Projet Monte-Carlo, Ensae

Mastermind et permutations

Kim Antunez, Romain Lesauvage et Alain Quartier-la-Tente

25/04/2020

Ensae — 2019-2020

Sommaire

Introduction

Le Mastermind, jeu à deux joueurs où :

- Joueur 1 choisit un *code* : n boules de couleur parmi m couleurs possibles (classiquement $n = 4$ et $m = 6$)
- Joueur doit deviner ce code en un minimum de coups. À chaque coup il propose un code et J1 donne :
  - nombre de boules bien placées = *nombre de boules noires*
  - nombre de boules de la bonne couleur, mais mal placées = *nombre de boules blanches*

Introduction

Le Mastermind, jeu à deux joueurs où :

- Joueur 1 choisit un *code* : n boules de couleur parmi m couleurs possibles (classiquement $n = 4$ et $m = 6$)
- Joueur doit deviner ce code en un minimum de coups. À chaque coup il propose un code et J1 donne :
  - nombre de boules bien placées = *nombre de boules noires*
  - nombre de boules de la bonne couleur, mais mal placées = *nombre de boules blanches*

$\longrightarrow$ Résolution par *Cross-Entropy*

Rappels sur la méthode de *Cross-Entropy*

Soit $\mathcal{X}$ un ensemble fini d'états et S une fonction de score, on cherche le maximum de S sur $\mathcal{X}$:

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

Pour le résoudre, on lui associe un problème stochastique, en définissant :

- un ensemble d'indicatrices $1_{\{S(x) \geq \gamma\}}$ sur $\mathcal{X}$ pour plusieurs seuils $\gamma \in \mathbb{R}$;
- $\{f(\cdot; v),\ v \in \mathcal{V}\}$ une famille discrète de probabilités sur $\mathcal{X}$, paramétrée par un paramètre vectoriel v.

Pour $u \in \mathcal{V}$, le problème est équivalent au problème d'estimation de la probabilité d'un événement rare :

$$\mathbb{P}_u(S(X) \geq \gamma) = \sum_x 1_{\{S(x) \geq \gamma\}} f(x; u) = \mathbb{E}_u[1_{\{S(x) \geq \gamma\}}]$$

Algorithme utilisé

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

Algorithme utilisé

1. **Initialisation :** on fixe arbitrairement $\hat{v}_0$, deux paramètres $N \in \mathbb{N}$ et $\rho \in]0,1[$ (ici $\rho = 0,1$), $t = 1$.
2. On génère un échantillon $X_1, \ldots, X_N$ de loi $f(\cdot, v_{t-1})$, on calcule le quantile $(1-\rho)$ de la fonction score qui donne $\hat{\gamma}_t$:

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

Si $\hat{\gamma}_t \geq \gamma^*$ on prend $\hat{\gamma}_t = \gamma^*$.

Algorithme utilisé

1. **Initialisation :** on fixe arbitrairement $\hat{v}_0$, deux paramètres $N \in \mathbb{N}$ et $\rho \in]0,1[$ (ici $\rho = 0,1$), $t = 1$.
2. On génère un échantillon $X_1, \ldots, X_N$ de loi $f(\cdot, v_{t-1})$, on calcule le quantile $(1-\rho)$ de la fonction score qui donne $\hat{\gamma}_t$:

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

   Si $\hat{\gamma}_t \geq \gamma^*$ on prend $\hat{\gamma}_t = \gamma^*$.
3. On utilise le même échantillon $X_1, \ldots, X_N$ pour trouver $\hat{v}_t$:

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

Algorithme utilisé

1. **Initialisation :** on fixe arbitrairement $\hat{v}_0$, deux paramètres $N \in \mathbb{N}$ et $\rho \in]0,1[$ (ici $\rho = 0,1$), $t = 1$.
2. On génère un échantillon $X_1, \ldots, X_N$ de loi $f(\cdot, v_{t-1})$, on calcule le quantile $(1-\rho)$ de la fonction score qui donne $\hat{\gamma}_t$:

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

   Si $\hat{\gamma}_t \geq \gamma^*$ on prend $\hat{\gamma}_t = \gamma^*$.
3. On utilise le même échantillon $X_1, \ldots, X_N$ pour trouver $\hat{v}_t$:

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

4. Arrêt : si pour un certain $t \geq d$, (ici $d = 5$), on a :

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

   alors on arrête l'algorithme.

