



Mastermind et permutations

KIM ANTUNEZ, ROMAIN LESAUVAGE ET ALAIN
QUARTIER-LA-TENTE
25/04/2020
Ensaë — 2019-2020

Sommaire

1. Introduction

1.1 Rappels sur la méthode de *Cross-Entropy*

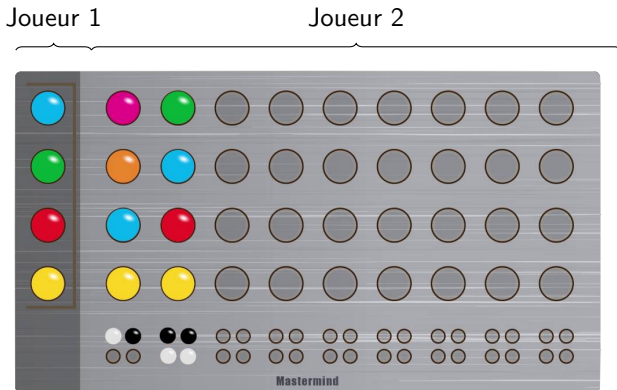
2. Application de la méthode de *Cross-Entropy* au Mastermind (Q1)

3. Restriction aux permutations (Q2)

4. Loi spécifique pour générer les permutations (Q3)

Introduction

- **code** : n boules de couleur parmi m couleurs possibles (classiquement $n = 4$ et $m = 6$)
- **boules noires** : boules bien placées
- **boules blanches** : boules de la bonne couleur, mais mal placées



→ Résolution par *Cross-Entropy*

Objectif de la *Cross-Entropy*

Résoudre ...

$$S(x^*) = \gamma^* = \max_{x \in \mathcal{X}} S(x) \quad (1)$$

... en lui associant un problème stochastique :

- $1_{\{S(x) \geq \gamma\}}$ sur \mathcal{X} pour plusieurs seuils $\gamma \in \mathbb{R}$;
- $\{f(\cdot; \nu), \nu \in \mathcal{V}\}$ famille de probabilités sur \mathcal{X} , paramétrée par ν .

Algorithme

1. **Initialisation** : on fixe \hat{v}_0 , $N \in \mathbb{N}$ et $\rho \in]0, 1[$, $t = 1$.

Algorithme

1. **Initialisation** : on fixe \hat{v}_0 , $N \in \mathbb{N}$ et $\rho \in]0, 1[$, $t = 1$.

2. On génère $X_1, \dots, X_N \sim f(\cdot, v_{t-1})$, on calcule

$$\hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$$

Algorithme

1. **Initialisation** : on fixe \hat{v}_0 , $N \in \mathbb{N}$ et $\rho \in]0, 1[$, $t = 1$.

2. On génère $X_1, \dots, X_N \sim f(\cdot, v_{t-1})$, on calcule

$$\hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$$

3. On utilise le même échantillon X_1, \dots, X_N pour trouver \hat{v}_t :

$$\hat{v}_t = \underset{v}{\operatorname{argmax}} \hat{D}(v) = \underset{v}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} \ln f(X_i; v) \quad (2)$$

Algorithme

1. **Initialisation** : on fixe \hat{v}_0 , $N \in \mathbb{N}$ et $\rho \in]0, 1[$, $t = 1$.

2. On génère $X_1, \dots, X_N \sim f(\cdot, v_{t-1})$, on calcule

$$\hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$$

3. On utilise le même échantillon X_1, \dots, X_N pour trouver \hat{v}_t :

$$\hat{v}_t = \underset{v}{\operatorname{argmax}} \hat{D}(v) = \underset{v}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} \ln f(X_i; v) \quad (2)$$

5. **Arrêt** : si pour un certain $t \geq d$, on a :

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \dots = \hat{\gamma}_{t-d}$$

Algorithme

1. **Initialisation** : on fixe \hat{v}_0 , $N \in \mathbb{N}$ et $\rho \in]0, 1[$, $t = 1$.

