Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 1. — FONCTIONS ITÉRÉES STOCHASTIQUES

## Préambule

Ces notes de travaux pratiques reprennent presque à l'identique celle du polycopié « Ingénierie Stochastique » de Pierre Del Moral [1]. Leur auteur fait, avec Bernard Ycart, depuis de longues années autorité dans le monde probabiliste français en ce qui concerne les « algorithmes stochastiques », et ce, aussi bien dans le cadre de la Recherche que de l'Enseignement. Ne pas les utiliser aurait été un péché de vanité déplacé. Nous transcrivons en *italique* les passages faisant une référence directe à ce polycopié, ainsi que les paragraphes où la voix de l'auteur se fait très clairement sentir et résonne avec son enseignement.

*L'objectif principal de cette série de travaux pratiques est d'explorer l'ensemble des techniques de simulation, de modélisation markovienne, et d'analyse probabiliste étudiées dans ce cours sur une série d'exemples concrets.*

*Nous recommendons au lecteur d'adopter une double réflexion. Tout d'abord, il conviendra de s'assurer, du moins intuitivement, de la stabilité du processus aléatoire étudié, et des fondements probabilistes du résultat recherché. Il est aussi recommandé de mener une réflexion sur les qualités numériques et physiques des algorithmes. On s'interrogera notamment sur leurs performances, et leurs limitations pratiques.*

## 1. Partie théorique

*Cette première série de travaux pratiques se focalise sur les fonctions itérées stochastiques, et la simulation d'images fractales. Ces algorithmes stochastiques sont fondés sur l'exploration d'une surface plane par une séquence de contractions affines. Le seul aléa réside simplement dans le choix à chaque étape de l'une de ces transformations.*

Exercice 1 (*théorique*). — Soit $E = \mathbb{R}^2$, ou $\mathbb{R}^n$, muni de sa topologie usuelle.

(i) Nous considérons une matrice $A$ de norme strictement inférieure à 1 et un vecteur $b \in E$. Montrer que pour tout $x \in E$, la suite $(x_n)_{n\geqslant 0}$ définie par $x_0 = x$, $x_{n+1} = S \cdot x_n = A \cdot x_n + b$, $n \geqslant 0$, est convergente. Constater que la limite ne dépend pas des premiers termes de la suite. Quelle est cette limite ?

(ii) Supposons maintenant que $x_{n+1} = S_n \cdot x_n = A_n \cdot x_n + b_n$, où $(A_n)_{n\geqslant 0}$ est une suite de matrices bornée par $0 \leqslant r < 1$ et $(b_n)_{n\geqslant 0}$ une suite de vecteurs bornée par $s \geqslant 0$. Montrer que la suite $(x_n)_{n\geqslant 0}$ est bornée. Est-elle nécessairement convergente ? (Considérer par exemple $A_n = 0$ et $b_n = (-1)^n b$ pour tout $n \geqslant 0$.) Montrer que deux suites $(x_n)_{n\geqslant 0}$ et $(x'_n)_{n\geqslant 0}$ ainsi construites et associées respectivement à des valeurs initiales $x$ et $x'$ sont asymptotiquement égales.

(iii) On rappelle qu'une valeur d'adhérence, ou point d'accumulation, d'une suite $(x_n)_{n\geqslant 0}$ est un point $a \in E$ tel que pour tout voisinage $V$ de $a$, ou toute boule $V = B(a, \varepsilon)$, le nombre d'indices $n \geqslant 0$ tels que $x_n \in V$ est infini.

Dans le cadre de la question précédente, montrer que l'ensemble des valeurs d'adhérence d'une telle suite est non vide et ne dépend que de la suite des transformations affines choisies et non de la valeur initiale.

*Ces modèles sont sans nul doute les chaînes de Markov les plus simples à simuler numériquement. Ils sont simplement définis par donnée d'une famille de transformations affines du plan $(S_i)_{i\in I}$ indexée par un ensemble fini $I$, muni d'une mesure de probabilité $\mu$. Pour les décrire, on introduit une suite $(\varepsilon_n)_{n\geqslant 1}$ de variables aléatoires indépendantes et de même loi $\mu$ sur $I$. Ces algorithmes stochastiques sont représentés par la donnée d'une chaîne de Markov $X = (X_n)_{n\geqslant 0}$ définie récursivement par les équations suivantes :*

$$X_0 \in \mathbb{R}^2, \qquad X_n = S_{\varepsilon_n} \cdot X_{n-1} = S_{\varepsilon_n} \circ S_{\varepsilon_{n-1}} \cdot X_{n-2} = \cdots = S_{\varepsilon_n} \circ \cdots \circ S_{\varepsilon_1} \cdot X_0.$$

Exercice 2 (*théorique*). — Se rappeler pourquoi le processus ainsi défini est une chaîne de Markov dans la filtration engendrée par $X_0$ et par $(\varepsilon_n)_{n\geqslant 1}$, ainsi que dans sa propre filtration naturelle. Noter que l'espace d'états est $E = \mathbb{R}^2$ qui n'est ni fini, ni dénombrable.