Smoothed updating

Plutôt que de mettre à jour directement $\hat{v}_{t-1}$ l'équation, nous faisons une mise à jour lissée – *smoothed updating* :

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

avec $\tilde{v}_t$ la valeur obtenue en résolvant le problème d'optimisation.

Intérêt : éviter l'occurence de 0 et de 1

Sommaire

Paramètres utilisés dans le projet

La fonction S de score correspond à la réponse du joueur 1 : plus il est grand plus le joueur 2 est proche de la bonne réponse. Pour toute proposition x on a :

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

Habituellement $\omega_{noir} = 2$ et $\omega_{blanc} = 1$.

Dans l'algorithme de *Cross-Entropy*, $\rho = 0,1$ (la maximisation est donc faite sur les 10 % meilleurs échantillons),
$N = C \times$ nombre de paramètres à estimer (avec $C = 5$ par défaut) et $d = 5$.

Application de la méthode de *Cross-Entropy*

Mastermind : choix de n boules parmi m couleurs, on les numérote de 1 à m.

- $\mathcal{X} = \{1, 2, \ldots, m\}^n$
- Génération des échantillons :

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

- $X = (X_1, \ldots, X_n) \in \mathcal{X}$ tirées aléatoirement selon $p_1, \ldots, p_n$, 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 $i$ème position.
- Initialisation : vecteurs de probabilité uniformes pour chaque couleur

$$\hat{v}_0 = \left(\frac{1}{m} \right)_{i=1..n, j=1..m}$$

- Estimation : $p_{k,l} = \dfrac{\sum_{i=1}^{N} 1_{\{S(X_i) \geq \hat{\gamma}_t\}} 1_{\left\{X_{i,k}=l\right\}}}{\sum_{i=1}^{N} 1_{\{S(X_i) \geq \hat{\gamma}_t\}}}$

Résultats

TABLE 1 – Médiane du numéro de simulation de convergence

		m						
		4	6	10	15	20	30	40
n	**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

Note :
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 2 – Moyenne de l'erreur à la simulation de convergence

		m						
		4	6	10	15	20	30	40
n	**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

Note :
L'erreur est définie comme 1 - gamma_T
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 3 – Nombre de simulations n'ayant pas convergé vers la bonne valeur

		m						
		4	6	10	15	20	30	40
n	**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

Note :
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 4 – Moyenne du temps de calcul jusqu'à la convergence (en secondes)

		m						
		4	6	10	15	20	30	40
n	**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

Note :
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Sommaire

Adaptation de l'algorithme précédent

Joueur 1 choisit obligatoirement une **permutation**. On adapte alors l'algorithme :

- Initialisation : la première boule est générée en tirant un entier x_1 selon la loi de probabilité discrète donnée par $p_{1,\cdot} = (p_{1,1}, \ldots, p_{1,m})$. 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 pour que leur somme valent 1. x_{k+1} est alors obtenu en faisant un tirage d'une 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 répéte l'étape 2.

Les autres étapes de l'algorithme restent les mêmes.

Estimation

- La méthode d'estimation upour mettre à jour les $p_{i,j}$ reste la même.
- $P = (p_{i,j})$ s'interprète de la même façon que précédemment : la loi des X_i est la même.
- La formule de mise à jour des paramètres s'écrit :

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

Il est donc possible de mettre à jour les paramètres en appliquant la méthode de génération des échantillons de la partie mais beaucoup d'échantillons ne seraient plus pertinents : le nouvel algorithme de génération permet juste d'améliorer le processus de génération en ne proposant que des permutations, on a $1_{\{X_i \text{ permutation}\}} = 1$.