2. On génère $X_1, \dots, X_N \sim f(\cdot, v_{t-1})$, on calcule

$$\hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$$

3. On utilise le même échantillon X_1, \dots, X_N pour trouver \hat{v}_t :

$$\hat{v}_t = \underset{v}{\operatorname{argmax}} \hat{D}(v) = \underset{v}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} \ln f(X_i; v) \quad (2)$$

4. *[FACULTATIF] smoothed updating* (éviter l'occurrence de 0 et de 1) :

$$\hat{v}_t = \alpha \tilde{v}_t + (1 - \alpha) \hat{v}_{t-1}$$

5. **Arrêt** : si pour un certain $t \geq d$, on a :

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \dots = \hat{\gamma}_{t-d}$$

Sommaire

1. Introduction

2. Application de la méthode de *Cross-Entropy* au Mastermind (Q1)

2.1 Paramètres utilisés

2.2 Algorithme de *Cross-Entropy*

2.3 Résultats

3. Restriction aux permutations (Q2)

4. Loi spécifique pour générer les permutations (Q3)

Question 1

Mettre en oeuvre un algorithme basé sur la méthode CE en détaillant :

1. la fonction score choisie ;
2. la famille paramétrique choisie (pour simuler des codes) ;
3. la méthode pour simuler une loi de cette famille ;
4. la méthode utilisée pour estimer le paramètre « optimal » à chaque étape.

Paramètres utilisés

- $\mathcal{X} = \{1, 2, \dots, m\}^n$

Paramètres utilisés

- $\mathcal{X} = \{1, 2, \dots, m\}^n$
- Famille paramétrique :

$$\mathcal{V} = \left\{ (p_{i,j})_{i,j} \in \mathcal{M}_{n,m}([0, 1]) : \forall i, \sum_{j=1}^m p_{i,j} = 1 \right\}$$

Remarque : $X = (X_1, \dots, X_n) \in \mathcal{X}$ tirées aléatoirement selon p_1, \dots, p_n , la j^{e} composante de p_i étant égale à $p_{ij} = \mathbb{P}(X_i = j)$: probabilité d'avoir une boule de couleur j en i^{e} position.

Paramètres utilisés

- $\mathcal{X} = \{1, 2, \dots, m\}^n$
- Famille paramétrique :

$$\mathcal{V} = \left\{ (p_{i,j})_{i,j} \in \mathcal{M}_{n,m}([0, 1]) : \forall i, \sum_{j=1}^m p_{i,j} = 1 \right\}$$

Remarque : $X = (X_1, \dots, X_n) \in \mathcal{X}$ tirées aléatoirement selon p_1, \dots, p_n , la j^{e} composante de p_i étant égale à $p_{ij} = \mathbb{P}(X_i = j)$: probabilité d'avoir une boule de couleur j en i^{e} position.

- Score :

$$S(x) = \frac{\omega_{\text{noir}} \times N_{\text{boules noires}} + \omega_{\text{blanc}} \times N_{\text{boules blanches}}}{\omega_{\text{noir}} \times n}$$

avec, par exemple, $\omega_{\text{noir}} = 2$ et $\omega_{\text{blanc}} = 1$.

Algorithme de *Cross-Entropy*

1. Initialisation

- $\hat{v}_0 = \left(\frac{1}{m}\right)_{i=1..n, j=1..m}$ (probabilités uniformes pour chaque couleur)
- $N = C \times m \times n$ ($C = 5$ par défaut)
- $\rho = 0,1$ (maximisation réalisée sur les 10 % meilleurs échantillons)

2. $X_1, \dots, X_N \sim f(\cdot, v_{t-1}), \hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$

3. Trouver \hat{v}_t , qui correspond ici à la matrice de terme :

$$p_{k,l} = \frac{\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} 1_{\{X_{i,k}=l\}}}{\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}}} \quad (3)$$

4. *[FACULTATIF] smoothed updating*

5. Arrêt : si pour un certain $t \geq d$ ($d = 5$), on a : $\hat{\gamma}_t = \dots = \hat{\gamma}_{t-d}$

Premiers résultats

Application web interactive

Menu

Simulations

Rapport

Résultats

Tableau

Convergence à l'itération n°9 (d=5)

Critère d'arrêt à l'itération n°4

Itération n°4



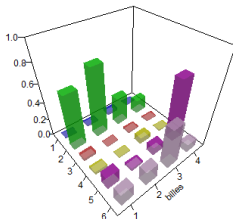
Gamma : 1

Smax : 1

min : 0.0045 / max_min : 0.0254

max : 0.7688 / min_max : 0.4577

Meilleure proposition :



Couleurs (m)

6

Billes (n)

4

Combinaison
cachée (y) :