## 2. Partie pratique

### 2.1. Modèle fractal de feuille

Dans ce premier modèle, on se donne une suite de variables aléatoires indépendantes $(\varepsilon_n)_{n\geqslant 1}$ de loi de Bernoulli de paramètre $p = 0{,}2993$, c'est-à-dire

$$\mathbb{P}\{\varepsilon_n = 1\} = 1 - \mathbb{P}\{\varepsilon_n = 0\} = 0{,}2993.$$

Pour chaque $i = 0$, 1, on choisit des fonctions affines $S_i \cdot x = A_i \cdot x + b_i$ avec les matrices

$$A_0 = \begin{pmatrix} +0{,}4000 & -0{,}3733 \\ +0{,}0600 & +0{,}6000 \end{pmatrix}, \qquad b_0 = \begin{pmatrix} +0{,}3533 \\ +0{,}0000 \end{pmatrix}$$

et

$$A_1 = \begin{pmatrix} -0{,}8000 & -0{,}1867 \\ +0{,}1371 & +0{,}8000 \end{pmatrix}, \qquad b_1 = \begin{pmatrix} +1{,}1000 \\ +0{,}1000 \end{pmatrix}.$$

Exercice 3 (*pratique*). — (i) On souhaite saisir dans un programme Scilab la donnée des matrices et des vecteurs. Si la définition des vecteurs ne nécessite que deux indices (numéro de vecteur, numéro de ligne), celle des matrices en nécessite trois. Puisque la dimension de l'espace $E = \mathbb{R}^2$ est suffisamment petite pour effectuer les calculs matriciels directement avec les coefficients, on pourra écrire

```
// Mod\'ele fractal de feuille

A = [0.4, -0.3733, 0.06, 0.6; -0.8, -0.1867, 0.1371, 0.8];
b = [0.3533, 0; 1.1, 0.1];
```

le premier indice désignant le numéro de matrice ou de vecteur, le second, le coefficient correspondant, *voire* tout stocker dans une matrice unique à 6 colonnes. (En fait, on peut définir `A = zeros(2, 2, 2); b = zeros(2, 2)`, et faire les calculs « normalement » ou presque.)

(ii) Le calcul, ou la simulation, des états successifs de la chaîne ne pose aucune difficulté, hormis le passage de l'indice 0 au numéro de ligne $1 = 0 + 1$, etc.

```
p = 0.2993; n = 10000; x = zeros(n+1, 2);
x(1, :) = [0, 0];// valeur ou \'etat initial
for i = 2:n+1
    if rand() < 1-p then r = 1; else r = 2; end
```

```
    x(i, 1) = A(r, 1)*x(i-1, 1)+A(r, 2)*x(i-1, 2)+b(r, 1);
    x(i, 2) = A(r, 3)*x(i-1, 1)+A(r, 4)*x(i-1, 2)+b(r, 2);
end
```

(iii) On représente graphiquement la trajectoire obtenue en se souvenant que si le premier argument est un vecteur colonne, c'est alors le vecteur des abscisses et après vient la colonne, ou les colonnes sous forme d'une matrice, d'ordonnées, et que s'il n'y a pas de premier vecteur colonne, l'indice sert d'abscisse.

```
clf(); plot(x(:, 1), x(:, 2), "o-");// ‘‘plot’’ \‘a la matlab
```

La combinaison d'options `"o-"` marque les points d'un rond et les joint d'un trait plein. On peut aussi essayer `"o--"` ou `"--"`, ou encore `"o"`.

(iv) Modifier la valeur initiale et constater qu'elle n'affecte qu'assez peu la suite (pour une même graine donnée `rand("seed")`). Pour des commodités de visualisation, choisir une valeur initiale proche du nuage de points obtenu.

(v) Augmenter le nombre de pas progressivement. Constater que joindre les points ne permet que d'avoir une petite information sur le début de la trajectoire et qu'ensuite cela nuit à l'identification des régions du plan dans lesquelles la suite s'accumule. Pousser le nombre de pas assez loin pour justifier le titre de cette sous-section.

