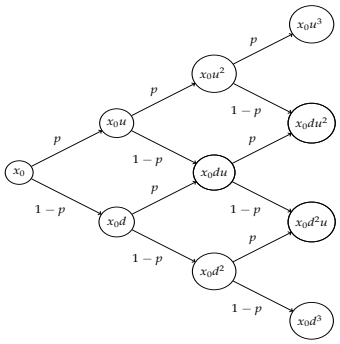
Chaînes de Markov

Application à la décision dans l'incertain

Bernard Lapeyre

http://cermics.enpc.fr/~bl



Plan

Probabilité et espérance conditionnelle

- Une *probabilité* $\mathbb{P}$ sur $(\Omega, \mathcal{A})$.
- Une espérance $\mathbb{E}$ (qui opére *linéairement* sur des variables aléatoires à valeurs dans $\mathbb{R}$).
- *Probabilité conditionnelle* : $A, B \subset E, \mathbb{P}(B) > 0$ par définition

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

- *Espérance conditionnelle* : $B \subset E, \mathbb{P}(B) > 0$, X v.a., par définition

$$\mathbb{E}(X|B) = \frac{\mathbb{E}(X\mathbf{1}_B)}{\mathbb{P}(B)}$$

- $\mathbb{E}(\mathbf{1}_A|B) = \mathbb{P}(A|B)$, linéarité.

Loi d'un variable aléatoire

- Un espace d'état fini $E = \{x_1, \ldots, x_N\}$, une variable aléatoire X à valeur dans E. Loi de X

$$p_X(x) = \mathbb{P}\left(X = x\right), x \in E$$

- $\sum_{x \in E} p_X(x) = 1$.
- A quoi ca sert ? A calculer $\mathbb{E}(f(X))$ pour toute fonction f de E dans $\mathbb{R}$!

$$\mathbb{E}(f(X)) = \sum_{x \in E} p_X(x) f(x).$$

Loi d'un vecteur aléatoire I

- $X = (X_1, \dots, X_n)$ un vecteur aléatoire valeur dans E^n. Loi de X

$$p_X(x_1, \dots, x_n) = \mathbb{P}\left(X_1 = x_1, \dots, X_n = x_n\right), x_1, \dots, x_n \in E$$

- $\sum_{x_1,\dots,x_n \in E} p_X(x_1, \dots, x_n) = 1$.
- A quoi ca sert ? A calculer $\mathbb{E}(f(X_1, \dots, X_n))$ pour toute fonction f de E^n dans $\mathbb{R}$!

$$\mathbb{E}(f(X_1, \dots, X_n)) = \sum_{x_1,\dots,x_n \in E} p_X(x_1, \dots, x_n) f(x_1, \dots, x_n).$$

- Par *contraction,* la loi de X permet de calculer les lois p^i des X_i

$$p_{X_i}(x) = \sum_{x_1,\dots,x_{i-1},x_{i+1},\dots,x_n \in E} p_X(x_1, \dots, x_{i-1}, x, x_{i+1}, \dots, x_n).$$

Loi d'un vecteur aléatoire II

- Une remarque simple (mais utile) :

$$\mathbb{E}(f(X_n)) = \sum_{x_n \in E} p_{X_n}(x_n) f(x_n) = \sum_{x_1,\dots,x_n \in E} p_X(x_1, \dots, x_n) f(x_n).$$

Loi d'un vecteur et indépendance

- $X = (X_1, \ldots, X_n)$ est un vecteur de v.a. indépendantes si et seulement si (au choix), pour tout $x_1, \ldots, x_n \in E$, pour tout $f_1, \ldots, f_n$ de E dans $\mathbb{R}$:
  - $\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \mathbb{P}(X_1 = x_1) \times \ldots \times \mathbb{P}(X_n = x_n)$
  - $\mathbb{E}(f_1(X_1) \times \cdots \times f_n(X_n)) = \mathbb{E}(f_1(X_1)) \times \cdots \times \mathbb{E}(f_n(X_n))$
- les lois marginales p^i des X_i *ne caractérisent pas* la loi du vecteur $(X_1, \ldots, X_n)$!
- ... sauf si on suppose que ces v.a. sont *indépendantes*.

ANDRY MARKOV

- Mathematicien russe 1856 – 1922, élève de Chebyshev, "known for his work in number theory, analysis, and probability theory."
- "He extended the weak law of large numbers and the central limit theorem to certain sequences of dependent random variables forming special classes of what are now known as Markov chains."
- As a lecturer, Markov demanded much of his students : "... The lectures were difficult, and only serious students could understand them. ... During his lectures he did not bother about the order of equations on the blackboard, nor about his personal appearance."

А. А. Марков (1886).

Pour des détail sur la vie et l'œuvre de Markov voir [Basharin et al., 2004].

SYSTÈME DYNAMIQUE ALÉATOIRE

- F application de E dans E définit un système dynamique. x_0 arbitraire, puis
$$x_{n+1} = f(x_n).$$
- $W = (W_n, n \geq 1)$ une suite de variables aléatoires indépendantes de même loi sur un espace G. "On tire F, application de $E \times G$ dans E, au hasard" : $f(x, W_{n+1})$.
- Sytème dynamique aléatoire, x_0 arbitraire, on définit par récurrence
$$X_{n+1} = f(X_n, W_{n+1})$$
- La connaissance de $f(x, w)$ permet de simuler la suite $(X_n, n \geq 0)$ à partir de la suite W.

SYSTÈMES DYNAMIQUES ALÉATOIRES : 3 EXEMPLES

$(W_n, n \geq 1)$ une suite de tirage à pile ou face indépendant, $0 \leq p \leq 1$.

$$\mathbb{P}(W_n = P) = p = 1 - \mathbb{P}(W_n = F)$$

- **Processus en "dents de scie"** : (# pile consécutifs avant n)

$$X_0 = 0, X_{n+1} = (1 + X_n)\mathbf{1}_{\{W_{n+1} = P\}}.$$

- **Marche aléatoire** ($p = 1/2$, marche aléatoire symétrique)

$$X_0 = 0, X_{n+1} = X_n + \left(\mathbf{1}_{\{W_{n+1} = P\}} - \mathbf{1}_{\{W_{n+1} = F\}}\right).$$

- **Processus de Cox-Ross** (cf. finance), $d < 1 < u$

$$X_0 = 1, X_{n+1} = X_n \left(u\, \mathbf{1}_{\{W_{n+1} = P\}} + d\, \mathbf{1}_{\{W_{n+1} = F\}}\right).$$

- autres exemples : gestion de stock, battage de cartes (voir [Berestycki, 2007]), ...

SYSTÈME DYNAMIQUE ALÉATOIRE ET PROPRIÉTÉ DE MARKOV

- Un système dynamique aléatoire vérifie la *propriété de Markov*

$$\mathbb{P}\left(X_{n+1} = x_{n+1} | X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n\right) = P(x_n, x_{n+1})$$

- où P une matrice de transition : $(P(x, y), x, y \in E)$ (parfois aussi noté $(P(x \to y))$ donnée par

$$P(x, y) = \mathbb{P}(f(x, W_1) = y).$$

- *Matrice de transition* : $P(x, y) \geq 0$ et $\sum_{y \in E} P(x, y) = 1$.
- On a forcement $P(x, y) = P\left(X_{n+1} = y | X_n = x\right)$ et

$$P\left(X_{n+1} = x_{n+1} | X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n\right) = P\left(X_{n+1} = x_{n+1} | X_n = x_n\right)$$

PREUVE

$$\begin{aligned}&\mathbb{P}\left(X_{n+1}=x_{n+1},X_n=x_n,\dots,X_0=x_0\right)\\&\quad=\mathbb{P}\left(f(X_n,W_{n+1})=x_{n+1},X_n=x_n,\dots,X_0=x_0\right)\\&\quad=\mathbb{P}\left(f(x_n,W_{n+1})=x_{n+1},X_n=x_n,\dots,X_0=x_0\right).\end{aligned}$$

- $f(x_n, W_{n+1})$ indépendante du vecteur $(W_1, \dots, W_n)$
- $X_0, \dots, X_n$ fonctions de $(W_1, \dots, W_n)$.
- Donc $f(x_n, W_{n+1})$ indépendante de $X_0, \dots, X_n$.

$$\begin{aligned}&\mathbb{P}\left(X_{n+1}=x_{n+1},X_n=x_n,\dots,X_0=x_0\right)\\&\quad=\mathbb{P}\left(f(x_n,W_{n+1})=x_{n+1}\right)\mathbb{P}\left(X_n=x_n,\dots,X_0=x_0\right)\\&\quad=P(x_n,x_{n+1})\mathbb{P}\left(X_n=x_n,\dots,X_0=x_0\right)\end{aligned}$$

- En sommant sur tous les $x_0, \dots, x_{n-1}$ dans E, on obtient aussi

$$\mathbb{P}\left(X_{n+1}=x_{n+1},X_n=x_n\right)=P(x_n,x_{n+1})\mathbb{P}\left(X_n=x_n\right).$$

MATRICE DE TRANSITION : EXEMPLES

- Exemple 1 : $x \in \mathbb{N}$, $P(x, x+1) = p$, $P(x, 0) = 1-p$, 0 sinon
- Exemple 2 : $x \in \mathbf{Z}$, $P(x, x+1) = p$, $P(x, x-1) = 1-p$, 0 sinon
- Exemple 3 : $P(x, xu) = p$, $P(x, xd) = 1-p$, 0 sinon

Système dynamique aléatoire et chaîne de Markov

- une suite de variable aléatoire qui vérifie la propriété de Markov, s'appelle une *chaîne de Markov*.
- $P(x,y) = \mathbb{P}(X_{n+1} = y | X_n = x)$, $x, y \in E$, est la *matrice de transition* de la chaîne de Markov.
- Un système dynamique aléatoire est une chaîne de Markov et reciproquement on peut représenter (en loi) une chaîne de Markov comme un système dynamique aléatoire.
- Bien que la matrice de transition $P(x,y)$ se déduise de F et de la loi de W, elle est plus utile que F. Tous les calculs de loi utilisent uniquement P.

Chaîne de Markov : définition

Definition 1

Une suite de v.a. $(X_n, n \geq 0)$ à valeurs dans E est un chaîne de Markov de *matrice de transition* $(P(x,y), x \in E, y \in E)$, si et seulement si, par définition, pour tout $x_0, x_1, \ldots, x_n, x_{n+1} \in E$,

$$\begin{aligned}\mathbb{P}(X_{n+1} = x_{n+1} | X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n) \\ = P(x_n, x_{n+1}) \\ = \mathbb{P}(X_{n+1} = x_{n+1} | X_n = x_n)\end{aligned}$$

La loi de X_0, μ_0, est appelée la *loi initiale*.

- Formellement, égalité vrai seulement si $\mathbb{P}(X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n) > 0$, sinon l'interpréter comme

$$\begin{aligned}\mathbb{P}(X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n, X_{n+1} = x_{n+1}) \\ = \mathbb{P}(X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n) P(x_n, x_{n+1}).\end{aligned}$$

- Matrice de transition P_n dépendant de n : chaîne de Markov *inhomogène*.

MATRICE DE TRANSITION ET CALCUL DE LOIS

- E fini
- $(X_n, n \geq 0)$ une chaîne de Markov de matrice de transition P sur E.
- μ_0 la loi de X_0 (loi initiale), $\mu_0(x) = \mathbb{P}(X_0 = x)$, $x \in E$.
- La loi du vecteur $(X_0, \ldots, X_n)$ se calcule explicitement en fonction de P et μ_0

$$\mathbb{P}(X_0 = x_0, \ldots, X_n = x_n) = \mu_0(x_0) P(x_0, x_1) \times \cdots \times P(x_{n-1}, x_n).$$

- Preuve = propriété de Markov

$$\begin{aligned}\mathbb{P}(X_0 = x_0, \ldots, X_{n-1} = x_{n-1}, X_n = x_n) \\ = \mathbb{P}(X_0 = x_0, \ldots, X_{n-1} = x_{n-1}) P(x_{n-1}, x_n),\end{aligned}$$

puis on itère.

MATRICE DE TRANSITION : QUELQUES NOTATIONS

- E fini : μ est un vecteur *ligne*, P une *matrice* et f un vecteur *colonne*.
- $(Pf)(x) = \sum_{y\in E} P(x,y)f(y)$, f une fonction de E dans $\mathbb{R}$. Pf est aussi une fonction de E dans $\mathbb{R}$.
- $(\mu P)(y) = \sum_{x\in E} \mu(x)P(x,y)$, μ une loi de probabilité sur E. μP est aussi une probabilité sur E
- $(PQ)(x,y) = \sum_{z\in E} P(x,z)Q(z,y)$, pour P et Q deux matrices de transition. PQ est aussi une matrice de transition.
- P^n est définie par récurrence par :
$$P^{n+1}(x,y) = \sum_{z\in E} P(x,z)P^n(z,y).$$
- μ loi sur E, f fonction de E dans $\mathbb{R}$: $\mu f = \sum_{x\in E} \mu(x)f(x)$.
- Si X a pour loi μ, $\mathbb{E}(f(X)) = \mu f$.

LOI DE X_n

- Par contraction de la loi du vecteur $(X_0, \ldots, X_n)$, on obtient
$$\mathbb{P}(X_n = x_n) = \sum_{x_0,x_1,\ldots,x_{n-1}\in E} \mu_0(x_0)P(x_0,x_1) \times \cdots \times P(x_{n-1},x_n).$$
- avec les notations précédentes :
$$\mathbb{P}(X_n = x_n) = (\mu_0 P^n)(x_n),$$
- $\text{loi}(X_n) = \mu_0 P^n$: un (simple) calcul matriciel à partir de μ_0 et P.

UN TEST D'ÉQUIRÉPARTITION

- Ces 100 tirages à pile ou face (réalisé par mes soins !) sont ils vraiment "aléatoire" ?

P	F	F	F	P	F	P	P	F	F
F	P	P	P	F	F	P	F	P	F
P	F	F	F	P	P	F	F	P	P
P	F	F	P	F	P	P	F	F	P
F	P	P	F	P	P	F	F	F	P
P	P	F	P	F	P	P	F	P	P
F	F	P	F	P	P	F	F	P	F
P	F	F	F	P	F	P	P	F	F
P	F	F	F	P	P	P	F	P	F
P	F	F	P	P	F	P	F	P	P

- Ça n'a pas l'air trop mal : 50P, 50F ...