Résultats

TABLE 5 – Médiane du numéro de simulation de convergence

		m						
		4	6	10	15	20	30	40
n	**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

Note :
S'il n'y a pas convergence les statistiques ne sont pas calculées
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 6 – Moyenne de l'erreur à la simulation de convergence

		m						
		4	6	10	15	20	30	40
n	**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

Note :
S'il n'y a pas convergence les statistiques ne sont pas calculées
L'erreur est définie comme 1 - gamma_T
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 7 – Nombre de simulations n'ayant pas convergé vers la bonne valeur

		m						
		4	6	10	15	20	30	40
n	**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

Note :
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 8 – Moyenne du temps de calcul jusqu'à la convergence (en secondes)

		m						
		4	6	10	15	20	30	40
n	**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

Note :
S'il n'y a pas convergence les statistiques ne sont pas calculées
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Comparaison questions 1 et 2

Le nombre d'itérations nécessaires pour converger est plus faible dans la méthode de la question 2, en adaptant l'algorithme pour ne tirer que des permutations, mais l'algorithme est plus gourmand en temps de calcul.

Cela vient du mécanisme utilisé pour tiré les échantillons qui a une complexité plus importante.

Utilisation d'une loi spécifique pour générer les permutations

Loi sur l'ensemble des permutations : $\pi_{\lambda,x^*}(x) \propto \exp\left(-\lambda d(x, x^*)\right)$

Pour générer les échantillons on utilise l'algorithme de Metropolis-Hastings.

Pour la mise en oeuvre de la méthode de *Cross-Entropy*, on va mettre à jour λ et x^* à chaque itération.

Le critère d'arrêt qui est utilisé est $x^* = y$ (i.e. $S(x^*) = 1$).

Nous avons ici $n+1$ paramètres à estimer, nous générons donc $N = C \times (n+1)$ échantillons à chaque itération de la *Cross-Entropy*.

Algorithme utilisé

- Initialisation : on tire aléatoire x_0^* et on prend $\lambda_0 = 1$.
- On génère un échantillon $X_1, \ldots, X_N$, $N = 5 \times (n+1)$, de loi π_{λ_t, x_t^*}. On calcule le quantile 0,90 de la fonction score qui donne $\hat{\gamma}_t$: $\hat{\gamma}_t = S_{\lceil 0.9N \rceil}$
- On utilise le même échantillon $X_1, \ldots, X_N$ pour 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 $\lambda_{t+1} = 1$
- Arrêt : si pour un certain t, $S(x_t^*) = 1$ alors on arrête l'algorithme.

Génération de l'échantillon : Metropolis-Hastings

Algorithme de Metropolis-Hastings avec le mécanisme de proposition suivant : inverser deux éléments de la permutation (symétrique).

Pour $m = n$, les X_i sont des vraies permutations sur $\{1, \ldots, m\}$.

- Initialisation : on choisit x_0 une permutation au hasard de $\{1, \ldots, m\}$ et on fixe $t = 0$.
- Itération :
  - On permute au hasard deux éléments de x_t et on note x' la nouvelle permutation (on fait donc une transposition de x_t).
  - On calcule la probabilité d'acceptation :
    $r(x', x_t) = \min\left(1, \frac{\pi_{\lambda, x^*}(x')}{\pi_{\lambda, x^*}(x_t)}\right) = \min\left(1, \mathrm{e}^{-\lambda(\mathrm{d}(\mathrm{x}', \mathrm{x}^*) - \mathrm{d}(\mathrm{x}_t, \mathrm{x}^*))}\right)$
  - Acceptation ou rejet : on génére 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

Pour $m > n$, x^* ne peut être une vraie permutation. Nous raisonnons de la même façon en modifiant deux étapes :