Exercice 4 (*théorique*). — D'après l'exercice précédent, on sait que l'ensemble des points d'accumulation observé ne dépend que de la suite des transformations affines, et encore, qu'asymtotiquement, c'est-à-dire ici de la suite $(\varepsilon_n)_{n\geqslant 1}$. Sans rien démontrer, quels types de résultats probabilistes permettraient de justifier que presque sûrement les trajectoires de la chaîne ont même ensemble de points d'accumulation, ensemble qui est l'ensemble fractal cherché.

### 2.2. Modèle fractal d'arbre

Dans ce second modèle, on se donne une suite de variables aléatoires indépendantes de Bernoulli uniformes sur $I = \{1, 2, 3\}$. Pour chaque $i = 1$, 2, 3, on choisit des fonctions affines $S_i \cdot x = A_i \cdot x + b_i$ avec les matrices

$$A_1 = \begin{pmatrix} 0 & 0 \\ 0 & c \end{pmatrix}, \qquad b_1 = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix},$$

$$A_2 = \begin{pmatrix} r\cos(\varphi) & -r\sin(\varphi) \\ r\sin(\varphi) & r\cos(\varphi) \end{pmatrix}, \qquad b_2 = \begin{pmatrix} 1/2 - r/2\cos(\varphi) \\ c - r/2\sin(\varphi) \end{pmatrix},$$

et

$$A_3 = \begin{pmatrix} q\cos(\psi) & -r\sin(\psi) \\ q\sin(\psi) & r\cos(\psi) \end{pmatrix}, \qquad b_3 = \begin{pmatrix} 1/2 - q/2\cos(\psi) \\ 3c/5 - q/2\sin(\psi) \end{pmatrix}.$$

Les paramètres précédents sont définis par

$$c = 0{,}255, \qquad r = 0{,}75, \qquad q = 0{,}625, \qquad \varphi = -\pi/8, \qquad \psi = \pi/5, \qquad |X_0| \leqslant 1.$$

Exercice 5 (*pratique*). — Ici, le choix de la transformation affine se fait de manière uniforme sur $I$. Définir une fonction, ou procédure, automatisant la simulation. La mettre en œuvre avec l'« arbre » proposé.

### 2.3. Triangle de Sierpinski

On se donne une suite de variables aléatoires indépendantes de loi uniforme sur $I = \{1, 2, 3\}$. Pour chaque $i = 1, 2, 3$, on choisit des fonctions affines $S_i \cdot x = A_i \cdot x + b_i$ avec la matrice

$$A_1 = A_2 = A_3 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$$

et les vecteurs

$$b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad b_2 = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}, \quad \text{et} \quad b_3 = \begin{pmatrix} 1/4 \\ \sqrt{3}/4 \end{pmatrix} \approx \begin{pmatrix} 0{,}250 \\ 0{,}433 \end{pmatrix}.$$

Exercice 6 (*pratique*). — Mettre en œuvre la procédure de simulation sur cet exemple. La structure étant très régulière, on peut plutôt imaginer approximativement énumérer les points de l'ensemble limite en partant de $x_0 = (0, 0)$, et en calculant les points suivants de manière récursive. Le faire.

### 2.4. Le Dragon de Heighways

On se donne une suite de variables aléatoires indépendantes de Bernoulli uniformes sur $I = \{1, 2\}$. Pour chaque $i = 1, 2$, on choisit des fonctions affines $S_i \cdot x = A_i \cdot x + b_i$ avec les matrices

$$A_1 = \begin{pmatrix} 1/2 & -1/2 \\ 1/2 & 1/2 \end{pmatrix} \qquad \text{et} \qquad A_2 = \begin{pmatrix} -1/2 & -1/2 \\ 1/2 & -1/2 \end{pmatrix},$$

et les vecteurs

$$b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad b_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Exercice 7 (*pratique*). — Mettre en œuvre la procédure de simulation et la procédure d'énumération. L'image obtenue justifie-t-elle la dénomination de « dragon » ?

# TRAVAUX PRATIQUES N° 1. — FONCTION ITÉRÉES STOCHASTIQUES, ÉLÉMENTS D'EXPLICATIONS

## 1. **Partie théorique**

Exercice 1. — (i) Pour $n \geqslant 0$, on a par l'identité géométrique

$$x_{m+n} = S^n \cdot x_m = A^n \cdot x_m + \sum_{k=0}^{n-1} A^k \cdot b = A^n \cdot x_m + \frac{\mathrm{Id}_E - A^n}{\mathrm{Id}_E - A} \cdot b \longrightarrow (\mathrm{Id}_E - A)^{-1} \cdot b$$

quand $n$ tend vers l'infini, la convergence de la série de matrices étant assuré par sa nature géométrique et le fait que l'espace vectoriel normé des endomorphismes de $E$ est un espace de Banach.