UN TEST D'ÉQUIRÉPARTITION

- mais ... le nombre maximum de piles consécutifs est de 3, ce qui est (très) peu compatible avec l'hypothèse d'un tirage au hasard.
- On a du mal a imiter le hasard, ...
- Nous allons voir qu'il faut s'attendre à 4 piles consécutifs (avec proba 0.97) voire 5 (avec proba 0.81).
- On va calculer le loi du nombre maximum de P consécutifs au bout de 100 tirages.
- Et en déduire une procédure de type test statistique.

Chaîne de Markov arrêtée

- $(X_n, n \geq 0)$ un système dynamique aléatoire

$$X_{n+1} = f(X_n, W_{n+1}).$$

- $x \in E$, $\tau = \inf\{n \geq 0, X_n = x_0\}$: le premier temps d'atteinte de x_0 (pas forcément fini, $+\infty$ si ensemble vide).
- $n \wedge \tau = \inf\{n, \tau\}$
- $Y_n = X_{n\wedge\tau}$ est aussi un système dynamique aléatoire (et donc une chaîne de Markov) puisque

$$Y_{n+1} = f(Y_n, W_{n+1})\mathbf{1}_{\{Y_n \neq x_0\}} + x_0\mathbf{1}_{\{Y_n = x_0\}} = G(Y_n, W_{n+1}).$$

$$G(x,u) = f(x,u)\mathbf{1}_{\{x \neq x_0\}} + x_0\mathbf{1}_{\{x = x_0\}}$$

Chaîne en dents de scie arrêtée

- X la chaîne en dents de scie, $l > 0$, $\tau_l = \inf\{n \geq 0, X_n = l\}$, $Y^l_n = X_{n\wedge\tau_l}$.
- Y^l_n est un chaîne de Markov à valeurs dans $\{0, \ldots, l\}$ de matrice de transition P donnée par (les autres élèments sont nuls) :

$$\begin{cases} P_l(x, x+1) = p & \text{si } x < l, \\ P_l(x, 0) = 1-p & \text{si } x < l, \\ P_l(l,l) = 1. \end{cases}$$

$$P_l = \begin{pmatrix} 1-p & p & 0 & \ldots & 0 & 0 \\ 1-p & 0 & p & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1-p & 0 & 0 & \ldots & p & 0 \\ 1-p & 0 & 0 & \ldots & 0 & p \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}.$$

Loi du nombre de piles consécutifs après 100 tirages

- $N^n_{\max}$: nombre maximum de piles consécutifs après n tirages.
- $\{Y^l_n = l\} = \{N^n_{\max} \geq l\}$.
- Loi initiale $\mu_0(0) = 1$, $\mu_0(k) = 0$ sinon. $\mu_0 = (1, 0, \ldots, 0)$.
- Loi de $Y^l_n = \mu_0(P_l)^n$.
- $\mathbb{P}(Y^l_n = l) = (P_l)^n(0, l)$.
- Il est facile d'écrire un programme `Scilab` qui fait cela (on le fera demain).
- On peut calculer la *loi* du nombre de piles consécutifs maximum après 100 tirages $N_{\max}$.

$$\begin{aligned}\mathbb{P}(N^{100}_{\max} = l) &= \mathbb{P}(Y^l_{100} = l) - \mathbb{P}(Y^{l+1}_{100} = l+1) \\ &= P^{100}_l(0, l) - P^{100}_{l+1}(0, l+1).\end{aligned}$$

Loi du nombre maximum de piles consécutifs

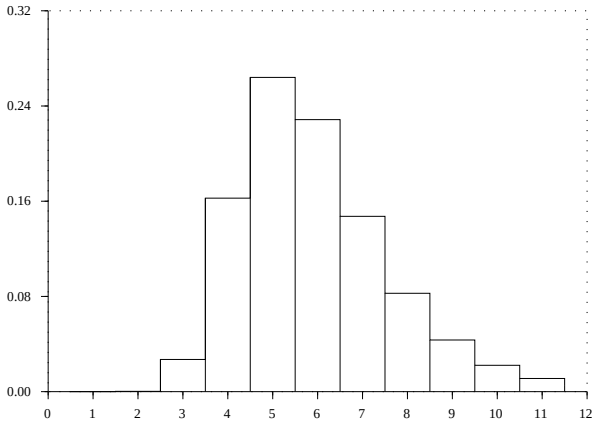


Figure – Loi du nombre de piles consécutifs.

- $\mathbb{P}(\text{nombre de piles consécutifs } \leq 3) = 0.0273$
- $\mathbb{P}(\text{nombre de piles consécutifs } \geq 5) = 0.8101$
- $\mathbb{P}(\text{nombre } \geq 10) > \mathbb{P}(\text{nombre } \leq 3)$!!

UNE RÉGLE DE DÉCISION

- Si l'on voit au moins 4 piles consécutifs, la suite sera déclarée aléatoire, sinon, la suite sera déclarée non aléatoire.
- Si la suite a été tirée aléatoirement, je vais me tromper avec moins de 3% d'erreur.
- Si la suite n'est pas aléatoire, je suis aidé par la tendance naturelle à ne pas faire de trop longues series.
- "Mon" exemple ne passe pas le test. Mais si on le lit par colonne ou lieu de ligne, il le passe aisément !
- À vérifier avec l'un de vos collègues ...

PROCESSUS DE MARKOV

- *Processus de Markov* et *propriété de Markov*

$$P\left(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_0 = x_0\right) = \mathbb{P}\left(X_{n+1} = x_{n+1} | X_n = x_n\right).$$

- la matrice de transition : $(P(x,y), x, y \in E)$

$$P(x,y) = \mathbb{P}\left(X_{n+1} = y | X_n = x\right).$$

- Loi de $(X_0, \ldots, X_N)$ déterminée par P et la loi de X_0 (notée μ_0).

$$\mathbb{P}\left(X_0 = x_0, \ldots, X_n = x_n\right) = \mu_0(x_0) P(x_0, x_1) \times \cdots \times P(x_{n-1}, x_n).$$

- Loi de X_N, μ_N donnée par $\mu_N = \mu_0 P^N$.

Loi de X_n : une autre façon d'écrire les choses

- $(X_n, n \geq 0)$ chaîne de Markov de matrice de transition P sur E.
- $\mathbb{P}(X_0 = x_0) = 1$ (i.e. $\mu_0(x_0) = 1$, $\mu_0(x) = 0$ si $x \neq x_0$.)
- On sait exprimer $\mathbb{E}(f(X_N))$

$$\mathbb{E}(f(X_N)) = \mu_N f = \mu_0 P^N f = \sum_{x \in E} \mu_0(x)(P^N f)(x) = (P^N f)(x_0).$$

Une formulation alternative plus algorithmique.

Theorem 2

Soit $(u(n,x), n = 0, \ldots, N, x \in E)$ la solution unique de

$$\begin{cases} u(n,x) = \sum_{y \in E} P(x,y) u(n+1,y), & n < N \\ u(N,x) = f(x), \end{cases} \tag{1}$$

Alors $\mathbb{E}(f(X_N)) = u(0, x_0)$.

Remarques

- La première équation de (1) peut aussi s'écrire

$$u(n,x) = P[u(n+1,.)](x) = \sum_{y \in E} P(x,y) u(n+1,y)$$

- $u(n,x)$ peut s'interpréter comme

$$u(n,x) = \overbrace{P \times \cdots \times P}^{N-n \text{ fois}} f(x) = P^{N-n} f(x),$$

- Lorsque $\mathbb{P}(X_n = x) > 0$, on a

$$u(n,x) = \mathbb{E}\left(f(X_N) | X_n = x\right) = P^{N-n} f(x).$$

PREUVE

$$\begin{aligned}&\mathbb{P}(X_0 = x_0, \dots, X_n = x_n, X_{n+1} = x_{n+1}, \dots, X_N = x_N)\\ &\quad = \mathbb{P}(X_0 = x_0)P(x_0, x_1) \times \cdots \times P(x_n, x_{n+1}) \times P(x_{n+1}, x_{n+2}) \times \cdots \times P(x_{N-1}, x_N)\end{aligned}$$

en sommant sur toutes les valeurs de $x_0, \cdots, x_{n-1}$ et $x_{n+1}, \cdots, x_{N-1}$ on obtient la loi de (X_n, X_N)

$$\mathbb{P}(X_n = x_n, X_N = x_N) = \mathbb{P}(X_n = x_n)P^{N-n}(x_n, x_N)$$

$$\begin{aligned}\mathbb{E}\left(f(X_N)\mathbf{1}_{\{X_n = x_n\}}\right) &= \mathbb{P}(X_n = x_n)\sum_{x_N \in E} P^{N-n}(x_n, x_N)f(x_N)\\ &= \mathbb{P}(X_n = x_n)(P^{N-n}f)(x_n)\end{aligned}$$

REMARQUES ET NOTATIONS

- (1) est *une équation de programmation dynamique*
- E est fini, (1) est un *algorithme* qui termine.

UNE NOTATION COMMODE "À LA `Scilab`"

- $x_{0:n}$ plutôt que $(x_0, \dots, x_n)$
- $X_{0:n}$ plutôt que $(X_0, \dots, X_n)$
- $X_{0:n} = x_{0:n}$ plutôt que $(X_0 = x_0, \dots, X_n = x_n)$

Preuve formelle

On le sait déjà ... mais voici une autre méthode de preuve. On va montrer que

$$\mathbb{E}\left(u(n+1, X_{n+1})\right) = \mathbb{E}\left(u(n, X_n)\right).$$

Si cela est vrai :

$$u(0, x_0) = \mathbb{E}\left(u(0, X_0)\right) = \cdots = \mathbb{E}\left(u(N, X_N)\right) = \mathbb{E}\left(f(X_N)\right).$$

La loi de $X_{0:n+1} = (X_{0:n}, X_{n+1})$ est donnée par (c'est un façon d'exprimer la propriété de Markov)

$$\mathbb{P}\left(X_{0:n+1} = x_{0:n+1}\right) = \mathbb{P}\left(X_{0:n} = x_{0:n}, X_{n+1} = x_{n+1}\right) = \mathbb{P}\left(X_{0:n} = x_{0:n}\right) P(x_n, x_{n+1}).$$

$$\begin{aligned}
\mathbb{E}\left(u(n+1, X_{n+1})\right) &= \sum_{x_{0:n+1} \in E^{n+2}} u(n+1, x_{n+1}) \mathbb{P}\left(X_{0:n+1} = x_{0:n+1}\right), \\
&= \sum_{x_{0:n} \in E^{n+1}, x_{n+1} \in E} u(n+1, x_{n+1}) \mathbb{P}\left(X_{0:n} = x_{0:n}\right) P(x_n, x_{n+1}), \\
&= \sum_{x_{0:n} \in E^{n+1}} \mathbb{P}\left(X_{0:n} = x_{0:n}\right) \sum_{x_{n+1} \in E} P(x_n, x_{n+1}) u(n+1, x_{n+1}), \\
&= \sum_{x_{0:n} \in E^{n+1}} \mathbb{P}\left(X_{0:n} = x_{0:n}\right) u(n, x_n) = \mathbb{E}\left(u(n, X_n)\right).
\end{aligned}$$

Calcul du prix d'une option

- Un modèle : le processus de Cox-Ross

$$X_0 = 1, X_{n+1} = X_n \left(u \, \mathbf{1}_{\{W_{n+1} = P\}} + d \, \mathbf{1}_{\{W_{n+1} = F\}}\right).$$

- Un profil d'option $f(X_N)$ (ce que je vais recevoir en N)

$$f(x) = (x - K)_+, \text{ option d'achat de prix d'exercice } K.$$

- Prix : $\mathbb{E}(f(X_N))$ ("intuitif", voir [Lamberton and Lapeyre, 1997]).
- Calcul du prix : $(u(n, x), n = 0, \ldots, N, x \in E)$

$$\begin{cases} u(n, x) = pu(n+1, xu) + (1-p)u(n+1, xd), \\ u(N, x) = f(x). \end{cases} \tag{2}$$

UN PROGRAMME

Un programme en `Scilab`, catastrophique (complexité 2^N), mais qui a le bon goût de terminer (lentement)

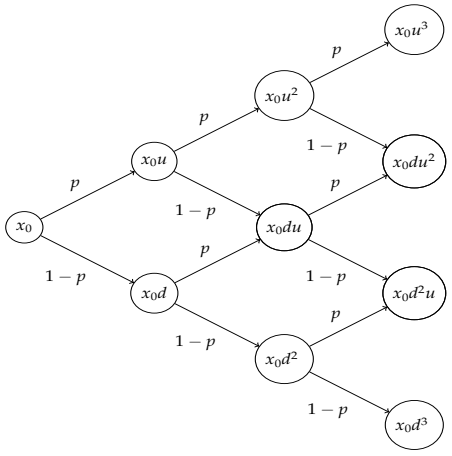
```
N=10;p=1/2;
d=1/2;u=2;
K=1;

function res=f(x)
   res=max(x-K,0)
endfunction

function res=u(n,x)
   if n==N then res=f(x); return; end;
   res = p * u(n+1,x*u)+ (1-p) * u(n+1,x*d);
endfunction

function res=prix(x)
  res=u(0,x);
endfunction
```


POUR ALLER PLUS VITE, UN DESSIN POUR $N = 3$



UN ALGORITHME QUI FAIT LA MÊME CHOSE, PLUS VITE ...

- Si $X_0 = x_0$ les seules valeurs que peut prendre X_n sont données par (k = nombre de fois où l'on tire u entre 0 et $n-1$)

$$(x_k^{(n)} = x_0 u^k d^{n-k}, 0 \leq k \leq n).$$

- Il suffit de calculer les vecteurs $U^{(n)} = (u(n, x_k^{(n)}), 0 \leq k \leq n)$, qui vérifient

$$\begin{cases} U^{(N)}(k) = f(x_k^{(N)}), \quad k = 0, \dots, N \\ U^{(n)}(k) = pU^{(n+1)}(k+1) + (1-p)U^{(n+1)}(k), \quad k = 0, \dots, n. \end{cases} \tag{3}$$

- Ce qui donne un agorithme (de complexité N^2).

