)} \wedge 1 \right) \text{ si } x \neq y \\ Q(x, y) &= 1 - \sum_{z \neq x} Q(x, z) \text{ sinon} \end{cases}$$

Pour simuler la chaîne de Markov, on tire un candidat  $y$ , et on le garde avec probabilité  $\left( \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)} \wedge 1 \right) = (e^{(l(x)-l(y))/\beta} \wedge 1)$ . Un avantage par rapport à la méthode précédente est de pouvoir sortir des minima locaux. Écrire la fonction `optim_metropolis(c,N)`.

**Optimisation 3 : Recuit simulé** Reprendre le code de la fonction `optim_metropolis(c,N)` en faisant varier la température  $\beta$  en fonction du temps. Appelez la fonction `optim_recuit(c,N)`.

**Visualisation** Modifier les trois fonctions précédentes afin qu'elles renvoient le chemin à l'étape  $N$  et un vecteur contenant les longueurs de tous les chemins. Quelques visualisations possibles :

- décroissance de l'énergie pour plusieurs simulations
- affichage dynamique des chemins + du tracé