(ii) Pour $n \geqslant 0$, on a

$$\|x_{n+1}\| = \|A_n \cdot x_n + b_n\| \leqslant \|A_n\| \times \|x_n\| + \|b_n\| \leqslant r \times \|x_n\| + s \leqslant \cdots$$
$$\cdots \leqslant r^{n+1} \times \|x_0\| + \sum_{k=0}^{n} r^k \times s \leqslant r^{n+1} \times \|x_0\| + \sum_{k=0}^{\infty} r^k \times s = r^{n+1} \times \|x_0\| + \frac{s}{1-r}$$

suite de majorants qui converge vers $s/(1-r)$ et qui est donc bornée.

L'exemple donné dans l'énoncé montre qu'on peut facilement construire de telles suites qui ne sont pas convergentes. Cet exemple est assez extrême puisque la condition initiale est oubliée dès le premier pas, et que les termes suivants sont « indépendants ».

Si $x$ et $x'$ sont deux points de $E$, on a

$$x_{n+1} - x'_{n+1} = A_n \cdot (x_n - x'_n) = \cdots = A_n \circ \cdots \circ A_0(x - x'),$$

et ainsi $\|x_n - x'_n\| \leqslant r^n \times \|x - x'\|$ qui tend vers 0 quand $n$ tend vers l'infini.

(iii) La suite $(x_n)_{n\geqslant 0}$ étant bornée, elle est incluse dans un compact ($E$ est de dimension finie). Par conséquent, elle admet des valeurs d'adhérence. Si $(x'_n)_{n\geqslant 0}$ est une suite similaire, on constate que toute valeur d'adhérence de $(x_n)_{n\geqslant 0}$ est aussi valeur d'adhérence de $(x'_n)_{n\geqslant 0}$, et réciproquement, puisque les deux suites sont asymptotiquement égales. □

Exercice 2. — Il s'agit d'un exercice traité en 2m12...

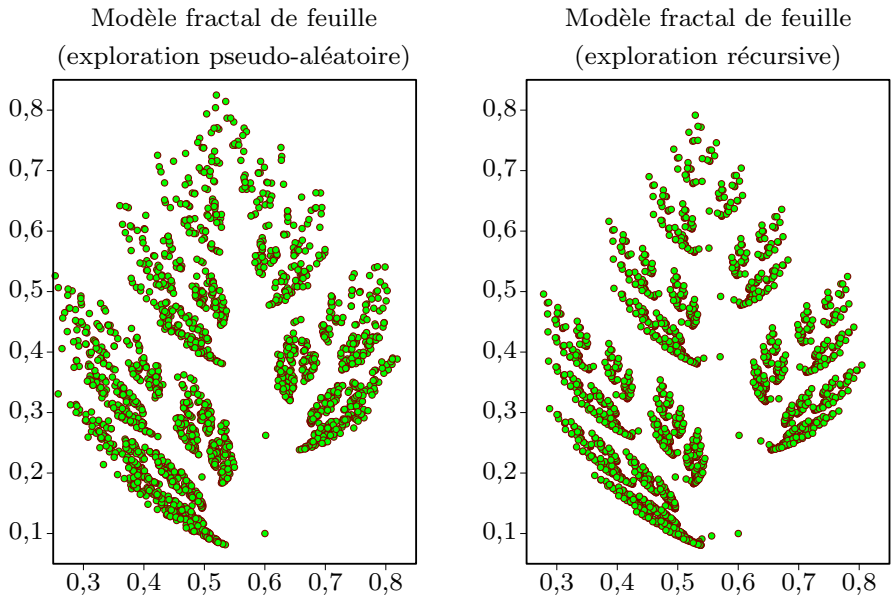
## 2. **Partie pratique**

### 2.1. Modèle fractal de feuille

Exercice 3. — Le code proposé ci-dessous est une alternative peut-être plus naturelle à celui donné dans l'énoncé :

```
clear(); mode(0); n = 1000;

// Mod\‘ele fractal de feuille
```

```
A = zeros(2, 2, 2);
A(:, :, 1) = [0.4, -0.3733; 0.06, 0.6];
A(:, :, 2) = [-0.8, -0.1867; 0.1371, 0.8];

b = zeros(2, 2);
b(:, 1) = [0.3533; 0];
b(:, 2) = [1.1; 0.1];

x = zeros(2, n); x(:, 1) = [0.6; 0.1]; p = 0.5;

for i = 1:n-1
    if grand(1, 1, "def") < 1-p then u = 1; else u = 2; end
    x(:, i+1) = A(:, :, u)*x(:, i)+b(:, u);
end

scf(1); clf();
xtitle("Modele fractal de feuille");
plot2d(x(1, :), x(2, :), 0, rect=[0, 0, 1.2, 0.9]);
```




Exercice 4. — On peut argumenter par le lemme de Borel–Cantelli et son application au « singe dactylographe ». Ça ne serait pas très juste. C'est bien une loi du 0–1 qui est à l'œuvre, mais c'est un peu compliqué...