UN ALGORITHME QUI FAIT LA MÊME CHOSE, PLUS VITE ...

```
function res=f(x); res=max(x-K,0);endfunction
function res=inc(n); res=n+1; endfunction;

function res=prix(x_0)
  U=zeros(inc(N),inc(N));
  for k=[0:N] do
    U(inc(N),inc(k)) = f(x_0 * u^k * d^(N-k));
  end;

  for n=[N-1:-1:0] do
    for k=[0:n] do
      U(inc(n),inc(k)) = p*U(inc(n+1),inc(k+1))+(1-p)*U(inc(n+1),inc(k));
    end;
  end;
  res=U(inc(0),inc(0));
endfunction;
```

BIBLIOGRAPHIE

Basharin, G. P., Langville, A. N., and Naumov, V. A. (2004).
The life and work of A. A. Markov.
Linear Algebra Appl., 386 :3–26.

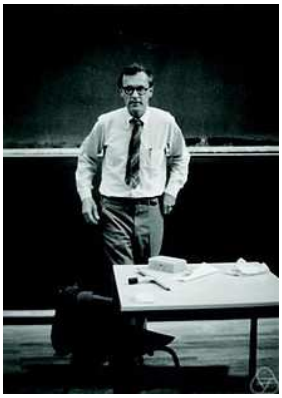
Berestycki, N. (2007).
Notes on card shuffling.
Technical report,
http://www.statslab.cam.ac.uk/~beresty/Articles/shuffle.pdf.

Graham, C. (2008).
Chaînes de Markov.
Dunod.

Lamberton, D. and Lapeyre, B. (1997).
Introduction au calcul stochastique appliqué à la finance.
Ellipses, Édition Marketing, Paris, second edition.

SNELL, JAMES LAURIE

- La théorie de l'arrêt optimal porte le nom d'enveloppe de Snell (voir [Snell, 1952]), "so named by the Russian mathematician, Kolmogorov".
- Snell (1925-2011), mathématicien américain, élève de Doob (l'un des pioniers de la théorie des martingales).
- Il utilise la théorie des martingales pour traiter le problème d'arrêt optimal.



LA QUESTION DU JOUR

- $(X_n, n \geq 0)$ une chaîne de Markov de matrice de transition P sur E. $\mathbb{P}(X_0 = x_0) = 1$.
- On cherche à calculer non pas $\mathbb{E}(f(X_N))$, mais $\sup_{\tau \leq N} \mathbb{E}(f(X_\tau))$.
- τ appartenant à une famille de temps aléatoires, *plus grand* que les temps déterministes, mais *plus petit* que tous les temps aléatoires.
- *plus grand* que les temps déterministes : parce que l'on souhaite pouvoir tenir compte des valeurs de X_n au fil du temps.
- *plus petit* que tous les temps aléatoires : parce que à l'instant n on ne connait pas la trajectoire future $X_{n+1}, \ldots, X_N$.
- $\tau = \text{Argmax}\{f(X_n), 0 \leq n \leq N\}$ est optimum, mais réclame de connaitre le futur !
- Voir Pour la Sciences [Hill, 2009] pour une introduction à ce genre de problème.

TEMPS D'ARRÊT

La notion adéquate est celle de *temps d'arrêt* : on souhaite prendre la décision d'arrêter avec l'information que l'on a au temps n.

Definition 3

τ est un *temps d'arrêt*, si, pour tout n, il existe $A_n \subset E^{n+1}$ tel que

$$\{\tau = n\} = \{(X_0, X_1, \ldots, X_n) \in A_n\}$$

- "Je peux déterminer si $\tau = n$ en ne considérant que la portion de trajectoire avant n".
- Argmax$\{f(X_n), 0 \leq n \leq N\}$ ne peut pas être un temps d'arrêt (sauf cas particulier).
- Le temps d'atteinte d'un point z de E est un temps d'arrêt

$$\tau = \inf\{n \geq 0, X_n = z\}.$$

En effet $\{\tau = n\} = \{X_0 \neq z, X_1 \neq z, \ldots, X_{n-1} \neq z, X_n = z\}$.

FORMULATION D'UN PROBLÈME D'ARRÊT OPTIMAL

- On se donne une chaîne de Markov $(X_n, n \geq 0)$ sur E, de matrice de transition P, issue de x_0 en 0.
- $f(n, x)$ donné, on cherche à calculer le sup suivant

$$\sup_{\tau \text{ t.a.} \leq N} \mathbb{E}(f(\tau, X_\tau))$$

- et aussi à identifier un τ qui réalise ce sup.
- ... on sait répondre complétement à ces deux questions.

Les données : P, x_0, f.

SOLUTION DU PROBLÈME D'ARRÊT OPTIMAL

Theorem 4

Si $(u(n,x), n = 0, \dots, N, x \in E)$ *est la solution unique de*

$$\begin{cases} u(n,x) = \max\left\{ \sum_{y\in E} P(x,y)u(n+1,y), f(n,x) \right\}, n < N, x \in E \\ u(N,x) = f(N,x), x \in E. \end{cases} \quad (4)$$

Alors

- $\sup_{\tau \le N} \mathbb{E}(f(\tau, X_\tau)) = u(0, x_0)$
- $\tau_0 = \inf\{n \ge 0, u(n, X_n) = f(n, X_n)\}$ *est un temps d'arrêt optimal.*

- Lorsque la matrice de transition dépend de n il faut remplacer P par P_n (la matrice de transition entre les instants n et $n+1$) dans l'équation.
- τ_0 est bien un temps d'arrêt $\le N$.

$$\{\tau_0 = n\} = \{u(0, X_0) \neq f(0, X_0), \dots, u(n-1, X_{n-1}) \neq f(n-1, X_{n-1}), u(n, X_n) = f(n, X_n)\}$$

- Preuve formelle du théorème : transparent 51.

COMMENTAIRES

- (4) est une équation de *programmation dynamique* proche de celle qui permet de calculer $\mathbb{E}(f(X_N))$.
- $u(n,x) = \sup_{n \le \tau \le N} \mathbb{E}(f(\tau, X_\tau) | X_n = x)$.
- Preuve informelle ("Principe d'optimalité") : *En n, si $X_n = x$ soit j'exerce en n et je gagne $f(n,x)$ soit j'attends $n+1$ où je peux gagner $u(n+1, X_{n+1})$, dont je dois calculer l'espérance en n sachant que $X_n = x$ qui est donnée par*

$$\sum_{y\in E} P(x,y)u(n+1,y).$$

- On l'utilise (4) pour écrire un algorithme (pas beaucoup plus compliqué (il suffit de rajouter le max) que celui qui permet de calculer $\mathbb{E}(f(X_N))$, on le fera demain).

Exemple 1 : Calcul du prix d'options américaines

- $(X_n, 0 \leq n \leq N)$ une chaîne de Markov de matrice de transition P décrivant l'évolution des prix des actifs (e.g. modèle de Cox-Ross).
- J'ai la possibilité si j'"exerce" en $n \leq N$ de gagner $f(n, X_n)$.
- Que vaut ce droit et à quel moment dois-je exercer ce droit pour maximiser mon gain ?
- Pour calculer le prix (la valeur de ce droit), il est naturel [1] de chercher à maximiser l'espérance du flux actualisé $\mathbb{E}(f(\tau, X_\tau))$ parmi tous les temps d'arrêt de X.

1. pour une justification compléte voir [Lamberton and Lapeyre, 1997].

Problème classique : put américain

- On cherche à calculer $\sup_{\tau \leq N} \mathbb{E}\left(f(\tau, X_\tau)\right)$, le prix de l'option et à determiner le moment d'exercice optimum.
- $(X_n, 0 \leq n \leq N)$ est le processus de Cox-Ross

$$X_0 = 1, X_{n+1} = X_n \left(u \mathbf{1}_{\{U_{n+1} = P\}} + d \mathbf{1}_{\{U_{n+1} = F\}} \right).$$

$(U_n, n \geq 1)$ une suite de tirage à pile ou face indépendant,

$$0 \leq p \leq 1, \quad \mathbb{P}(U_n = P) = p = 1 - \mathbb{P}(U_n = F).$$

- r le taux d'intérêt sur une période, K le strike, $d < 1 + r < u$.

$$f(n, x) = \frac{1}{(1+r)^n} (K - x)_+$$

SOLUTION

- On commence par calculer (on le fera demain) $(u(n,x), n = 0, \dots, N, x \in E)$ la solution de l'équation (4) du théorème 4, ici

$$\begin{cases} u(n,x) = \max\left(p\, u(n+1,xu) + (1-p)\, u(n+1,xd), \dfrac{(K-x)_+}{(1+r)^n}\right), \\ u(N,x) = \dfrac{(K-x)_+}{(1+r)^N}. \end{cases} \tag{5}$$

- $u(0,x) = \sup_{\tau \le N} \mathbb{E}\left(\frac{(K-X_\tau)_+}{(1+r)^\tau}\right)$.
- un temps d'arrêt optimal

$$\tau_0 = \inf\left\{n \ge 0, u(n, X_n) = (K - X_n)_+/(1+r)^n\right\}.$$

UNE VARIANTE UTILE

- Souvent on calcule $v(n,x) = (1+r)^n u(n,x)$ plutot que $u(n,x)$. On obtient l'algorithme de Cox-Ross pour les options put américain.
- v est solution de

$$\begin{cases} v(n,x) = \max\left(\dfrac{p\, v(n+1,xu) + (1-p)\, v(n+1,xd)}{1+r}, (K-x)_+\right), \\ v(N,x) = (K-x)_+. \end{cases} \tag{6}$$

- $v(0,x)(= u(0,x)) = \sup_{\tau \le N} \mathbb{E}\left(\frac{(K-X_\tau)_+}{(1+r)^\tau}\right)$.
- un temps d'arrêt optimal $\tau_0 = \inf\left\{n \ge 0, v(n, X_n) = (K - X_n)_+\right\}$.

Preuve du théorème 4 : τ temps d'arrêt quelconque

- u solution (4), on va voir que, pour tout τ t.a., $\mathbb{E}\left(u\left(n \wedge \tau, X_{n\wedge\tau}\right)\right)$ **décroît en** n.
- En admettant ceci, on obtient

$$\begin{aligned} u(0,x_0) &= \mathbb{E}\left(u(0,X_0)\right) = \mathbb{E}\left(u\left(0\wedge\tau, X_{0\wedge\tau}\right)\right) \\ &\geq \mathbb{E}\left(u\left(N\wedge\tau, X_{N\wedge\tau}\right)\right) = \mathbb{E}\left(u\left(\tau, X_\tau\right)\right) \geq \mathbb{E}\left(f\left(\tau, X_\tau\right)\right). \end{aligned}$$

  Ce qui permet d'obtenir $\sup_{0\leq\tau\leq N, \text{t.a.}} \mathbb{E}\left(f\left(\tau, X_\tau\right)\right) \leq u(0,x_0)$.

- Pour montrer la décroissance annoncée, on remarque que

$$\begin{aligned} \Delta_{n+1} &= \mathbb{E}\left(u\left((n+1)\wedge\tau, X_{(n+1)\wedge\tau}\right)\right) - \mathbb{E}\left(u(n\wedge\tau, X_{n\wedge\tau})\right) \\ &= \mathbb{E}\left[\left(u(n+1, X_{n+1}) - u(n, X_n)\right)\mathbf{1}_{\{\tau \geq n+1\}}\right]. \end{aligned}$$

Preuve du théorème 4 : τ temps d'arrêt quelconque

- τ est un temps d'arrêt, $\{\tau \geq n+1\} = \{\tau \leq n\}^c$, s'écrit sous la forme

$$\{\tau \geq n+1\} = \{X_{0:n} \in \bar{A}_n\}.$$

- La loi de $(X_{0:n}, X_{n+1})$ est donnée par (c'est la propriété de Markov)

$$\mathbb{P}\left(X_{0:n} = x_{0:n}, X_{n+1} = x_{n+1}\right) = \mathbb{P}\left(X_{0:n} = x_{0:n}\right) P(x_n, x_{n+1}).$$

$$\begin{aligned} \Delta_{n+1} &= \mathbb{E}\left[\left(u(n+1, X_{n+1}) - u(n, X_n)\right)\mathbf{1}_{\left\{X_{0:n} \in \bar{A}_n\right\}}\right], \\ &= \sum_{x_{0:n}\in\bar{A}_n, x_{n+1}\in E} \left(u(n+1, x_{n+1}) - u(n, x_n)\right)\mathbb{P}\left(X_{0:n} = x_{0:n}\right) P(x_n, x_{n+1}), \\ &= \sum_{x_{0:n}\in\bar{A}_n} \mathbb{P}\left(X_{0:n} = x_{0:n}\right)\left(\sum_{x_{n+1}\in E} u(n+1, x_{n+1})P(x_n, x_{n+1}) - u(n, x_n)\right). \end{aligned}$$

u sol. de (4), $\sum_{x_{n+1}\in E} u(n+1, x_{n+1})P(x_n, x_{n+1}) \leq u(n, x_n)$, d'où $\Delta_{n+1} \leq 0$.

Preuve du théorème 4 : τ_0 temps d'arrêt optimal

- $\tau_0 = \inf\{n \geq 0, u(n, X_n) = f(n, X_n)\}$.
- On va montrer que $\mathbb{E}\left(u\left(n \wedge \tau_0, X_{n\wedge\tau_0}\right)\right)$ **est constant en** n (i.e. $\Delta_{n+1} = 0$).
- En admettant ceci, il est facile de conclure

$$\begin{aligned} u(0, x_0) &= \mathbb{E}\left(u(0, X_0)\right) = \mathbb{E}\left(u\left(0 \wedge \tau_0, X_{0\wedge\tau_0}\right)\right), \\ &= \mathbb{E}\left(u\left(N \wedge \tau_0, X_{N\wedge\tau_0}\right)\right) = \mathbb{E}\left(u\left(\tau_0, X_{\tau_0}\right)\right), \end{aligned}$$

  Mais, par définition de τ_0, $u\left(\tau_0, X_{\tau_0}\right) = f\left(\tau_0, X_{\tau_0}\right)$, donc

$$u(0, x_0) = \mathbb{E}\left(f\left(\tau_0, X_{\tau_0}\right)\right).$$

- Ce qui finit la démonstration puisque τ_0 réalise alors le sup.
- Il nous reste à montrer que, pour ce temps d'arrêt $\Delta_{n+1} = 0$.