Changer combinaison

Paramètres du modèle

Méthode

- ☒ Tirage avec remise
- ☐ Tirage sans remise
- ☐ Loi avec distance de Hamming

Nombre d'itérations (maxiters)



d

5

N

120

rho

0,1

alpha

0,7

ON

smoothing

Résultats sur de nombreuses simulations

Itération médiane
de convergence

n / m	4	6	10	15	20	30	40
4	8	8	9	10,0	9,5	10,0	10,0
6	9	10	11	11,0	11,0	11,0	12,0
10	11	12	13	13,5	14,0	14,0	14,0
15	12	13	15	16,0	17,0	17,5	18,0
20	13	14	16	18,0	19,5	19,5	19,5
30	14	16	19	21,0	23,0	24,0	24,0
40	16	17	19	22,0	23,0	27,0	27,0

Nombre de simulations
n'ayant pas convergé
vers la bonne valeur

n / m	4	6	10	15	20	30	40
4	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
15	0	0	1	1	1	0	0
20	0	1	1	0	1	1	0
30	1	1	3	3	4	1	2
40	1	0	2	4	6	3	3

Moyenne de l'erreur
à la simulation
de convergence

n / m	4	6	10	15	20	30	40
4	0,000	0,000	0,000	0,000	0,000	0,000	0,000
6	0,000	0,000	0,000	0,000	0,000	0,000	0,000
10	0,000	0,000	0,000	0,000	0,000	0,000	0,000
15	0,000	0,000	0,003	0,003	0,003	0,000	0,000
20	0,000	0,005	0,005	0,000	0,005	0,005	0,000
30	0,003	0,003	0,010	0,005	0,008	0,002	0,007
40	0,003	0,000	0,005	0,005	0,009	0,004	0,004

Moyenne du temps de
calcul jusqu'à
la convergence

n / m	4	6	10	15	20	30	40
4	0	0	0	0	0	0	0
6	0	0	0	0	0	1	1
10	0	0	1	1	1	2	3
15	0	1	1	2	3	6	8
20	1	1	2	4	6	11	16
30	1	3	6	11	17	30	43
40	3	5	11	21	31	60	88

Sommaire

1. Introduction

2. Application de la méthode de *Cross-Entropy* au Mastermind (Q1)

3. Restriction aux permutations (Q2)

3.1 Adaptation de l'algorithme précédent

3.2 Résultats

4. Loi spécifique pour générer les permutations (Q3)

Question 2

Le Joueur 1 choisit obligatoirement une **permutation** (chaque couleur ne peut apparaître qu'une seule fois, donc $m \geq n$).

1. Mettre en oeuvre l'algorithme précédent en l'adaptant ;
2. La méthode d'estimation utilisée dans la question précédente est toujours valide ?

Adaptation du mécanisme de génération...

Par rapport à l'algorithme précédent, on ne change que le **mécanisme de génération** des échantillons :

Adaptation du mécanisme de génération...

Par rapport à l'algorithme précédent, on ne change que le **mécanisme de génération** des échantillons :

Les échantillons sont désormais générés grâce à une loi « sans remise ».

- Initialisation : on tire la première boule, entier x_1 , selon la loi discrète donnée par $p_{1,\cdot} = (p_{1,1}, \dots, p_{1,m})$. On pose $k = 1$ et $P^{(1)} = P$.

Adaptation du mécanisme de génération...

Par rapport à l'algorithme précédent, on ne change que le **mécanisme de génération** des échantillons :

Les échantillons sont désormais générés grâce à une loi « sans remise ».

- Initialisation : on tire la première boule, entier x_1 , selon la loi discrète donnée par $p_{1,\cdot} = (p_{1,1}, \dots, p_{1,m})$. On pose $k = 1$ et $P^{(1)} = P$.
- Itération : $P^{(k+1)}$ est obtenue en remplaçant la colonne k de $P^{(k)}$ par 0 et en normalisant les lignes (somme = 1). On tire x_{k+1} selon la loi discrète donnée par la ligne $k + 1$ de $P^{(k+1)}$.

Adaptation du mécanisme de génération...

Par rapport à l'algorithme précédent, on ne change que le **mécanisme de génération** des échantillons :

Les échantillons sont désormais générés grâce à une loi « sans remise ».