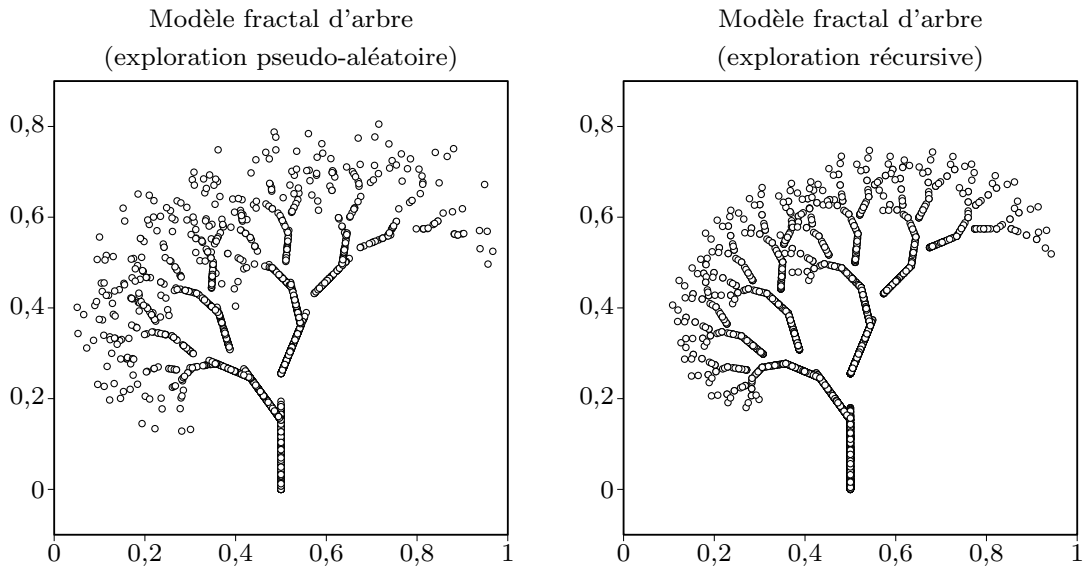
## 2.2. Modèle fractal d'arbre

Exercice 5. — Un code possible :

```
// Mod\'ele fractal d'arbre

k = 3;
c = 0.255; r = 0.75; q = 0.625; phi = -%pi/8; psi = %pi/5;

A = zeros(2, 2, k);
A(:, :, 1) = [0, 0; 0, c];
A(:, :, 2) = [r*cos(phi), -r*sin(phi); r*sin(phi), r*cos(phi)];
A(:, :, 3) = [q*cos(psi), -r*sin(psi); q*sin(psi), r*cos(psi)];
```

```
b = zeros(2, k);
b(:, 1) = [1/2; 0];
b(:, 2) = [1/2-r/2*cos(phi); c-r/2*sin(phi)];
b(:, 3) = [1/2-q/2*cos(psi); 3*c/5-q/2*sin(psi)];

x = zeros(2, n); x(:, 1) = [0.6; 0.1];

for i = 1:n-1
    u = floor(1+k*rand(1, 1, "def"));
    x(:, i+1) = A(:, :, u)*x(:, i)+b(:, u);
end

scf(2); clf();
xtitle("Modele fractal d''arbre");
plot2d(x(1, :), x(2, :), 0, rect = [0, 0, 1, 1])
```

L'automatisation demandée aura été faite en MetaPost.