Preuve du théorème 4 : τ_0 temps d'arrêt optimal

- τ_0 est un temps d'arrêt, $\{\tau_0 \geq n+1\} = \{X_{0:n} \in \bar{A}_n^0\}$ où

$$\bar{A}_n^0 = \{u(0, x_0) \neq f(0, x_0), \ldots, u(n, x_n) \neq f(n, x_n)\}.$$

- Sur l'événement $\{\tau_0 \geq n+1\}$, on a $u(n, X_n) \neq f(n, X_n)$ et comme u sol. de (4)

$$\sum_{x_{n+1}\in E} u(n+1, x_{n+1}) P(X_n, x_{n+1}) = u(n, X_n).$$

- On termine alors comme tout à l'heure, mais en tenant compte de cette égalité :

$$\begin{aligned} \Delta_{n+1} &= \mathbb{E}\left[\left(u(n+1, X_{n+1}) - u(n, X_n)\right) \mathbf{1}_{\left\{X_{0:n} \in \bar{A}^0{}_n\right\}}\right], \\ &= \sum_{x_{0:n}\in\bar{A}_n^0, x_{n+1}\in E} \left(u(n+1, x_{n+1}) - u(n, x_n)\right) \mathbb{P}\left(X_{0:n} = x_{0:n}\right) P(x_n, x_{n+1}), \\ &= \mathbb{E}\left(\mathbf{1}_{\left\{X_{0:n} \in \bar{A}_n^0\right\}} \left(\sum_{x_{n+1}\in E} u(n+1, x_{n+1}) P(X_n, x_{n+1}) - u(n, X_n)\right)\right) = 0. \end{aligned}$$

EXEMPLE 2 : UN PROBLÈME DE RECRUTEMENT

- Je reçois, consécutivement, N candidats à un poste. Les circonstances m'imposent de décider tout de suite du recrutement (soit je recrute la personne que je viens de recevoir, soit je la refuse définitivement).
- La *seule* information que j'ai sur les candidats est leur classement.
- Je souhaite maximiser la probabilité de recruter le meilleur candidat.
- Quelle est la meilleure façon de s'y prendre ?

LE RÉSULTAT

- Il faut recevoir (environ) 37% des candidats (en fait $1/e$), puis choisir le premier candidat qui suit qui est meilleur que tous les précédents (le dernier si cela n'arrive jamais).
- On obtient ainsi un temps d'arrêt optimal.
- La probabilité d'obtenir le meilleur candidat est (environ) de 37%.
- Les transparents qui suivent expliquent comment arriver à ces résultats à l'aide de la théorie précédente.

Le modèle

- $\omega = (\omega_1, \dots, \omega_N)$ une permutation de $(1, \dots, N)$.
- ω_k le classement du $\#k$-ième individu dans la permutation.

$$\begin{array}{lccccccc} \textit{Indice} & (\#1 & \#2 & \dots & \#k & \dots & \#N) \\ \textit{Rang} & (\omega_1 & \omega_2 & \dots & \omega_k & \dots & \omega_N) \end{array}$$

- Ω_N l'ensemble des permutations de $(1, \dots, N)$ muni de la probabilité uniforme.
- B_n l'évenement "le n-ième candidat est le meilleur".
- On cherche un temps d'arrêt τ qui maximise $\mathbb{P}(B_\tau)$.

Où est la chaîne de Markov ?

- Un temps d'arrêt mais pour quel processus de Markov ?
- $R_k(\omega)$ le rang du $\#k$-ième individu parmi les k premiers individus.
- $B_n = \{R_n = 1, R_{n+1} > 1, \dots, R_N > 1\}$.
- Quelle est la loi de $(R_1, \dots, R_N)$?

UN EXEMPLE DE CALCUL DE R

- $\omega = (2\,3\,1\,4)$ donne $R = (1\,2\,1\,4)$.
- $R = (1\,2\,1\,4)$ donne $\#3 \leq \#1 \leq \#2 \leq \#4$

$$
\begin{aligned}
(1) &\to (\#1) && \#1 \text{ est le premier puisqu'il est tout seul !}\\
(1\,2) &\to (\#1 \leq \#2) && \#2 \text{ est le deuxième parmi les 2 premiers}\\
(1\,2\,1) &\to (\#3 \leq \#1 \leq \#2) && \#3 \text{ est le premier parmi les 3 premiers}\\
(1\,2\,1\,4) &\to (\#3 \leq \#1 \leq \#2 \leq \#4) && \#4 \text{ est le quatrième parmi les 4 premiers}
\end{aligned}
$$

On obtient la permutation de départ $\omega = (2\,3\,1\,4)$

Indice	(#1	#2	#3	#4)
Rang	(2	3	1	4)

EN Scilab

- À une permutation ω, on fait correspondre le vecteur des rangs d'insertion R.
- À un vecteur de rangs d'insertion , on fait correspondre une unique permutation ω.

Si vous avez un doute voici, deux fonctions Scilab qui font le job.

```
function [R] = Omega2R(omega)
  // Calcule les rang d'insertion pour un omega donne
  for n=[1:length(omega)] do
    y=gsort(omega(1:n),'g','i');// classe le vecteur omega(1:n) en croissant
    R(n)=find(omega(n)==y);// indice de omega(n) dans le tableau classe
                          // (son classement parmis les n premiers)
  end
endfunction

function [omega] = R2Omega(R)
  // Calcule omega connaissant les rangs d'insertion
  // J'insére n à l'indice R(n)
  omega=[];
  for n=[1:length(R)] do omega=[omega(1:R(n)-1),n,omega(R(n):n-1)];end;
  // On inverse la permutation
  for  n=[1:length(R)] do temp(omega(n))=n;end;
  omega=temp;
endfunction
```

Calcul de lois

- À un R correspond un et un seul ω, **donc**, pour $\alpha_k \in \{1, \dots, k\}$

$$P(R_1 = \alpha_1, \dots, R_N = \alpha_N) = \mathbb{P}(\{\omega\}) = \frac{1}{N!}.$$

- (Par contraction) les R_k suivent des lois uniformes sur $\{1, \dots, k\}$. Elles sont indépendantes.
- $S_k = \mathbf{1}_{\{R_k = 1\}}$ sont aussi des variables aléatoires indépendantes. Elles suivent des lois de Bernouilli de paramêtre $p_k = \mathbb{P}(S_k = 1) = 1/k$.
- La suite de variable $(S_1, \dots, S_N)$ suffira pour traiter le problème.

Le critère

- B_n l'évenement "le n-ième candidat est le meilleur".

$$B_n = \{R_n = 1, R_{n+1} > 1, \dots, R_N > 1\} = \{S_n = 1, S_{n+1} = 0, \dots, S_N = 0\}.$$

- τ un temps d'arrêt par rapport au processus $R = (R_1, \dots, R_N)$. À l'instant n, $(R_1, \dots, R_n)$ "contient toute l'information disponible".
- Pour un temps d'arrêt[2] τ, on va voir que

$$\mathbb{P}(B_\tau) = \mathbb{E}\left(\frac{\tau}{N} S_\tau\right).$$

ce qui permettra de mettre le problème "sous forme markovienne".

2. Ce n'est plus vrai sinon ... Exercice : trouver un contre-exemple.

PREUVE

$$\{\tau = n\} = \{(R_1, \dots, R_{n-1}, R_n) \in A_n\} = \{R_{1:n} \in A_n\}.$$

Notation : $R_{1:n} = (R_1, \dots, R_{n-1}, R_n)$.

$$\begin{aligned}\mathbb{P}(\tau = n, B_\tau) &= \mathbb{P}(\tau = n, B_n).\\ &= \mathbb{P}(\tau = n, R_n = 1, R_{n+1} > 1, \dots, R_N > 1)\\ &= \mathbb{P}(R_{1:n} \in A_n, R_n = 1, R_{n+1} > 1, \dots, R_N > 1)\\ &= \mathbb{P}(R_{1:n} \in A_n, R_n = 1)\mathbb{P}(R_{n+1} > 1, \dots, R_N > 1)\end{aligned}$$

car indépendance des vecteurs $R_{1:n}$ et $R_{n+1:N}$, puis par indépendance des R_n entre eux

$$\mathbb{P}(R_{n+1} > 1, \dots, R_N > 1) = \frac{n}{n+1}\frac{n+1}{n+2} \times \cdots \times \frac{N-1}{N} = \frac{n}{N}$$

PREUVE

$$\begin{aligned}\mathbb{P}(\tau = n, B_\tau) &= \frac{n}{N}\mathbb{P}(R_{1:n} \in A_n, R_n = 1) = \frac{n}{N}\mathbb{P}(\tau = n, R_n = 1)\\ &= \frac{n}{N}\mathbb{E}\left(\mathbf{1}_{\{\tau = n, R_n = 1\}}\right) = \mathbb{E}\left(\mathbf{1}_{\{\tau = n\}}\frac{n}{N}S_n\right)\\ &= \mathbb{E}\left(\mathbf{1}_{\{\tau = n\}}\frac{\tau}{N}S_\tau\right).\end{aligned}$$

En sommant pour n variant de 1 à N

$$\mathbb{P}(B_\tau) = \mathbb{E}\left(\frac{\tau}{N}S_\tau\right).$$

RÉSUMONS NOUS !

- $(S_1, \dots, S_N)$ est une suite de variables aléatoires indépendantes de Bernouilli $1/k$, **donc** une chaîne de Markov sur l'espace $E = \{0, 1\}$, non homogène, de matrice de transition, dépendant du temps, P_n

$$P_n(0,0) = P_n(1,0) = 1 - \frac{1}{n+1} = \frac{n}{n+1}$$
$$P_n(0,1) = P_n(1,1) = \frac{1}{n+1}$$

- On chercher à maximiser $\mathbb{P}(B_\tau) = \mathbb{E}\left(\frac{\tau}{N} S_\tau\right) = \mathbb{E}\left(f(\tau, S_\tau)\right)$ parmi tous les temps d'arrêt.

RÉSOLUTION

- On peut résoudre le problème grâce à (4)
- On calcule (on le fera demain) $(u(n, 0 \text{ ou } 1), 0 \leq n \leq N)$

$$\begin{cases} u(n,x) = \max\left\{\frac{n}{n+1} u(n+1,0) + \frac{1}{n} u(n+1,1), \frac{n}{N} x\right\}, n < N, \\ u(N,x) = x, \end{cases}$$

- Une fois ceci fait, un temps d'arrêt optimal est obtenu par

$$\tau = \inf\left\{n \geq 0, u(n, S_n) = \frac{n}{N} S_n\right\}.$$

- Noter que l'on a forcément $u(0,0) = u(0,1)$ et que cette valeur commune donne $\mathbb{P}(B_\tau) = \mathbb{E}\left(\frac{\tau}{N} S_\tau\right)$ pour le temps d'arrêt optimal.

RÉSOLUTION

- Dans ce cas, on peut mener des calculs explicites (voir [Delmas and Jourdain, 2006]) pour obtenir les résultats annoncés.
- Noter que l'équation s'écrit (comme $u(n,x) \geq 0$) :

$$\begin{cases} u(n,0) = \frac{n}{n+1}u(n+1,0) + \frac{1}{n}u(n+1,1), n < N, \\ u(n,1) = \max\left\{u(n,0), \frac{n}{N}x\right\}, n < N, \\ u(N,0) = 0, u(N,1) = 1. \end{cases}$$

- Il est facile de vérifier (par récurrence) à l'aide de l'équation que $u(n,x)$ ne peut être nul que si $n = N$ et $x = 0$.
- On ne peut donc exercer lorsque $S_n = 0$ (le nouveau candidat n'est pas le meilleur relatif) que lorsque $n = N$.
- τ est le premier instant où $S_n = 1$ et $u(n+1,1) = n/N$

BIBLIOGRAPHIE I

Delmas, J.-F. and Jourdain, B. (2006).
Modèles aléatoires.
Mathématiques & Applications. Springer-Verlag, Berlin.

Hill, T. (2009).
Savoir quand s'arrêter.
Pour La Science, 381.

Snell, J. L. (1952).
Applications of martingale system theorems.
Trans. Amer. Math. Soc., 73 :293–312.

RICHARD BELLMAN

- Sur l'origine de la programmation dynamique et du nom voir [Dreyfus, 2002]. "It was something not even a congressman could object to" selon Bellman.

SYSTÈME DYNAMIQUE CONTRÔLÉ

- $X_{n+1} = f(X_n, U_n, W_{n+1})$, à l'instant n on peut choisir un paramètre de contrôle U_n choisit dans un ensemble fini A.
- il est légitime de supposer que U_n soit fonction de la trajectoire connue à l'intant n, $(X_1, \ldots, X_n)$

$$U_n = \tilde{u}(n, X_1, \ldots, X_n).$$

- Evidemment X_n dépend de la suite des fonctions $\tilde{u} = (\tilde{u}(n, x_1, \ldots, x_n), n \geq 0)$ et devrait donc être noté $X_n^{\tilde{u}}$. Je ne le fait pas pour alléger les notations, mais il faut bien le garder en tête.

UN EXEMPLE : GESTION DE STOCK

- W_{n+1} représente la demande entre t et $t+1$
- $(W_n, n \geq 0)$ est supposé être une suite i.i.d., de loi donnée
- La commande (paramètre de contrôle) U_n décidée en n permettra de satisfaire la demande exprimée entre t et $t+1$
- Le stock (qui peut être négatif) à l'instant n, X_n vérifie

$$X_{n+1} = X_n + U_n - W_{n+1}$$

- C'est bien un système dynamique controlé.

APPROCHE FORMELLE

- On se donne une suite de fonctions (ou contrôles) $\tilde{u} = (\tilde{u}(n, x_1, \dots, x_n), n \geq 0)$ de E^n à valeurs dans A
- $(W_n, n \geq 1)$ une suite i.i.d. à valeurs dans F
- x_0 un point donné de E. On pose $X_0 = x_0$, puis itérativement on note $U_n = \tilde{u}(n, X_1, \dots, X_n)$ et l'on définit X_{n+1} par

$$X_{n+1} = f(X_n, U_n, W_{n+1}). \tag{7}$$

- Noter que X n'est pas (forcément) une chaîne de Markov.
- C'est le cas, lorsque $\tilde{u}$ ne dépend que de n et x_n, on parle alors de contrôle "feedback". On s'intéresse à savoir si le contrôle optimal est de ce type.
- La chaîne est homogène uniquement si u ne dépend que de x_n.