- Initialisation : on tire la première boule, entier x_1 , selon la loi discrète donnée par $p_{1,\cdot} = (p_{1,1}, \dots, p_{1,m})$. On pose $k = 1$ et $P^{(1)} = P$.
- Itération : $P^{(k+1)}$ est obtenue en remplaçant la colonne k de $P^{(k)}$ par 0 et en normalisant les lignes (somme = 1). On tire x_{k+1} selon la loi discrète donnée par la ligne $k + 1$ de $P^{(k+1)}$.
- Si $k = n$ alors on arrête, sinon on pose $k = k + 1$ et on répète l'étape 2.

... Mais la méthode d'estimation reste la même

Les $p_{k,l}$ s'interprètent de la même façon qu'en question 1 et la formule de mise à jour des paramètres s'écrit :

$$p_{k,l} = \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_{i,k}=l\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}$$

... Mais la méthode d'estimation reste la même

Les $p_{k,l}$ s'interprètent de la même façon qu'en question 1 et la formule de mise à jour des paramètres s'écrit :

$$p_{k,l} = \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_{i,k}=l\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}$$

Possibilité d'appliquer la méthode de génération des échantillons de la **question 1** ...

... Mais la méthode d'estimation reste la même

Les $p_{k,l}$ s'interprètent de la même façon qu'en question 1 et la formule de mise à jour des paramètres s'écrit :

$$p_{k,l} = \frac{\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} 1_{\{X_{i,k}=l\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}{\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}$$

Possibilité d'appliquer la méthode de génération des échantillons de la **question 1** ...

... mais dans ce cas, beaucoup d'échantillons non pertinents (les non-permutations)

... Mais la méthode d'estimation reste la même

Les $p_{k,l}$ s'interprètent de la même façon qu'en question 1 et la formule de mise à jour des paramètres s'écrit :

$$p_{k,l} = \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_{i,k}=l\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} \mathbf{1}_{\{X_i \text{ permutation}\}}}$$

Possibilité d'appliquer la méthode de génération des échantillons de la **question 1** ...

... mais dans ce cas, beaucoup d'échantillons non pertinents (les non-permutations)

—> Algorithme de la **question 2** améliore le processus en ne conservant que les permutations (telles que $\mathbf{1}_{\{X_i \text{ permutation}\}} = 1$)

Résultats

Itération médiane
de convergence

➔ **converge + vite...**

n / m	4	6	10	15	20	30	40
4	7	8	9,0	9	9	10	10
6		8	9,5	10	10	11	11
10			10,0	12	12	13	13
15				12	14	15	15
20					14	17	17
30						18	19
40							21

Nombre de simulations
n'ayant pas convergé
vers la bonne valeur

n / m	4	6	10	15	20	30	40
4	0	0	0	0	0	0	0
6		0	0	0	0	0	0
10			0	0	0	0	0
15				0	0	0	0
20					0	0	0
30						0	0
40							0

Moyenne de l'erreur
à la simulation
de convergence

n / m	4	6	10	15	20	30	40
4	0	0	0	0	0	0	0
6		0	0	0	0	0	0
10			0	0	0	0	0
15				0	0	0	0
20					0	0	0
30						0	0
40							0

Moyenne du temps de calcul
jusqu'à la convergence

➔ **... mais est + gourmand**

n / m	4	6	10	15	20	30	40
4	0	0	0	0	0	1	1
6		0	0	1	1	2	3
10			1	2	3	6	9
15				5	9	15	22
20					15	29	44
30						69	109
40							211

Sommaire

1. Introduction

2. Application de la méthode de *Cross-Entropy* au Mastermind (Q1)

3. Restriction aux permutations (Q2)

4. Loi spécifique pour générer les permutations (Q3)

4.1 Application de l'algorithme de CE

4.2 Génération des données

4.3 Résultats

Question 3

Considérons désormais la loi suivante sur l'ensemble des permutations :

$$\pi_{\lambda, x^*}(x) \propto \exp(-\lambda d(x, x^*))$$

avec :

- $\lambda > 0$
 - $d(x, x^*)$: distance de Hamming (nombre de positions où les deux permutations x et x^* diffèrent).
1. Proposer un algorithme de MCMC pour simuler selon une telle loi ;
 2. Utiliser cet algorithme au sein d'une approche CE ;
 3. Comparer la performance de l'algorithme obtenu à l'algorithme proposé en question 1.