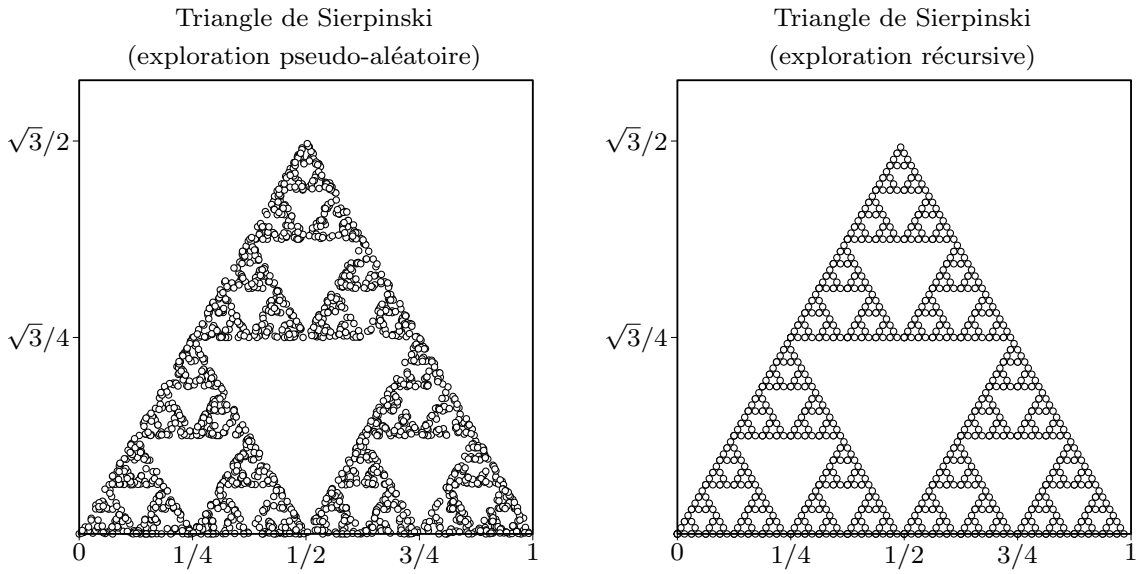

### 2.3. Triangle de Sierpinski

Exercice 6. — Voici du code :

```
// Triangle de Sierpinski

k = 3;

A = diag([1/2, 1/2]);

b = zeros(2, k);
b(:, 1) = [0; 0];
b(:, 2) = [1/2; 0];
b(:, 3) = [1/4; sqrt(3)/4];

x = zeros(2, n);

for i = 1:n-1
    x(:, i+1) = A*x(:, i)+b(:, floor(1+k*rand(1, 1, "def")));
end
```

```
scf(3); clf();
xtitle("Triangle de Sierpinski");
plot2d(x(1, :), x(2, :), 0, rect = [-0.1, 0, 1, 0.9]);
```

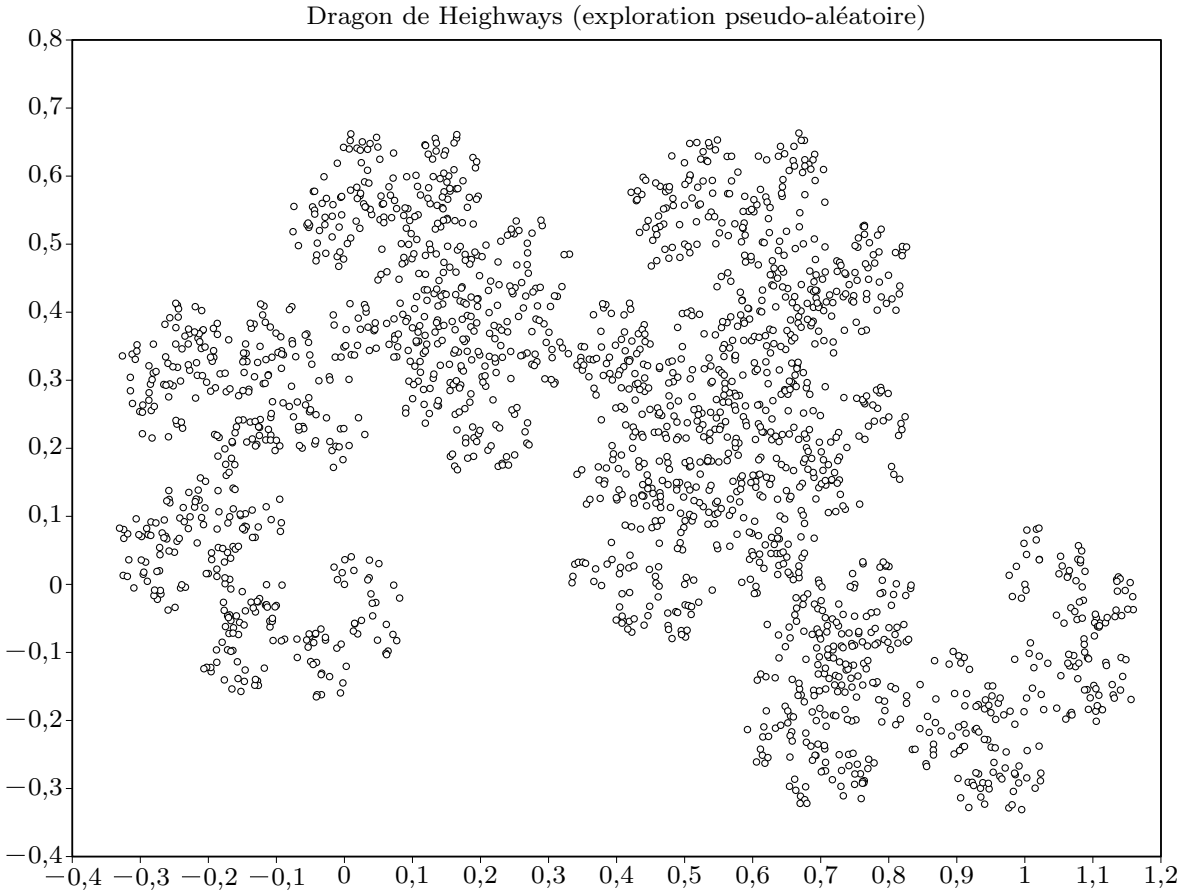
L'« énumération » récursive demandée aura été faite en MetaPost.