SYSTÈME DYNAMIQUE ET CHAÎNE DE MARKOV CONTRÔLÉE

- Au système dynamique contrôlé (7) on peut associer une famille de matrice de transition $(P_u(x, y), x, y \in E, u \in A)$, en posant

$$P_u(x, y) = \mathbb{P}(f(x, W_1, u) = y)$$

- A u fixé, $P_u(x, y)$ est une matrice de transition d'un chaine de Markov.
- La loi de de la chaîne contrôlée se calcule en fonction de $P_u(x, y)$ et de $\tilde{u}$

$$\mathbb{P}(X_{0:n+1} = x_{0:n+1}) = \mathbb{P}(X_{0:n} = x_{0:n}) P_{\tilde{u}(n, x_{1:n})}(x_n, x_{n+1}) \tag{8}$$

Preuve

$$\begin{aligned}
&\mathbb{P}\left(X_{0:n+1} = x_{0:n+1}\right) \\
&\quad = \mathbb{P}\left(X_{0:n} = x_{0:n}, f(X_n, \tilde{u}(X_{0:n}), W_{n+1}) = x_{n+1}\right) \\
&\quad = \mathbb{P}\left(X_{0:n} = x_{0:n}, f(x_n, \tilde{u}(x_{0:n}), W_{n+1}) = x_{n+1}\right)
\end{aligned}$$

- $f(x_n, u(x_{0:n}), W_{n+1})$ indépendante du vecteur $(W_1, \dots, W_n)$
- $X_0, \dots, X_n$ fonctions de $(W_1, \dots, W_n)$.
- Donc $f(x_n, u(x_{0:n}), W_{n+1})$ indépendante de $X_{0:n}$.

$$\begin{aligned}
&\mathbb{P}\left(X_{0:n+1} = x_{0:n+1}\right) \\
&\quad = \mathbb{P}\left(X_{0:n} = x_{0:n}\right) \mathbb{P}\left(f(x_n, u(x_{0:n}), W_{n+1}) = x_{n+1}\right) \\
&\quad = \mathbb{P}\left(X_{0:n} = x_{0:n}\right) P_{u(x_{0:n})}(x_n, x_{n+1})
\end{aligned}$$

Système dynamique et Chaîne de Markov contrôlée

- On voit grâce à cette égalité, que la loi de la chaîne contrôlée s'exprime à l'aide de la famille de fonction $\tilde{u}$, de la loi en X_0 et de la famille de matrice de transition $P_u(x, y)$. On n'a pas besoin de connaître la fonction f pour spécifier complètement le problème (en loi), mais seulement de la famille des $P_u(x, y)$.
- On peut prendre cette dernière égalité comme définition formelle d'une chaîne de Markov contrôlée de famille de transition $P_u(x, y)$ (c'est l'équivalent de la propriété de Markov classique lorsque il n'y a pas de contrôle). Cela évite d'avoir à préciser un système dynamique contrôlé correspondant à P_u.
- On écrit souvent pour décrire ce contexte (avec un léger abus de notation)

$$\mathbb{P}(X_{n+1} = x_{n+1} | X_n = x_n, U_n = u) = P_u(x_n, x_{n+1}).$$

Le coût

Pour spécifier le coût de la stratégie $\tilde{u}$, on suppose que

- le choix de $U_j = \tilde{u}(j, X_{1:j})$ à l'instant j, coûte $l(j, X_j, U_j)$
- la valeur en l'instant final N coûte (ou rapporte en changeant de signe) $h(X_N)$
- le coût d'une stratégie $\tilde{u}$ s'écrit

$$j(\tilde{u}) = \sum_{j=0}^{N-1} l(j, X_j, U_j) + h(X_N)$$

- et son coût moyen (dont on remarquera qu'il ne dépend que de la loi de X et de $\tilde{u}$)

$$J(\tilde{u}) = \mathbb{E}\left(\sum_{j=0}^{N-1} l(j, X_j, U_j) + h(X_N) \right).$$

Le coût : Gestion de stock

- C_M represente le coût du à la demande non satisfaite
- C_s represente le coût des invendus (stockage)
- c le coût d'achat d'une unité de stock, c_F un coût fixé (éventuellement nul).
- X_n représente le stock (algébrique) en n, U_n la quantité acheté en n (disponible en $n+1$), W_{n+1} la demande en $n+1$

$$X_{n+1} = X_n + U_n - W_{n+1}.$$

- Le coût l pour $0 < n < N$ est donné par

$$l(n, X_n, U_n) = c_F \mathbf{1}_{\{U_n > 0\}} + cU_n + C_s(X_n)_+ + C_M(-X_n)_+.$$

- Le coût initial $l(0, X_0, U_0) = c_F \mathbf{1}_{\{U_0 > 0\}} + cU_0$.
- Et le coût final $h(X_N) = C_s(X_N)_+ + C_M(-X_N)_+$.

La question

- Trouver le choix de $\tilde{u}$ qui minimise $J(\tilde{u})$ parmi tous les choix possibles. La réponse est (étonnamment) simple (elle est due à Bellman dans les années 1950-1955).

La réponse

- Il faut résoudre l'équation de (Hamilton-Jacobi-)Bellman (HJB) suivante

$$\begin{cases} v(n,x) = \min_{u\in A}\left\{\sum_{y\in E} P_u(x,y)v(n+1,y) + l(n,x,u)\right\}, \\ \quad n < N \\ \\ v(N,x) = h(x), \end{cases}$$

  qui définit une *unique* fonction v (c'est facile à prouver dans notre formalisme puisque E et A sont des ensembles finis).
- Noter que

$$\sum_{y\in E} P_u(x,y)v(n+1,y) = \mathbb{E}(v(n+1, f(x,u,W_1)),$$

  forme pouvant être utile pour les calculs. Mais v ne dépend que de P_u et non de f.

La réponse

- Une fois la fonction v obtenue, on calcule un (celui que l'on veut, s'il y en a plusieurs) $\hat{u}(n,x)$ qui réalise, x et n étant fixés :

$$\min_{u\in A}\left\{\sum_{y\in E} P_u(x,y)v(n+1,y) + l(n,x,u)\right\}$$

- Ce choix $\hat{u} = (\hat{u}(n,x_n), n \geq 0)$ est optimal (mais pas forcement unique) parmi tous les contrôles de type $\tilde{u} = (\tilde{u}(n,x_1,\ldots,x_n), n \geq 0)$.
- Noter que ce contrôle optimal est sous une forme feedback mais qu'il est optimum parmi tous les contrôles possibles.
- La preuve suit. Elle est proche dans l'esprit des démonstrations déjà faites. Ceux sont des preuves de type "théorème de vérification" qui utilisent (sans expliciter la notion) des techniques de martingales.
- On peut généraliser l'approche à des cas bien plus complexes, ce qui a donné lieu (et donne encore lieu) à des développements techniques très nombreux.

Le théorème

- On se donne un système dynamique contrôlé spécifié par $(P_u(x,y), x,y \in E, u \in A)$ et la valeur initiale de $X_0 = x_0 \in E$.
- On cherche à minimiser $J(\tilde{u}) = \mathbb{E}(j(\tilde{u}))$, parmi tous les contrôles $\tilde{u} = (\tilde{u}(n,x_1,\ldots,x_n), 0 \leq n \leq N-1)$ avec

$$j(\tilde{u}) = \sum_{j=0}^{N-1} l(j,X_j,U_j) + h(X_N).$$

- Soit v la solution unique de l'équation de HJB

$$\begin{cases} v(n,x) = \min_{u\in A}\left\{\sum_{y\in E} P_u(x,y)v(n+1,y) + l(n,x,u)\right\}, \ n < N \\ v(N,x) = h(x), \end{cases} \tag{9}$$

- Alors $\min_{\tilde{u}} J(\tilde{u}) = v(0,x_0) = J(\hat{u})$ où $\hat{u}(n,x)$ realise le minimum.

PREUVE I

On va montrer que quel que soit le choix de $\tilde{u}$, on a, si

$$U_n = \tilde{u}(n, X_1, \dots, X_n),$$

$$\mathbb{E}(v(n, X_n) - l(n, X_n, U_n)) \leq \mathbb{E}(v(n+1, X_{n+1})). \tag{10}$$

et que (noter que les 2 processus X sont différents !)

$$\mathbb{E}(v(n, X_n) - l(n, X_n, \hat{u}(n, X_n))) = \mathbb{E}(v(n+1, X_{n+1})). \tag{11}$$

On obtient, à partir de (10), par récurrence, en utilisant que $v(N, x) = h(x)$:

$$v(0, x_0) \leq \mathbb{E}\left(\sum_{j=0}^{N-1} l(j, X_j, U_j) + h(X_N)\right) = J_{\tilde{u}},$$

PREUVE II

et à partir de (11)

$$v(0, x_0) = \mathbb{E}\left(\sum_{j=0}^{N-1} l(j, X_j, \hat{u}(j, X_j)) + h(X_N)\right) = J_{\hat{u}}.$$

Ce qui prouve que $v(0, x_0) = J_{\hat{u}} \leq J_{\tilde{u}}$ pour tout $\tilde{u}$, i.e. l'optimalité de $\hat{u}$, sous réserve de montrer (10) et (11).
Pour montrer (10), on remarque que

$$\begin{aligned}
&\mathbb{E}\left(v(n+1, X_{n+1})\right) \\
&= \sum_{x_{0:n+1} \in E^{n+2}} \mathbb{P}(X_{0:n+1} = x_{0:n+1}) v(n+1, x_{n+1}) \\
&= \sum_{x_{0:n+1} \in E^{n+2}} \mathbb{P}(X_{0:n} = x_{0:n}, X_{n+1} = x_{n+1}) v(n+1, x_{n+1})
\end{aligned}$$

PREUVE III

puis en utilisant (8), puis l'équation de HJB (9)

$$\begin{aligned}
&\mathbb{E}\left(v(n+1, X_{n+1})\right) \\
&= \sum_{x_{0:n} \in E^{n+1}} \mathbb{P}(X_{0:n} = x_{0:n}) \sum_{x_{n+1} \in E} P_{\tilde{u}(n,x_{1:n})}(x_n, x_{n+1}) v(n+1, x_{n+1}) \\
&\geq \sum_{x_{0:n} \in E^{n+1}} \mathbb{P}(X_{0:n} = x_{0:n}) \left[v(n, x_n) - l(n, x_n, \tilde{u}(n, x_{1:n}))\right] \\
&= \mathbb{E}\left(v(n, X_n) - l(n, X_n, \tilde{u}(n, X_{1:n}))\right).
\end{aligned}$$

Pour montrer (11), il suffit de remarquer que, l'inégalité est remplacée par une égalité lorsque l'on utilise $\hat{u}(n, X_n)$ (qui réalise alors le minimum).

APPLICATION : GESTION DE STOCK

- $(W_n, n \geq 1)$ est supposé être une suite i.i.d. à valeurs entières dans $F = \{0, \ldots, M_F\}$.
- La commande U_n décidée en n qui sera livrée entre t et $t+1$, $A = \{0, \ldots, M_A\}$.
- Le stock à l'instant n, X_n vérifie

$$X_{n+1} = X_n + U_n - W_{n+1} = f(X_n, U_n, W_{n+1}).$$

- Le coût l pour $0 < n < N$ est donné par

$$\begin{aligned}
l(0, X_0, U_0) &= c_F \mathbf{1}_{\{U_0 > 0\}} + cU_0 \\
l(n, X_n, U_n) &= c_F \mathbf{1}_{\{U_n > 0\}} + cU_n + C_s(X_n)_+ + C_M(-X_n)_+ \\
h(X_N) &= C_s(X_N)_+ + C_M(-X_N)_+
\end{aligned}$$

- $J(\tilde{u}) = \mathbb{E}\left(\sum_{j=0}^{N-1} l(j, X_j, U_j) + h(X_N)\right) = \mathbb{E}(j(\tilde{u}))$.

GESTION DE STOCK : UN SEUL PAS DE TEMPS

- *Deux* instants 0 et 1, mais *un* pas de temps.
- $W = W_1$, $X_0 = x$, la commande $U_0 = u(x)$ décidée en 0.
- Le stock à l'instant 1, X_1 vérifie $X_1 = x + u - W$.
- Le coût initial, $l(0, x, u) = c_F \mathbf{1}_{\{u > 0\}} + cu$
- Le coût final $h(X_1) = C_s(X_1)_+ + C_M(-X_1)_+$.
- $\tilde{J}(u) = \mathbb{E}\left(\tilde{j}(u)\right)$ avec

$$\tilde{j}(u) = c_F \mathbf{1}_{\{u > 0\}} + cu + C_s(X_1)_+ + C_M(-X_1)_+.$$

- On va voir que la commande optimale est de la forme

$$\hat{u}(x) = (S - x)\mathbf{1}_{\{x \leq s\}},$$

UN SEUL PAS DE TEMPS : RÉSOLUTION

- $\tilde{J}(u)$ s'écrit $\tilde{J}(u) = c_F \mathbf{1}_{\{u > 0\}} + J(u + x) - cx$ avec

$$\begin{aligned} J(u) &= \mathbb{E}\left(cu + C_s(u - W)_+ + C_M(W - u)_+\right), \\ &= C_M\mathbb{E}(W) + (c - C_M)u + (C_M + C_s)\mathbb{E}\left((u - W)_+\right). \end{aligned}$$

- On suppose $C_M \geq c$ (sinon J est croissante et ne rien faire ($u = 0$) est optimal !).
- $S = \arg\min J(u)$, J est décroissante avant S et croissante après. Si $F(z) = \mathbb{P}(W \leq z)$,

$$J(u) = C_M\mathbb{E}(W) + (c - C_M)u + (C_M + C_s)\int_0^u F(z)dz.$$

- $\tilde{J}(u) = c_F \mathbf{1}_{\{u > 0\}} + J(u + x) - cx$.

UN SEUL PAS DE TEMPS : RÉSOLUTION

- Alors, si $x \geq S$, il est optimal de ne rien faire. En effet, comme J est croissante après S et $c_F \geq 0$, pour $u \geq 0$

$$\tilde{J}(0) = J(x) - cx \leq J(x+u) - cx + c_F = \tilde{J}(u).$$

- Si $x \leq S$ et *si on commande*, il faut (forcement) commander $u = S - x$ puisque, pour tout u

$$\tilde{J}(S-x) = J(S) - cx + c_F \leq J(u+x) - cx + c_F$$

- Comme ne rien faire coûte $J(x) - cx$, il faut porter son stock à S si et seulement si x vérifie $J(x) - cx \geq J(S) - cx + c_F$.
- Il est facile de vérifier (par décroissance de J avant S) que

$$\{x \leq S, J(x) \geq c_F + J(S)\} = \{x \leq s\},$$

avec $s = \sup\{z \leq S, J(z) \geq c_F + J(S)\}$

- La commande optimale est donnée par

$$\hat{u}(x) = (S-x)\mathbf{1}_{\{x \leq s\}},$$

où $S = \arg\min J(u)$ et $s = \sup\{z \leq S, J(z) \leq c_F + J(S)\}$.