- Dans le mécanisme de proposition nous inversons deux coordonnées mais en imposant qu'au moins une des deux soit plus petite que n ;
- Dans le calcul de la probabilité d'acceptation nous ne calculons la distance de Hamming que sur les n premières coordonnées x' et x_t (comme x^* est un vecteur de taille n) ;

$d(x', x^*) - d(x_t, x^*) \in \{-2, -1, 0, 1, 2\}$: si par rapport à x_t dans x' ne diminue pas la distance de Hamming à x^* alors on accepte le nouvel état ; s'il y a pas de nouvel élément mal placé alors on accepte x' avec une probabilité de $e^{-\lambda}$.

Pour $\lambda > 2$ la probabilité d'accepter un nouvel état moins proche que x_t de x^* est donc inférieure à 14 % et pour $\lambda > 3$ cette probabilité inférieure à 5 %. Pour λ grand on converge donc vers x^* et tous les échantillons seront égaux à x^*.

Traitements spécifiques

L'algorithme de Metropolis-Hastings a deux désavantages :

1. Pour générer la loi cible, il construit itérativement une chaîne de Markov qui converge vers cette loi cible. Les échantillons initiaux peuvent donc suivre une distribution très différente de celle recherché. Il est donc nécessaire de rejetter une partie importante des échantillons initiaux (*burn-in*). C'est corrigé en enlevant les $250 \times m$ premières observations.
2. Par construction, les échantillons proches sont corrélés entre eux. Pour générer des échantillons indépendants il faut donc en rejeter et ne garder que les $n$ièmes échantillons. Dans notre cas nous prenons $n = 100$ pour toutes les simulations.

Gestion du burn-in : n = 4 et m = 6

Gestion du burn-in : n = m = 40

Gestion des autocorrélations

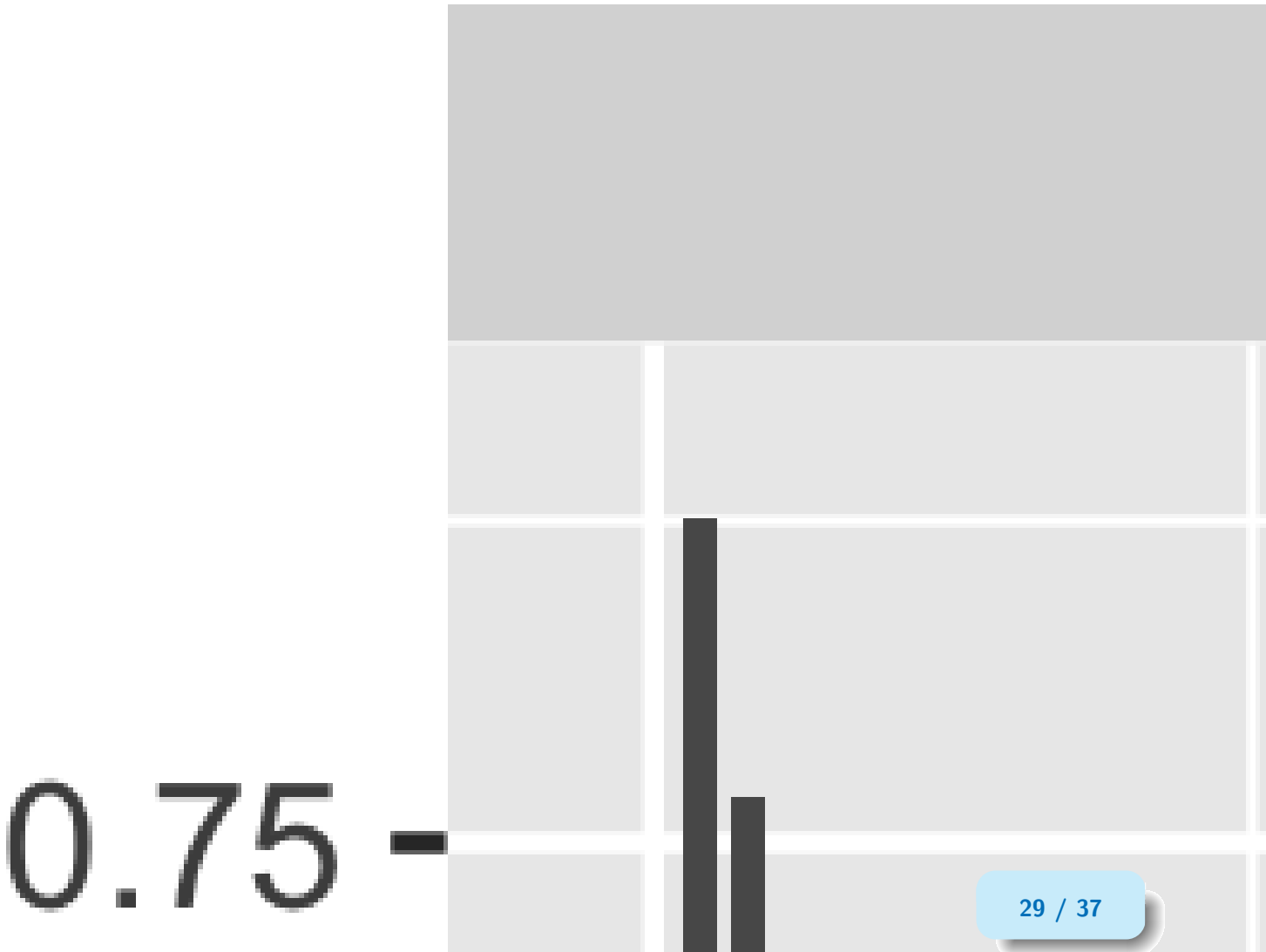
Par construction, les échantillons proches sont corrélés.