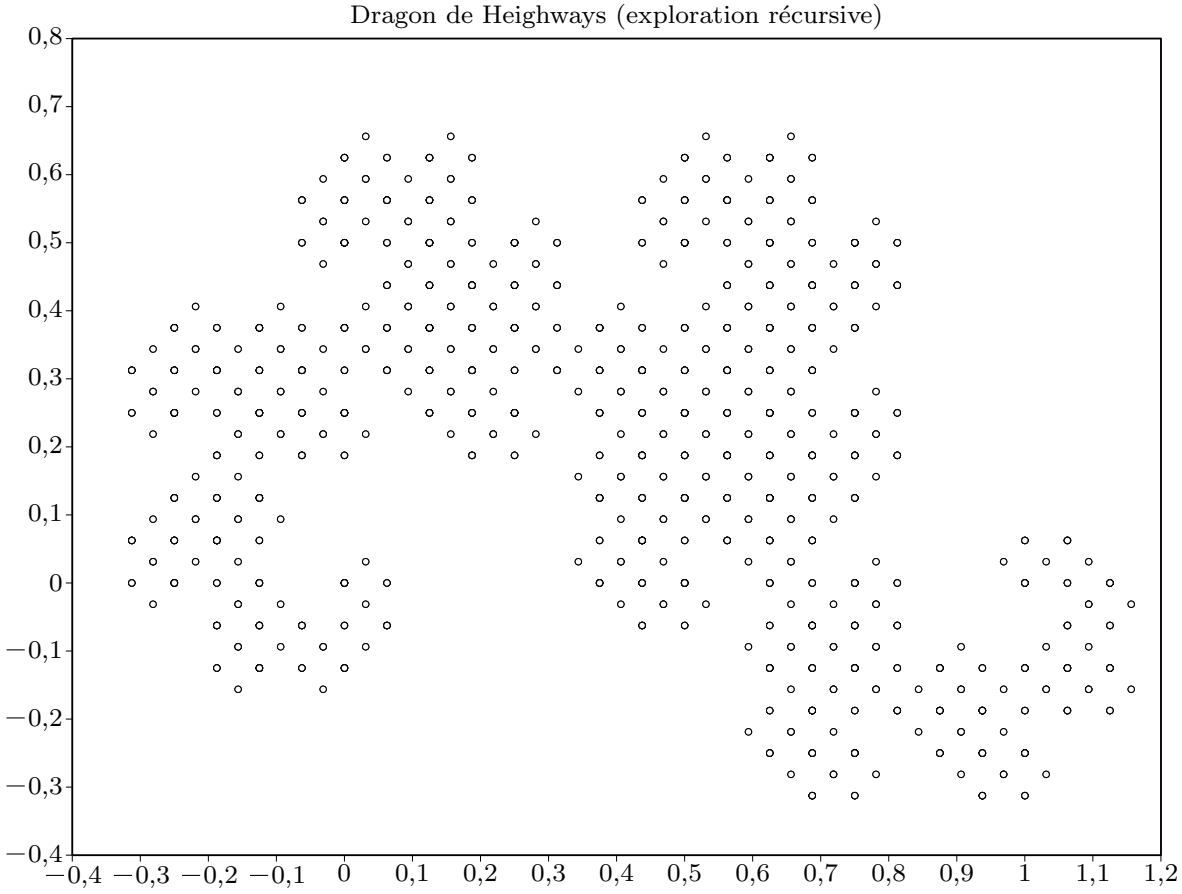



## 2.4. Le dragon de Heighways

Exercice 7. — Pour finir, du code :

```
// Dragon de Heighways

k = 2;

A = zeros(2, 2, k);
A(:, :, 1) = [1/2, -1/2; 1/2, 1/2];
A(:, :, 2) = [-1/2, -1/2; 1/2, -1/2];

b = zeros(2, k);
b(:, 1) = [0; 0];
b(:, 2) = [1; 0];

x = zeros(2, n);

for i = 1:n-1
    u = floor(1+k*rand(1, 1, "def"));
    x(:, i+1) = A(:, :, u)*x(:, i)+b(:, u);
end

scf(4); clf();
xtitle("Dragon de Heighways");
plot2d(x(1, :), x(2, :), 0, rect = [-0.4, -0.4, 1.2, 0.8]);
```