GESTION DE STOCK : NOMBRE DE PAS ARBITRAIRES

- lorsque le nombre de pas T est arbitraire, on peut sous certaines hypothèses (dont une variante affaiblie de la notion de convexité, la C-convexité) montrer que le contrôle optimal est toujours du type $\hat{u}(n,x) = (S_n - x)\mathbf{1}_{\{x \leq s_n\}}$. C'est assez délicat (voir [Delmas and Jourdain, 2006], chapitre 3 et les références citées), même si les outils restent élèmentaires.
- Ceci dit l'équation de Bellman, dans le cas où l'espace d'état et l'espace de contrôle sont finis, permet toujours de calculer la solution optimale.
- On peut le vérifier "expérimentalement" sur la solution calculée informatiquement (voir expérience numérique).

GESTION DE STOCK : GÉNÉRALISATION

- L'extension aux cas des espaces d'état/de contrôle arbitraires reste mumériquement complexe (i.e. "presque" inacessible, on peut faire des choses, mais via une simplifcation drastique du problème).
- Un bon exemple de problème difficile est celui de la gestion de la production d'un groupe de centrale électrique pour répondre à une demande aléatoire bien sûr).

BIBLIOGRAPHIE I

Dreyfus, S. (2002).
Richard bellman on the birth of dynamic programming.
Operations Resarch, 50(1) :48–51.

Chaîne de Markov : calcul de loi

Bernard Lapeyre, Ecole des Ponts ParisTech.

Juin 2017

Les programmes suivant on été prévu pour être exécuté à l'aide `Scicoslab`. Ce programme peut être téléchargé librement sur `www.scicoslab.org`. Il est disponible sur MacOSX, Windows ou Linux (Debian, Ubuntu ou Fedora). Noter toutefois qu'il faut installer le server `X`, `XQuartz` sur MacOSX. Ces programmes peuvent aussi fonctionner, moyennant de légère modification, avec la version officielle de `Scilab`.

Le fichier source en `Scilab` correspondant à ce document est accessible sur

`http://cermics.enpc.fr/~bl/Liesses/markov-scilab.sci`.

Il est partiellement mais pas totalement corrigé. Le correction complète est accessible sur

`http://cermics.enpc.fr/~bl/Liesses/markov-scilab_corrige.sci`.

1 Un test d'équirépartition

Simulation

On commence par simuler des tirages à pile ou face répétés. On utilise la fonction `scilab`, `grand` avec l'option `"bin"` (loi binomiale) mais avec un seul tirage (donc loi de Bernouilli). A partir de ce tirage on calcule le nombre de pile consécutifs maximum dans le vecteur.

```
function X=tirage_pf(n,p)
// Effectue n tirage a Pile (P) ou face (F) (p,1-p)
  X=grand(1,n,"bin",1,p);
  X=string(X);// transforme le tableau de nombres en
              // tableau de caractères
  X=strsubst(X,'0','F');// remplace les 0 par des F
  X=strsubst(X,'1','P');// remplace les 1 par des P
endfunction

function [MAX] = max_length(U)
// Calcule le nombre maximum de 1 consecutifs
// dans la suite U
  MAX=0;N=0;
  for n=[1:size(U,'*')] do
    if U(n)=='P' then A COMPLETER else A COMPLETER end;
    MAX=max(MAX,N);
  end
endfunction
```

On aura besoin de tracer des histogrammes.

```
function H=histo_discret(samples, maxsize, varargin)
// histogramme de tirages selon une loi discrète à valeurs entières
// supposé prendre des valeurs entre 0 et max.
// Si tracer_histo=%f pas de dessin
  H=0
  for k=0:maxsize
    // Calcul du nbre de tirages valant k / Taille
    H(k+1)=length(find(samples==k)) ./ size(samples,'*');
  end;

  [lhs,rhs]=argn(0);
  if (rhs == 3) & (varargin(1)==[%t]) then
    xbasc;plot2d3(0:maxsize,H);
    f=gcf();
    Dessin=f.children(1).children(1).children(1);
    Dessin.thickness=10;Dessin.foreground=5;
  end;
endfunction
```

On calculer par simulation une approximation de la loi du nombre maximum de piles consecutifs (un histogramme de tirages i.i.d.).

```
function main_1(load_flag)
  p=1/2;
  // On teste cette fonction avec N=20
  // rand(1:20) renvoie 100 tirages uniformes sur l'intervalle [0,1]
  U=tirage_pf(20,p)// 20 tirages a pile ou face 1/2,1/2  (1=Pile,0=Face)
  max_length(U)// nombre mximum de P consecutifs

  // On effectue 1000 tirages avec N=100
  N=100;X=0;
  Taille=1000;// nbre de simulation
  for i=[1:Taille] do
    U=tirage_pf(N,p);
    X(i)=max_length(U);
  end;

  // On trace l'histogramme si load_flag=%t
  histo_discret(X,20,load_flag)
endfunction
```

On simule des trajectoires du processus (“en dents de scie”) X.

```
function [X] = trajectoire(U)
// On calcule la trajectoire de X
  X=0;val=0;
  for n=[1:prod(size(U))] do
    if U(n)=='P' then A COMPLETER else A COMPLETER end;
    X(n)=val;
  end
endfunction

function main_2(load_flag)
  N=100;
  p=1/2;
  rectangle=[1,0,N,8];

  // On trace une trajectoire de X
  U=tirage_pf(N,p);
  X=trajectoire(U);
  xbasc;plot2d2(1:N,X,rect=rectangle);

  // 3 trajectoires
  for i=1:4 do
    U=tirage_pf(N);
    plot2d2(1:N,trajectoire(U),rect=rectangle,style=i);
    xclick;// attend un click dans la fenêtre graphique
  end;
endfunction;
```

On calcule exactement la probabilité de voir au moins l piles consécutifs.

```
function Pro= proba(N,l,p)
  // Calcule la probabilite de voir au moins l piles consecutifs
  // dans N tirages a pile (p) ou face (1-p)

        // la matrice de transition de la chaîne
        // de taille (l+1,l+1)
        P=[p*diag(ones(l,1));zeros(1,l-1),1];
        P=[[(1-p)*ones(l,1);0],P];
        PN=P^N;
        Pro = QUE VAUT LA PROBA DE VOIR AU MOINS l PILES;
endfunction
```

On en déduit la loi du nombre maximum de piles consécutifs.

```
function [loi]=calculer_loi(N,p)
  MAXMAX=30;// a choisir assez grand (mais pas trop ...)
  // attention au decalage de 1 pour les vecteurs :
  //      loi(1)=P(X=0), ..., loi(n+1)=P(X=n)

  previous=1;// proba d'avoir au moins 0 pile = 1 !
  // le support de la loi est [0,1,...,N] que l'on tronque en MAXMAX
  for l=0:min(N,MAXMAX) do
    current=A COMPLETER;// proba d'avoir au moins l+1 pile
    loi(l+1)=previous - current;// proba d'avoir exactement l pile
    previous=current;
  end
  loi=loi';// c'est plus joli comme ca !
endfunction
```

On teste la cohérence des résultats donnés par le programme dans des cas simples.

```
function main_3(load_flag)

  // On teste avec N=1 et N=2, p=1/2
  // Pour N=1, 0 pile avec proba 1/2 et 1 pile avec proba 1/2
  calculer_loi(1,1/2)
  // Pour N=2, on doit trouver (1/4,1/2,1/4) pour (0,1,2)
  calculer_loi(2,1/2)
  // en principe ca marche ...

  N=100;p=1/2;
  loi=calculer_loi(N,p);
  sum(loi)// on verifie que ca somme a 1

  // dessin
  // ATTENTION on est decale de 1, donc 0:20 devient 1:21
  xbasc;plot2d3(0:20,loi(1:21));
  f=gcf();Dessin=f.children(1).children(1).children(1);Dessin.thickness=10;

  // comparaison avec les simulations

  Taille=10000;
  for i=[1:Taille] do
    U=tirage_pf(N);
    X(i)=max_length(U);
  end

  histo=0;
  for i=0:20 do
    histo(i+1)=sum(X==i)/Taille;
  end
  histo=histo';

  epsilon=norm(histo(1:21)-loi(1:21))
          // doit etre "petit", pour bien faire il faudrait faire un
          // test du ξ^2 pour savoir ce que "petit" veut dire ici.
  printf("epsilon=%f, petit en principe.\n",epsilon)
endfunction
```

Test du critère

On vérifie que notre critère fonctionne pas trop mal.

```
function main_4(load_flag)
// Test du critère lorsque les tirages sont aléatoires
// Ca marche "souvent" mais pas "toujours".
  N=100;
  p=1/2;
  Taille=10;
  for i=[1:Taille] do
    U=tirage_pf(N,p);
    if max_length(U) >= 4 then
      printf("OK\n");
    else
      // Ca arrive "rarement" mais ca arrive 3 fois sur 100
      printf("test négatif\n");
    end
  end;
endfunction;
```

Comment le loi varie t'elle en fonction de N ?

On regarde ce qui se passe lorsque N devient grand.

```
function main_5(load_flag)
// Calcul du maximum de vraisemblance de la loi
// imax = indice du maximum,  m = le maximum

  printf('N: indice du max -> maximum\n');
  for N=[10,100,1000] do
    loi=calculer_loi(N,p);
    [m,imax]= ON CHERCHE LE MAX DE LA LOI ET L'INDICE DU MAX
    imax=imax-1;// A cause du déclage de 1
    printf('%d: %d -> %f\n',N,imax,m);
  end
endfunction
```

```
function s=moyenne(loi)
  N=size(loi,'*')-1;
  s = CALCULER LA MOYENNE D'UNE V.A. DE LOI 'loi' ?
endfunction

function main_5_bis(load_flag)
// La moyenne varie (approximativement) comme C log(N).
// Ca peut se prouver.
  i=0;
  for N=[10,50,100,500,1000,5000,10000] do
    i=i+1;
    loi=calculer_loi(N,p);
    x(i)=log(N);
    y(i)=moyenne(loi);
    printf('%d: %f ?=? %f\n',N,y(i),x(i));
  end;
  plot2d(x,y);
endfunction
```

2 Le modèle de Cox-Ross

Simulation du modèle

```
function X=simul_cox_ross(N,x_0,p,u,d)
  U=grand(1,N,"bin",1,p);// tirages a pile ou face (p,1-p)
  X=x_0 * [1, cumprod(u^U .* d^(1-U))];
  // Plus long mais plus comprehensible ...
  // X(1)=x_0;
  // for i=1:prod(length(U)) do
  //  if U(i) then
  //    X(i+1)=X(i)*u
  //  else
  //    X(i+1)=X(i)*d;
  //  end
  //end;
endfunction;

function main_6(load_flag)
  N=50;
  sigma=0.3;
  p=1/2;u=1-sigma/sqrt(N);d=1+sigma/sqrt(N);
  x_0=1;

  X=simul_cox_ross(N,x_0,p,u,d);
  plot2d2(0:N,X);
endfunction
```

```
K=100;

function res=f(x)
  res=max(x-K,0);
endfunction
```

```
function res=prix_recursif(x,k,N)
  if k==N then res=f(x); return; end;
  res = RECOPIER L'EQUATION DU COURS !;
endfunction

function res=prix_slow(x,N)
  res=prix_recursif(x,0,N);
endfunction
```

```
N=10;

// On calcule les paramètre pour converger vers le modèle de Black et Scholes
sigma=0.3;
p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);

prix_slow(100,N);// Faire N=20 pour savoir ce que slow veut dire !
```

```
function res=inc(n);
// Permet de faire comme si les tableaux était indicés
// à partir de 0 (et non de 1)
 res=n+1;
endfunction;

function res=prix(x_0,N)
  // U=zeros(inc(N),inc(N));
  U=zeros(N+1,N+1);
  for k=[0:N] do
    // U(inc(N),inc(k)) = f(x_0 * u^k * d^(N-k));
    U(N+1,k+1) = f(x_0 * u^k * d^(N-k));
  end;

  for n=[N-1:-1:0] do // le temps decroit de N-1 a 0
    U(n+1,1:n+1)=RECOPIER L'EQUATION DU COURS;// ATTENTION AU DECALAGE DE 1
    // Version "vectorisée" qui fait la même chose que
    //     for k=[0:n] do
    //       U(inc(n),inc(k)) = p*U(inc(n+1),inc(k+1))+(1-p)*U(inc(n+1),inc(k));
    //     end;
  end;
  // res=U(inc(0),inc(0));
  res=U(1,1);
endfunction;

function main_7(load_flag)
  N=10;
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;x_0=100;

  prix(x_0,N)

  // Les deux algos font ils le même chose ?
  // on verifie : prix_slow(x_0,N) \approx prix(x_0,N)
  printf('différence entre les 2 resultats: %e\n', ...
          abs(prix_slow(x_0,N) - prix(x_0,N)));
endfunction
```

Convergence lorsque N est grand

Que se passe t'il pour N grand ?

```
function main_8(load_flag)
// Avec cet algorithme on peut augmenter N
// mais il faut renormaliser convenablement u et d pour
// rester borné.
// Essayer avec N=10,100,200,...,1000
  sigma=0.6;
  couleur=1
  for N=[3,5,10,20,50,100,200] do
    d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);

    n=0;
    for x=[50:150] do
      n=n+1;
      courbe(n)=prix(x,N);
    end

    couleur=couleur+1;
    plot2d([50:150],courbe,style=couleur);
    plot2d([50:150],max([50:150]-K,0));
  end
  // Ca converge, mais vers quoi ?
endfunction
```

Un cas numériquement délicat : les options sur moyenne

```
function res=f_moy(x,s); res=max((s/N)-K,0);endfunction

function res=prix_moyenne(x,s,k,N)
  if k==N then res=f_moy(x,s); return; end;
  res = A COMPLETER;
endfunction

function res=prix_slow_moyenne(x,N)
  res=prix_moyenne(x,x,0,N);
endfunction

function main_9(load_flag)
  N=10;sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  x_0=100;K=100;

  // Ca marche mais ce n'est pas très efficace ...
  printf('Prix option sur moyenne: %f\n',prix_slow_moyenne(x_0,N));
endfunction
```

```
function liste=liste_moyenne_rec(x,s,k,N)
// On constitue la liste des points visités par la chaine
// Si un point est visité deux fois, il y figure 2 fois.
  if k==N then
    liste=[x;s];    //printf("%d: %f %f\n",x,s);
    return;
  end;
  liste = A COMPLETER;
endfunction

function liste=liste_moyenne(x,N)
  // On part de (x,s=x) a l'instant 0
  liste=liste_moyenne_rec(x,x,0,N)
endfunction

function main_10(load_flag)
  x_0=100;
  liste=liste_moyenne(x_0,N);

  // Tri des points selon les valeurs de la somme.
  // Les valeurs de x peuvent etre egales, mais pas celle de s.
  // Nous allons le verifier.
  [y,p]=sort(liste(2,:));
  liste1=liste(:,p);

  // On regarde si tous les points sont differents
  // en parcourrant le tableau ainsi classé
  epsilon=0.001;
  Taille=size(liste);
  exception=[];
  for i=[1:Taille(2)-1] do
    if (norm(liste1(:,i)-liste1(:,i+1)) < epsilon) then
      printf("Warning: (%f,%f) ~ (%f,%f)\n",...
                 liste1(1,i),liste1(2,i),liste1(1,i+1),liste1(2,i+1));
      exception=[exception,[liste1(1,i),liste1(2,i),liste1(1,i+1),liste1(2,i+1)]];
    end;
  end;
  if size(exception,'*') == 0 then
     printf("Aucun point n''est dupliqué.");
  end
endfunction
```

Chaîne de Markov : arrêt optimal

Bernard Lapeyre, Ecole des Ponts ParisTech.