On peut utiliser la fonction `portes::LjungBox` pour calculer les autocorrélation : séries multivariées.

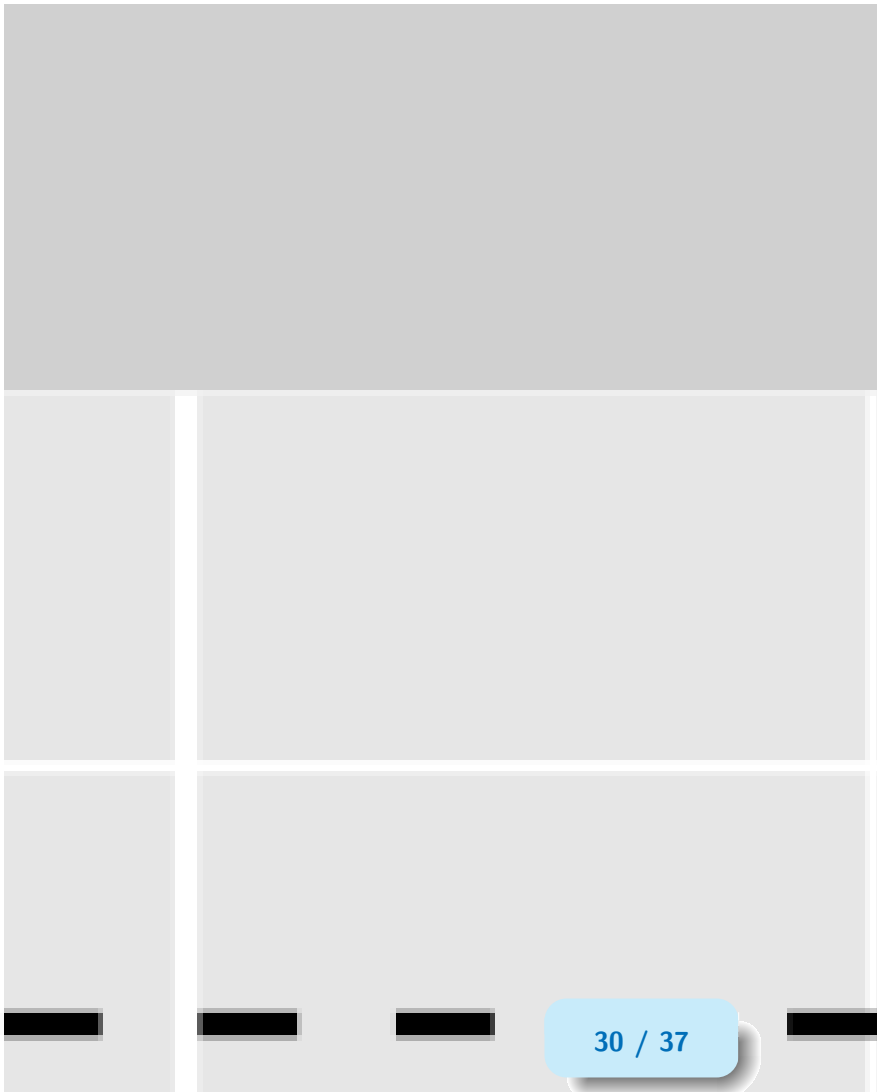
Mais intégrer ce test dans le mécanisme de proposition est très couteux en temps.