Adaptation de l'algorithme de CE

1. Initialisation

- *Tirage aléatoire de x_0^* et $\lambda_0 = 1$.*
- $N = C \times (n + 1)$ ($C = 5$ par défaut)
- $\rho = 0,1$ (maximisation réalisée sur les 10 % meilleurs échantillons)

2. X_1, \dots, X_N générés avec la loi π_{λ_t, x_t^*} (Metropolis-Hastings), calcul de $\hat{\gamma}_t = S_{\lceil (1-\rho)N \rceil}$

3. *Trouver \tilde{x}_{t+1} . Si $S(\tilde{x}_{t+1}) \geq S(x_t^*)$ alors $x_{t+1}^* = \tilde{x}_{t+1}$, sinon $x_{t+1}^* = x_t^*$. On fixe $\lambda_{t+1} = 1$*

4. *[FACULTATIF] Pas de smoothed updating*

5. *Arrêt : si pour un certain t , $S(x_t^*) = 1$ alors on arrête l'algorithme.*

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m = n$ (vraies permutations)

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m = n$ (vraies permutations)

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.
- Itération :
 - On permute au hasard deux éléments i et j de x_t et on note x' la nouvelle permutation.

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m = n$ (vraies permutations)

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.
- Itération :
 - On permute au hasard deux éléments i et j de x_t et on note x' la nouvelle permutation.
 - On calcule la probabilité d'acceptation :
$$r(x', x_t) = \min \left(1, \frac{\pi_{\lambda, x^*}(x')}{\pi_{\lambda, x^*}(x_t)} \right) = \min \left(1, e^{-\lambda(d(x', x^*) - d(x_t, x^*))} \right)$$

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m = n$ (vraies permutations)

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.
- Itération :
 - On permute au hasard deux éléments i et j de x_t et on note x' la nouvelle permutation.
 - On calcule la probabilité d'acceptation :
$$r(x', x_t) = \min \left(1, \frac{\pi_{\lambda, x^*}(x')}{\pi_{\lambda, x^*}(x_t)} \right) = \min \left(1, e^{-\lambda(d(x', x^*) - d(x_t, x^*))} \right)$$
 - Acceptation ou rejet : on génère une loi uniforme $u \in [0, 1]$. Si $u \leq r(x', x_t)$ alors on accepte le nouvel état et on pose $x_{t+1} = x'$, sinon $x_{t+1} = x_t$.

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m = n$ (vraies permutations)

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.
- Itération :
 - On permute au hasard deux éléments i et j de x_t et on note x' la nouvelle permutation.
 - On calcule la probabilité d'acceptation :
$$r(x', x_t) = \min \left(1, \frac{\pi_{\lambda, x^*}(x')}{\pi_{\lambda, x^*}(x_t)} \right) = \min \left(1, e^{-\lambda(d(x', x^*) - d(x_t, x^*))} \right)$$
 - Acceptation ou rejet : on génère une loi uniforme $u \in [0, 1]$. Si $u \leq r(x', x_t)$ alors on accepte le nouvel état et on pose $x_{t+1} = x'$, sinon $x_{t+1} = x_t$.
 - Incrémentation : $t = t + 1$.

Génération de l'échantillon : Metropolis-Hastings

➡ **mécanisme de proposition** : inverser deux éléments de la permutation (symétrique).

Pour $m > n$

- Initialisation : on choisit x_0 une permutation au hasard et on fixe $t = 0$.
- Itération :
 - On permute au hasard deux éléments i et j de x_t et on note x' la nouvelle permutation. → i ou $j \leq n$
 - On calcule la probabilité d'acceptation :
$$r(x', x_t) = \min \left(1, \frac{\pi_{\lambda, x^*}(x')}{\pi_{\lambda, x^*}(x_t)} \right) = \min \left(1, e^{-\lambda(d(x', x^*) - d(x_t, x^*))} \right)$$
→ *distances calculées sur n premières coordonnées*
 - Acceptation ou rejet : on génère une loi uniforme $u \in [0, 1]$. Si $u \leq r(x', x_t)$ alors on accepte le nouvel état et on pose $x_{t+1} = x'$, sinon $x_{t+1} = x_t$.
 - Incrémentation : $t = t + 1$.

Génération de l'échantillon : Metropolis-Hastings

Remarque : Pour λ grand on converge vers x^* et tous les échantillons seront égaux à x^* .