Juin 2017

Le fichier source en Scilab correspondant à ce document est accessible sur

http://cermics.enpc.fr/~bl/Liesses/arret-scilab.sci.

Il est partiellement mais pas totalement corrigé. Le correction complète est accessible sur

http://cermics.enpc.fr/~bl/Liesses/arret-scilab_corrige.sci.

1 Calcul du prix du put américain

1.0.1 Préliminaires et rappels

On aura besoin de tracer des histogrammes.

```
function H=histo_discret(samples, maxsize, varargin)
// histogramme de tirages selon une loi discrète à valeurs entières
// supposé prendre des valeurs entre 0 et max.
// Si varargin=\%t on trace l'histogramme.
  H=0
  for k=0:maxsize
    // Calcul du nbre de tirages valant k / Taille
    H(k+1)=length(find(samples==k)) ./ size(samples,'*');
  end;

  [lhs,rhs]=argn(0);
  if (rhs == 3) & (varargin(1)==[%t]) then
    xbasc;plot2d3(0:maxsize,H);
    f=gcf();
    Dessin=f.children(1).children(1).children(1);
    Dessin.thickness=10;Dessin.foreground=5;
  end;
endfunction
```

Pour le modèle de Cox-Ross, on a vu comment calculer des prix d'options européennes (call et put) qui se ramène à des calculs d'espérance pour une chaîne de Markov.

```
// Payoff du put
function res=gain_put(x); res=max(K-x,0);endfunction

// Payoff du call
function res=gain_call(x); res=max(x-K,0);endfunction

// Une fonction pour gerer le décalge de 1 dans les vecteurs
function res=inc(n); res=n+1; endfunction;

function res=prix_eu(x_0,N,gain)
// Calcul du prix europeen a l'instant 0
  // ATTENTION AU DECALAGE DE 1 EN TEMPS ET EN ESPACE
  // la fonction "inc" tente de le rendre indolore ...
  U=zeros(inc(N),inc(N));
  for k=[0:N] do
    U(inc(N),inc(k)) = gain(x_0 * u^k * d^(N-k))/(1+r)^N;
  end;
  for n=[N-1:-1:0] do // le temps decroit de N-1 a 0
    U(inc(n),inc(0):inc(n)) =  ...
        p*U(inc(n+1),inc(1):inc(n+1))+(1-p)*U(inc(n+1),inc(0):inc(n));
  end;
  res=U(inc(0),inc(0));
endfunction;

function res=prix_eu_n(n,x_0,N,gain)
// Calcul du prix a l'instant n
  res=prix_eu(x_0,N-n,gain);
endfunction
```

```
function res=prix_recursif_am(x,k,N,gain)
  if k==N then res=gain(x); return; end;
  res = p * prix_recursif_am(x*u,k+1,N) + ...
                (1-p) * prix_recursif_am(x*d,k+1,N);
  res = res  / (1+r);
  res = max( res, gain(x) );
endfunction

function res=prix_slow_am(x,N,gain)
  res=prix_recursif_am(x,0,N,gain);
endfunction

function res=prix_am(x_0,N,gain)
// Calcul du prix americain a l'instant 0
  U=zeros(N+1,N+1);
  x=x_0 * u^[0:N] .* d^[N:-1:0]; // les N+1 points de calcul en N
  U(N+1,1:N+1)=gain(x);// Valeur de U en N

  // ATTENTION AU DECALAGE DE 1 en espace et en temps
  for n=[N-1:-1:0] do // le temps decroit de N-1 a 0
    U(n+1,1:n+1)=p*U(n+2,2:n+2) + (1-p)* U(n+2,1:n+1);
    U(n+1,1:n+1)= U(n+1,1:n+1) / (1+r);// actualisation
    x = x_0 * u^[0:n] .* d^[n:-1:0]; // les points de calcul en n
    U(n+1,1:n+1)= QUOI DE NOUVEAU POUR UNE OPTION AMERICAINE ?
  end;
  res=U(1,1);
endfunction;

function res=prix_am_n(n,x_0,N,gain)
  res=prix_am(x_0,N-n,gain);
endfunction
```

On compare les 2 méthodes pour vérifier que tout fonctionne.

```
function main_1(load_flag)
  // qques tests pour voir si ca marche

  sigma=0.3;
  r_0=0.1;

  N=10;

  r=r_0/N;
  d=1-sigma/sqrt(N);
  u=1+sigma/sqrt(N);

  p=1/2;// p= (1+r-d)/(u-d); en principe pour Cox-Ross

  K=100;x_0=100;

  prix_am(x_0,N,gain_put)
  prix_slow_am(x_0,N,gain_put)

  // Les deux algos font ils le même chose ?
  // on verifie : prix_slow(x_0,N) \approx prix(x_0,N)
  abs(prix_slow_am(x_0,N,gain_put) - prix_am(x_0,N,gain_put)) <= 10^(-10)

  N=1000;
  sigma=0.3;
  r_0=0.1;

  r=r_0/N;
  d=1-sigma/sqrt(N);
  u=1+sigma/sqrt(N);
  p=1/2;

  K=100;x_0=100;

  prix_am(x_0,N,gain_put)
endfunction;
```

On trace les courbes de prix américaines et européennes que l'on supperpose au "payoff".

```
function main_2(load_flag)
  // tracé de courbes
  N=50;
  vmin=50;
  vmax=150;
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;x_0=100;
  r_0=0.1;
  r=r_0/N;

  n=0;
  for x=[vmin:vmax] do
    n=n+1;
    courbe_am(n)=prix_am(x,N,gain_put);
    courbe_eu(n)=prix_eu(x,N,gain_put);
  end
  // On compare les courbes "Américaines" et "Euroéennes"
  plot2d([vmin:vmax],courbe_eu,style=2);
  plot2d([vmin:vmax],courbe_am,style=3);
  plot2d([vmin:vmax],gain_put([vmin:vmax]),style=4);
endfunction;
```

On regarde comment le prix évolue au fils du temps.

```
function main_3(load_flag)
  // évolution de la courbe en fonction de n
  for n=[0,10,20,30,40,45,47,49,50] do
    i=0;
    for x=[vmin:vmax] do
      i=i+1;
      courbe_am(i)=prix_am_n(n,x,N,gain_put);
    end
    plot2d([vmin:vmax],courbe_am);
  end
endfunction
```

On peut faire tendre N vers $+\infty$ de façon à converger vers un modèle continu (le modèle de Black et Scholes en l'occurence).

Dabns le cas européen, le résultat se prouve grace au théorème de la limite centrale. Dans le cas américain, c'est un peu plus compliqué !

```
function main_4(load_flag)
  // Avec cet algorithme on peut augmenter N
  // mais il faut renormaliser convenablement u et d.
  // Essayer avec N=10,100,200,...,1000

  for N=[5,10,20,30,50] do
    d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
    r=r_0/N;
    n=0;
    for x=[vmin:vmax] do
      n=n+1;
      courbe(n)=prix_am(x,N,gain_put);
    end
    plot2d([vmin:vmax],courbe);
    plot2d([vmin:vmax],max(K-[vmin:vmax],0));
  end
endfunction
```

1.1 Simulation de la loi du temps d'arrêt optimal

Pour être efficace on doit calculer un fois pour toute tous les prix sur l'arbre de Cox-Ross.

```
function V=prix_am_vect(x_0,N,gain)
// On calcule les valeurs de "v(n,x)" (voir question précédente)
// mais, ici, on les conservent dans un vecteur "V"
// ce qui évite d'avoir a les re-calculer un grand
// nombre de fois

  // ATTENTION AU DECALAGE DE 1 EN ESPACE ET EN TEMPS
  V=zeros(N+1,N+1);
  x=x_0 * u^[0:N] .* d^[N:-1:0]; // les N+1 points de calcul a l'instant N
  V(N+1,1:N+1)=gain(x);

  for n=[N-1:-1:0] do // le temps décroit de N-1 a 0
    V(n+1,1:n+1) = p * V(n+2,2:n+2) + (1-p) * V(n+2,1:n+1);
    V(n+1,1:n+1) = V(n+1,1:n+1) / (1+r);// actualisation
    x=x_0 * u^[0:n] .* d^[n:-1:0]; // les points de calcul à linstant n
    V(n+1,1:n+1)=max(V(n+1,1:n+1),gain(x));// calcul du max
  end;
endfunction;
```

```
function Bin=simul_bin(N,p)
  // simule des tirages de Bernouilli consécutifs bin
  // et en fait la somme partielle noté Bin.
  // Le trajectoire du processus de Cox Ross se calcule
  //          X(n)=x_0*u^Bin(n)*d^(n-Bin(n))
  bin=grand(1,N,"bin",1,p);// tirages a pile ou face (p,1-p)
  Bin=cumsum(bin);// sommes partielles du vecteur bin
endfunction;

function res=temps_optimal(Bin,V,gain)
  // Calcule le temps d'arrêt optimal
  // de la trajectoire X caracterisée par le vecteur Bin:
  //        X(n)=x_0*u^Bin(n)*d^(n-Bin(n))
  if gain(x_0) == V(1,1)  then  res=0;return;end;
  for n=1:size(Bin,'*')-1 do
    x=x_0*u^Bin(n)*d^(n-Bin(n));// Valeur de X(n)
    if (C'EST QUOI LA CONDITION D'ARRET OPTIMAL and (gain(x) > 0) then
        res=n;return;
    end;
  end;
  res=N;
endfunction

function tau=histo_temps_optimal(x_0,N)
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;
  r_0=0.1;
  r=r_0/N;

  V=prix_am_vect(x_0,N,gain_put);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal(Bin,V,gain_put);
  end
  histo_discret(tau,N,%t);
endfunction;
```

Des exemples de loi de temps d'arrêt.

```
function main_4(load_flag)
  N=100;
  x_0=100;tau=un_test(x_0,N);

  N=50;
  x_0=60;tau=histo_temps_optimal(x_0,N);// on exerce toujours en 0
  x_0=70;tau=histo_temps_optimal(x_0,N);// on n'exerce plus (jamais) en 0
  x_0=80;tau=histo_temps_optimal(x_0,N);// On exerce de plus en plus tard ...
  x_0=100;tau=histo_temps_optimal(x_0,N);
  x_0=120;tau=histo_temps_optimal(x_0,N);// Le plus souvent en N
endfunction
```

On regarde le cas du call européen qui est particulier puisqu'il ne s'exerce jamais avant l'écheance N. Le fait que le prix européen du call soit toujours plus grand que l'obstacle permettrait de le prouver rigoureusement.

```
function main_5(load_flag)
// Teste le cas du call (x-K)_+.

  N=50;

  sigma=0.3;
  r_0=0.1;

  r=r_0/N;
  u=(1+r)*exp(sigma/sqrt(N));
  d=(1+r)*exp(-sigma/sqrt(N));
  p=1/2;  //p= (1+r-d)/(u-d); // en principe pour Cox-Ross

  K=100;
  x_0=100;
  V_call=prix_am_vect(x_0,N,gain_call);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal(Bin,V_call,gain_call);
  end
  histo_discret(tau,N,%t);// le call s'exerce toujours en N
end
```

On calcule par simulation la probabilité d'exercice avant l'échéance qui est (rarement) égal à 1.

```
function proba=compute_proba(x_0,N)
  // Proba que tau<N
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;
  r_0=0.1;
  r=r_0/N;

  V=prix_am_vect(x_0,N,gain_put);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal(Bin,V,gain_put);
  end

  proba=QUE VAUT CETTE PROBA;
endfunction;

function main_6(load_flag)
  N=50;
  x_0=60;compute_proba(x_0,N)
  x_0=70;compute_proba(x_0,N)
  x_0=80;compute_proba(x_0,N)
  x_0=100;compute_proba(x_0,N)
  x_0=120;compute_proba(x_0,N)
endfunction
```

On trace la frontière d'exercice en fonction du temps.

```
function s=frontiere(V)
  // Calcule la frontière d'exercice t -> s(t) t\in[0,N]
  N=size(V);
  N=N(1)-1;
  s=0;
  for n=0:N do
    for j=[0:n] do
      x=x_0*u^j*d^(n-j);
      // printf("n=%d j=%d V=%f obt=%f\n",n,j, V(n+1,j+1), obstacle(x));
      if V(n+1,j+1) > gain_put(x) then
        s(n+1)=x_0*u^(j-1)*d^(n-j+1);break;
      end;
    end
  end;
endfunction

function main_7(load_flag)
  N=1000;
  r_0=0.05;
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  r=r_0/N;

  K=100;x_0=70;
  r_0=0.05;

  V=prix_am_vect(x_0,N,gain_put);
  front=frontiere(V);
  plot2d(front);
endfunction;
```

2 un problème de recrutement

Quelques préliminaires : tirages de ω et R.

```
function [R] = Omega2R(omega)
  // Calcule les rang d'insertion pour un omega donne
  for n=[1:length(omega)] do
    y=gsort(omega(1:n),'g','i');// classe le vecteur omega(1:n) en croissant
    R(n)=find(omega(n)==y);// indice de omega(n) dans le tableau classe
                          // (son classement parmis les n premiers)
  end
  R=R';
endfunction

function [omega] = R2Omega(R)
  // Calcule omega connaissant les rangs d'insertion
  // D'abord "l'inverse de omega"
  for n=[1:length(R)] do omega=[omega(1:R(n)-1),n,omega(R(n):n-1)]; end;
  // On inverse la permutation
  for  n=[1:length(R)] do temp(omega(n))=n;end;
  omega=temp';
endfunction

function main_8()
  // test sur une permutation
  omega=[1 5 4 6 3 2];
  // que vaut R
  R=Omega2R(omega)
  // Retrouve t'on omega
  and(R2Omega(R)==omega)
end
```