Garder tous les 80ièmes observations corrige ce problème.

Gestion des autocorrélations



Gestion des autocorrélations



Estimation des paramètres

La littérature montre le problème de maximisation de la vraisemblance d'une loi de Mallow peut se décomposer en deux étapes :

- Estimation de x^* qui minimise la somme des distances de Hamming.
- Estimation de λ par un algorithme de type Newton-Raphston.

L'estimation des paramètres dans l'algorithme de *Cross-Entropy* est équivalent à calculer l'estimateur de vraisemblance sur les 10 % meilleurs échantillons en terme de score.

Estimation de x^*

Intuitivement, le x^* qui minimise la somme :

$$\sum_{i=1}^{N} 1_{\{S(X_i) \geq \hat{\gamma}_t\}} d(X_i, x^*)$$

est le $x^* = (x_1^*, \ldots, x_n^*)$ tel que x_j^* correspond au chiffre le plus fréquent dans la $j$ème coordonnée des 10 % meilleurs échantillons.

On part de ce principe mais 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

On trouve dans la littérature la constante de normalisation de π_{λ,x^*} :

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

Le maximum de vraisemblance est alors obtenu en prenant λ tel que :

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

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.

Plusieurs test pour estimer λ : croissance linéaire, décroissance, constance et maximum de vraisemblance. Meilleure solution : $\lambda = 1$.

Résultats

Il n'y a pas toujours convergence de x^* vers y en 100 itérations.

TABLE 9 – Médiane du numéro de simulation de convergence

		m						
		4	6	10	15	20	30	40
n	**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							

Note :
S'il n'y a pas convergence les statistiques ne sont pas calculées
Statistiques sur 10 seeds
N = 5 x n x m simulations
Au maximum 100 itérations

Résultats

TABLE 10 – Nombre de simulations n'ayant pas convergé vers la bonne valeur

		m						
		4	6	10	15	20	30	40
n	**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

Note :
Statistiques sur 10 seeds
N = 5 x (n + 1) simulations
Au maximum 100 itérations

Résultats

TABLE 11 – Moyenne du temps de calcul jusqu'à la convergence (en secondes)

		m						
		4	6	10	15	20	30	40
n	**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							

Note :
S'il n'y a pas convergence les statistiques ne sont pas calculées
Statistiques sur 10 seeds
N = 5 x (n + 1) simulations
Au maximum 100 itérations

Merci pour votre attention

L'ensemble du projet est disponible sous :
https://github.com/ARKEnsae/Mastermind_Simulation

IP PARIS