Génération de l'échantillon : Metropolis-Hastings

Remarque : Pour λ grand on converge vers x^* et tous les échantillons seront égaux à x^* .

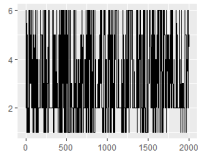


L'algorithme de Metropolis-Hastings a **deux désavantages** :

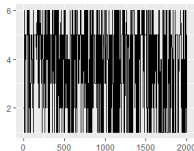
1. Le *burn-in*.
2. Les échantillons générés à des périodes proches sont corrélés entre eux.

Gestion du burn-in

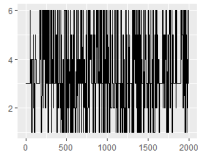
Trace 1ère coordonnée



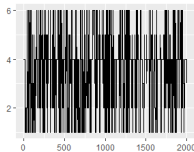
Trace 2ème coordonnée



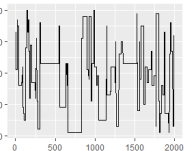
Trace 3ème coordonnée



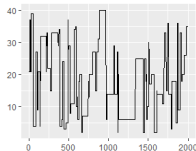
Trace 4ème coordonnée

FIGURE 1 - $n = 4$, $m = 6$

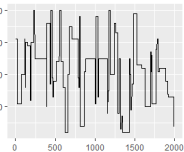
Trace 1ère coordonnée



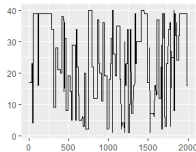
Trace 2ème coordonnée



Trace 3ème coordonnée



Trace 4ème coordonnée

FIGURE 2 - $n = 40$, $m = 40$

→ burn-in $\simeq 250 \times m$

Gestion des autocorrélations

Autocorrélogrammes des 4 premières composantes des échantillons ($\lambda = 1$, $n = 10$ et $m = 40$)

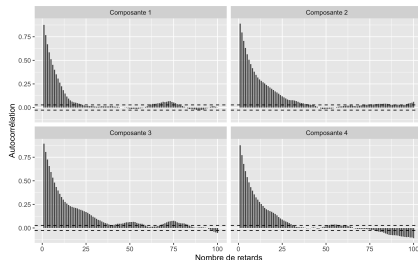


FIGURE 3 – Toutes les autocorrélations

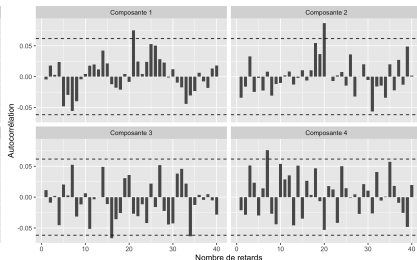



FIGURE 4 – Pas de 80 périodes


→ Conserver uniquement les simulations séparées de $t = 80$ périodes.

→ Intégrer ce test dans le mécanisme de proposition est **très coûteux en temps**.

Estimation de x^*

 (Arrieta, 2014) montre qu'on peut séparer le problème d'estimation de x^* et λ

Estimation de x^*

 (Arrieta, 2014) montre qu'on peut séparer le problème d'estimation de x^* et λ

Trouver le x^* qui minimise la somme $\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)$

Estimation de x^*




(Arrieta, 2014) montre qu'on peut séparer le problème d'estimation de x^* et λ

Trouver le x^* qui minimise la somme $\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)$

- Intuitivement, il s'agit de $x^* = (x_1^*, \dots, x_n^*)$ tel que x_j^* correspond au chiffre le plus fréquent dans la j ème coordonnée des 10 % meilleurs échantillons.

Estimation de x^*

 (Arrieta, 2014) montre qu'on peut séparer le problème d'estimation de x^* et λ

Trouver le x^* qui minimise la somme $\sum_{i=1}^N 1_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)$

- Intuitivement, il s'agit de $x^* = (x_1^*, \dots, x_n^*)$ tel que x_j^* correspond au chiffre le plus fréquent dans la j ème coordonnée des 10 % meilleurs échantillons.
- On part de ce principe en imposant que x^* soit bien une permutation :
 1. On crée une matrice $F = (f_{i,j}) \in \mathcal{M}_{n,m}(\mathbb{N})$ telle $f_{i,j}$ soit égal au nombre de fois que l'entier j apparait en i ème position parmi les 10 % meilleurs échantillon.
 2. On sélectionne une composante par ligne et par colonne de F de façon à ce que leur somme soit maximale