On calcule $(u(n,\{0,1\}), 0 \leq n \leq N)$ à l'aide de l'équation de programmation dynamique. On désigne l'"obstacle" par $(f(n,\{0,1\}), 0 \leq n \leq N)$. Notez que $f(n,0) = 0$ et $f(n,1) = n/N$.

```
function u=compute_u(N)
  u=zeros(N,2);
  u(N,:)=[0,1];
  for n=[N-1:-1:1] do
    temp = (n/(n+1))*u(n+1,1) + (1/(n+1))*u(n+1,2);
    // temp >= 0, donc u(n,0)=max(temp,0)=temp
    u(n,:)=[temp,max(temp,n/N)];
  end;
endfunction;
```

Le temps d'arret optimal τ est donné par

$$\tau = \inf\{n \geq 0, u(n, S_n) = f(n, S_n)\}.$$

Il sera toujours postérieur au premier temps où $u(n,1) = f(n,1) = n/N$, puisque $f(n,0) = 0$ et $u(n,0) > 0$ (sauf lorsque $n = N$). $f(n,0)$ et $u(n,0)$ ne peuvent être egaux sauf en $n = N$.

```
epsilon= 0.00001;

function res=temps_echant(N)
  // Calcule le premier temps où u(n,1) = f(n,1) = n/N
  // à ϵ près.
  u=compute_u(N);
  for n=[1:N] do
    if (abs(u(n,2) - (n/N)) < epsilon) then
        break;
    end
  end
  res=n;
endfunction
```

On vérifie par simulation que ce temps (déterministe) "vaut environ" N/e pour N grand.
On peut le démontrer sans trop de difficulté à l'aide de la série harmonique.

```
function main_9()
  // Vérification que temps_echant vaut environ N/e.
  Taille=1000;
  x=0;
  for n=[1:Taille] do
    x(n)=temps_echant(n)/n;
  end
  x=x-1/exp(1);
  nb=size(x,'*');
  plot2d(1:nb,x);
endfunction
```

On évalue la probabilité de succès du temps optimal donné par $u(0,1)$.

```
function main_10()
  N=1000;
  obstacle=[1:N]/N;

  u=compute_u(N);
  plot2d(1:N,obstacle,style=2);
  plot2d(1:N,u(:,2),style=3);// u(n,1)
  plot2d(1:N,u(:,1),style=4);// u(n,0)

  // u(1,2) = P(succés pour la stratégie optimale)
  // On vérifie que cette proba tends vers 1/e = 37\%
  i=0;
  courbe=0;
  for N=[10,25,50,100,250,500,1000] do
    u=compute_u(N);
    i=i+1;courbe(i)=u(1,2);
  end
  plot2d(courbe)
endfunction
```

On vérifie ce calcul par simulation.

```
function n=temps_optimal(omega,u)
// Realise un tirage du temps d'arret optimum

  S=(Omega2R(omega)==1);// On simule S (omega->R->S)
  nb=temps_echant(N);// nb : le premier temps où u(n,1)=f(n,1)

  // Comme f(n,0)=0 et u(n,0) > 0 (sauf pour n=N), le temps
  // optimal se situe après nb et c'est le premier temps ou S(n)=1 apres nb.
  // Sauf si n=N, auquel cas on est oblige de prendre le candidat.
  for n=[nb:N] do
    if CONDITION D'ARRET OPTIMAL then break; end;
  end
  // Si on sort de la boucle avec n=N et N est bien le temps optimal.
endfunction
```

On teste dans un cas particulier.

```
function main_11()
  N=100;
  obstacle=[1:N]/N;

  u=compute_u(N);
  plot2d(1:N,u(:,2),style=3);
  plot2d(1:N,obstacle,style=2);

  // Verification que la probabilité d'obtenir
  // le meilleur est de l'ordre de 1/e
  N=100;
  Taille=1000;
  u=compute_u(N);

  ss=0;
  for i=[1:Taille] do
    omega=grand(1,'prm',[1:N]');
    tirages(i)=temps_optimal(N,omega);

    // sample(i) est il le meilleur ?
    y=gsort(omega(1:N),'g','i');
    ind=find(omega(tirages(i))==y);
    if ind==1 then ss=ss+1;end;
  end
  proba=ss/Taille;

  histo_discret(tirages,N,%t);
end
```

Un problème de gestion de stock

March 14, 2017

Le fichier source en `Scilab` correspondant à ce document est accessible sur

`http://cermics.enpc.fr/~bl/Liesses/controle-scilab.sci`.

Il est partiellement mais pas totalement corrigé. Le correction complète est accessible sur

`http://cermics.enpc.fr/~bl/Liesses/controle-scilab_corrige.sci`.

Vendeur de journaux : le cas de deux indices de temps

On commence par définir les caractéristiques d'une loi discrète par le vecteur $(w_i, 1 \leq i \leq N)$ des valeurs possibles et la probabilité $(p_i, 1 \leq i \leq N)$ de chaque valeur.

```
// une loi discrète
w_i=[30,50,80];
p_i=[1/2,1/4,1/4];
mu=p_i*w_i'; // la moyenne
```

On tire un échantillon de taille N selon cette loi. Pour la simulation on utilise `grand` pour simuler une chaine de Markov dont toutes les lignes sont égales à $(p_i, 1 \leq i \leq N)$. Ce qui fait le travail (exercice sur ... les chaînes de Markov), même si c'est un peu "tordu".

```
N=100000;
P=ones(size(w_i,'*'),1)*p_i;
Y=grand(N,'markov',P,1);// Y prend ces valeur dans 1,2,3
// Les valeurs de W
W=w_i(Y);
```

Définition des paramètres de la fonction coût.

```
c=10,cm=20,cs=5;cf=200;

function y=J(u)
// La fonction J(u) est le cout moyen de  j(u,w) sous la loi p_i
//          j(u,w) = c*u + cs*max(u - w,0) + cm*max(w - u,0)
// Cette espérance se calcule donc par :
  y= c*u + cs*p_i*max((u- w_i'),0) + cm*p_i*max(( w_i'-u),0);
endfunction
```

La fonction suivante calcule le minimum de $J(u)$.

```
function [uopt,m]=minimum(J)
  // calcul de J(u) pour u entre -10 et 100.
  U=-10:100;
  Ju=[];for u=U, Ju=[Ju,J(u)];end

  // recherche du minimum de J pour u entre -10 et 100.
  [m,kmin]=min(Ju);
  uopt=U(kmin);
endfunction
```

Pour tracer la fonction $J(u)$ et matérialiser le minimum remplacer `%f` par `%t` dans ce qui suit.

```
if %f then
  xset('window',1);xbasc();

  // Tracé de J(u)
  U=-10:100;
  Ju=[];for u=U, Ju=[Ju,J(u)];end
  plot2d(U,Ju,style=2);
  xtitle("La fonction J");

  [uopt,m]=minimum(J);

  printf("Nombre optimal de journaux a commander %f\n",uopt);
  printf("Moyenne de la demande %f\n",mu);

  // materialisation du point minimisant
  plot2d(uopt,m,style=-4);
  plot2d3(uopt,m);
  // xclick();// permet d'arrêter le programme
end
```

On regarde maintenant un cas où le vendeur a déjà des journaux et où il paye un coup fixe c_F s'il commande des journaux. On s'attend, en optimisant, à obtenir une une stratégie de type $[s, S]$. On peut le vérifier. On introduit la function $\tilde{J}$.

```
function y=Jtilde(u,x)
  y= cf*(u>0) + J(u+x) -c*x
endfunction
```

On calcule, pour x (entier) variant de 0 à $2\max(w_i)$, la commande optimale de journaux correspondant à chaque valeur de x.

```
xv=0:1:2*max(w_i);// les valeurs de x prises en compte
U=0:2*max(w_i);// les valeurs de U prises en compte
xuopt=[];
for x0=xv
  Ju=[];for u=U, Ju=[Ju,Jtilde(u,x0)];end
  [m,kmin]=min(Ju);
  xuopt=[xuopt,U(kmin)];
end
```

On veut calculer s où s est la valeur maximum à partir de laquelle on passe une commande > 0. `xuopt=0` signifie que l'on commande rien pour le x correspondant. On va chercher toutes les indices correspondant à des valeurs pour lesquelles `xuopt=0`.

```
I=find(xuopt==0);
```

Puis on selectionne la plus petite des valeurs d'indice i.e. `I(1)`, la valeur de x correspondant à

l'indice juste avant (`I(1)-1`) va nous donner s par `s=xv(I(1)-1);`

```
s=xv(I(1)-1);
```

On peut alors vérifier que la stratégie optimale est bien de la forme $[s, S]$.

```
xset('window',1);
xbasc();
plot2d2(xv,xuopt,style=2);
plot2d2(xv(1:s),xv(1:s)+xuopt(1:s),style=1);
xtitle(sprintf("Valeur de s=%d et de S=%d",xv(s),xv(s-1)+xuopt(s-1)));

// Calcul de S le minimiseur de  min J(u) = JS = J(S)
[S,JS]=minimum(J);

// On calcule maitenant s
x=0:2*max(w_i);
Jv=[]; for i=1:size(xv,'*'); Jv=[Jv,J(x(i))];end
I=find(Jv <= cf + JS);
if isempty(I) then s=0; else s=x(I(1)-1); end;

printf("On trouve s=%d et S=%d\n",s,S);

// Il faut verifier que I(i+1)=I(i)+1 pour tout i=1..size(I)-1;
// pour prouver que l'ensemble d'exercice est de la forme [s,S]
// On doit trouver T
and(I(2:$)==(I(1:$-1)+1))
```

Vendeur de journaux : le cas de T pas de temps

On commence par décrire le système dynamique contrôlé en se restreignant à un ensemble borné.

```
// on transforme la dynamique pour rester dans un ensemble
// borné [-100,100]
XMIN=- 100; XMAX = 100;

function y=f(x,u,w)
  y = min(max(x+u-w,XMIN),XMAX)
endfunction
```

La fonction de coût.

```
alpha=1.0;//le coefficient d'actualisation

function y=ct(u,x)
// coût instantané pour t autre que la date de départ
   y = cf*(u>0) + c*u + alpha*cs*max(x,0) + alpha*cm*max(-x,0)
endfunction

function y=c0(u)
// co^ut instantané pour t=0
   y = cf*(u>0) + c*u
endfunction

function y=K(x)
// coût final, noté h(x) dans le texte
   y = cs*max(x,0) + cm*max(-x,0)
endfunction

function res=cout(t,x,u)
// Noté l(n,x,u) dans le texte.
// Vaut c0 pour t=0 et ct sinon
   if t==1 then
      res=c0(u);
   else
      res=ct(u,x);
   end
endfunction

function [cost]=ecost(t,x,u,cout,Vtp1)
// Pour t, u et x fixé calcul de
//        l(t,u,x) + E(v(t+1,f(x,u,W_1))
  cost = 0 ;
  for l=1:size(w_i,'*');
    // boucle sur les aléas pour calculer
    //          E(v(t+1,f(x,u,W_1))
    xtp1= f(x,u,w_i(l));// calcul du nouvel état
    ix = find(xtp1==ensemble_etats);// on retrouve son indice
                                    // dans le tableau des états
    cost = cost + p_i(l) * Vtp1(ix);
  end;
  cost = cout(t,x,u) + cost;  // On rajoute l(t,x,u)
  cost = alpha^t * cost;      // on actualise
endfunction
```

On calcule $v(t,x)$ à l'aide de l'équation de Bellman. On commencera avec $T = 2 : 2$ indices de temps (0 et 1) qui correspond le problème à un pas de temps.

```
T=2; // l'horizon de temps
ensemble_etats= XMIN:XMAX;  // les etats possibles (fini)
ensemble_controles= 0:XMAX; // les commandes de journaux possibles (fini)
```

On commence par diverses initialisations. On affecte $\alpha^T \times K(.)$ à $v(T,.)$. On va calculer $V(t,.)$ pour $T-1,...,1$ et stocker les résultats du calcul de $V(t,.)$ dans une liste `L` et de la commande (feedback) optimale $U(t,.)$ dans `U`.

```
// La fonction v_T(x) = K(x)
VT = alpha^T*K(ensemble_etats);
L=list(VT); // liste qui permet de stocker (V_T,V_{T-1},.....,V_1)
U=list(0); // liste qui permet de stocker (-- ,U_{T-1},.....,U_1)
```

On exécute la boucle rétrograde en temps pour résoudre l'équation de Bellman.

```
for t=T-1:-1:1
  printf("t=%d\n",t);
  Vt = zeros(1,size(ensemble_etats,'*'));
  Ut = %nan*ones(1,size(ensemble_etats,'*'));
  Vtp1= L($);  // Vtp1 est la fonction de Bellman au temps t+1

  // On cherche a calculer ici Vt et Ut
  for i=1:size(ensemble_etats,'*')
    x = ensemble_etats(i);
    mcost= %inf;
    // boucle sur les commandes possibles
    for k = 1:size(ensemble_controles,'*')
      u = ensemble_controles(k);
      // on affecte à cost le coût associé à la commande u.
      cost=ecost(t,x,u,cout,Vtp1);
      // Il faut maintenant comparer cost au meilleur coût (mcost)
      // trouvé jusque là (on minimise) et mettre à jour mcost
      // et kumin (l'indice de la commande qui donne le meilleur coût)
      if cost < mcost then
        mcost = cost;
        kumin = k;
      end
    end
    // On stocke pour l'état x d'indice i le coût optimal
    Vt(i) =  mcost;
    // et la commande optimale
    Ut(i) = ensemble_controles(kumin);
  end
  // On range Vt et Ut a la fin(=$) de la liste
  L($+1)=Vt;
  U($+1)=Ut;
end
```

Dessin de la fonction de Bellman et de la commande optimale au temps $t = 1$.

```
xset('window',1);
xbasc();
plot2d2(ensemble_etats,L($));// la fonction de Bellman au temps t=1;

xset('window',1);
xbasc();
plot2d2(ensemble_etats,U($));// la commande optimale au temps t=1;
```