Estimation de λ

Plusieurs tests réalisés pour estimer λ :

1. (Arrieta, 2014) nous fournit la constante de normalisation de π_{λ, x^*} :

$$m! \exp(-\lambda m) \sum_{k=0}^m \frac{(\exp(\lambda) - 1)^k}{k!}$$

le λ qui maximise la vraisemblance est alors tel que :

$$\frac{\exp(\lambda) \sum_{k=0}^{m-1} \frac{(\exp(\lambda)-1)^k}{k!} - m \sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}}{\sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}} + \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}}} = 0$$

Problème : croissance rapide de λ à chaque itération. Si y n'est pas décodé dans les premières itérations, les échantillons X_i générés seront très proches de x^* et on ne trouvera pas y .

Estimation de λ

Plusieurs tests réalisés pour estimer λ :

1. (Arrieta, 2014) nous fournit la constante de normalisation de π_{λ, x^*} :

$$m! \exp(-\lambda m) \sum_{k=0}^m \frac{(\exp(\lambda) - 1)^k}{k!}$$

le λ qui maximise la vraisemblance est alors tel que :

$$\frac{\exp(\lambda) \sum_{k=0}^{m-1} \frac{(\exp(\lambda)-1)^k}{k!} - m \sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}}{\sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}} + \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}}} = 0$$

Problème : croissance rapide de λ à chaque itération. Si y n'est pas décodé dans les premières itérations, les échantillons X_i générés seront très proches de x^* et on ne trouvera pas y .

2. croissance linéaire à chaque itération

Estimation de λ

Plusieurs tests réalisés pour estimer λ :

1. (Arrieta, 2014) nous fournit la constante de normalisation de π_{λ, x^*} :

$$m! \exp(-\lambda m) \sum_{k=0}^m \frac{(\exp(\lambda) - 1)^k}{k!}$$

le λ qui maximise la vraisemblance est alors tel que :

$$\frac{\exp(\lambda) \sum_{k=0}^{m-1} \frac{(\exp(\lambda)-1)^k}{k!} - m \sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}}{\sum_{k=0}^m \frac{(\exp(\lambda)-1)^k}{k!}} + \frac{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)}{\sum_{i=1}^N \mathbf{1}_{\{S(X_i) \geq \hat{\gamma}_t\}}} = 0$$

Problème : croissance rapide de λ à chaque itération. Si y n'est pas décodé dans les premières itérations, les échantillons X_i générés seront très proches de x^* et on ne trouvera pas y .

2. croissance linéaire à chaque itération
3. constance

→ **Meilleure solution** : $\lambda = 1$.

Résultats $N = 5 \times (n + 1)$

Itération médiane
de convergence

➡ **converge - vite**

n / m	4	6	10	15	20	30	40
4	14,5	4	8,5	17	25,5	21,5	91
6		6	15,0	19	90,0		
10			9,0	31	73,5		
15				14	49,0		
20					23,0		
30						95,0	
40							

Nombre de simulations
n'ayant pas convergé
vers la bonne valeur

➡ **converge - souvent**

n / m	4	6	10	15	20	30	40
4	0	0	0	0	4	8	6
6		0	0	2	6	10	10
10			0	3	6	10	10
15				0	7	10	10
20					0	10	10
30						7	10
40							10

Moyenne de l'erreur
à la simulation
de convergence


n / m	4	6	10	15	20	30	40
4	0,025	0,312	0,462	0,525	0,550	0,662	0,725
6		0,208	0,425	0,492	0,567	0,633	0,725
10			0,295	0,440	0,530	0,640	0,700
15				0,360	0,460	0,587	0,667
20					0,385	0,528	0,622
30						0,425	0,535
40							0,450


Moyenne du temps de calcul
jusqu'à la convergence

➡ **et est + gourmand**

n / m	4	6	10	15	20	30	40
4	1	0	2	4	6	5	26
6		1	3	5	19		
10			2	10	15		
15				5	18		
20					12		
30						64	
40							

Merci pour votre attention

 ARKEnsae/Mastermind_Simulation

 Application web : antuki.shinyapps.io/mastermind

 Rapport du projet