Dragon de Heighways (exploration pseudo-aléatoire)
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
−0,1
−0,2
−0,3
−0,4
−0,4 −0,3 −0,2 −0,1 0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2


Dragon de Heighways (exploration récursive)
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
−0,1
−0,2
−0,3
−0,4
−0,4 −0,3 −0,2 −0,1 0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2

**Code MetaPost**

Tout d'abord nous chargeons quelques fichiers de macros personnelles, modifions quelques paramètres visuels dans un programme MetaPost (détails omis). Ensuite, nous pouvons effectuer avec MetaPost les activités de travaux pratiques.

```
% Il aurait \'et\'e plus \'el\'egant de d\'efinir les transformations
% \`a l'aide des variables de type ``transform'' de MetaFont/Post...
% Bon, ce qui est fait est fait.

data(feuille)
0.4, -0.3733, 0.06, 0.6, 0.3533, 0,
-0.8, -0.1867, 0.1371, 0.8, 1.1, 0.1;

c := 0.255; r := 0.75; q := 0.625; phi := -180/8; psi := 180/5;

data(arbre)
0, 0, 0, 0.255, 0.5, 0,
r*cosd(phi), -r*sind(phi), r*sind(phi), r*cosd(phi),
1/2-r/2*cosd(phi), c-r/2*sind(phi),
q*cosd(psi), -r*sind(psi), q*sind(psi), r*cosd(psi),
1/2-q/2*cosd(psi), 3*c/5-q/2*sind(psi);

data(sierpinski)
0.5, 0, 0, 0.5, 0, 0,
0.5, 0, 0, 0.5, 0.5, 0,
0.5, 0, 0, 0.5, 0.25, sqrt(3)/4;

data(heighways)
0.5, -0.5, 0.5, 0.5, 0, 0,
-0.5, -0.5, 0.5, -0.5, 1, 0;

% La macro suivante explore toutes les trajectoires possibles
% des suites de mani\`ere r\'ecursive.

def fractalset(expr x, y, depth) =
    gdot(x, y);
    if depth > 0:
        for i = 0 upto A.n/6-1:
            fractalset(A[i*6+1]*x+A[i*6+2]*y+A[i*6+5],
            A[i*6+3]*x+A[i*6+4]*y+A[i*6+6], depth-1);
        endfor
    fi
enddef;

% Cette macro d\'efinit les coordonn\'ees d'une suite pseudo-al\'eatoire.
% Elle ne trace rien du tout. Il faut le faire par soi-m\^eme ensuite.
% Utiliser epsilon comme incr\'ement assure de pouvoir consid\'erer un
% tr\` es grand nombre de points...

def fractalpath(expr xinit, yinit, nepsilon) =
%    declare(x, nmax);
    x.n := nepsilon;
    x[0] := xinit; y[0] := yinit;
    for j = epsilon step epsilon until nepsilon:
        i:= floor(uniformdeviate(A.n/6));
```

```
        x[j] := A[i*6+1]*x[j-epsilon]+A[i*6+2]*y[j-epsilon]+A[i*6+5];
        y[j] := A[i*6+3]*x[j-epsilon]+A[i*6+4]*y[j-epsilon]+A[i*6+6];
    endfor;
%    lastpath := (x[0], y[0]) for j=epsilon step epsilon until
%    nepsilon: --(x[j], y[j]) endfor;
enddef;

for flag = true, false:
    beginfig(thisfig);
        setrange(0.25, 0.85, 4cm, 0.05, 0.85, 8cm*2/3);
        pickup rule.nib;
        usedata(feuille, A);
        graphcolor := 0.5red; background := green;
        if flag:
            fractalpath(0.6, 0.1, 1/32);
            for i = 0 step epsilon until x.n: gdot(x[i], y[i]); endfor
            gtitle.top(btex \multline{Mod\‘ele fractal de feuille
            \\(exploration pseudo-al\’eatoire)} etex, 0);
        else:
            fractalset(0.6, 0.1, 10);
            gtitle.top(btex \multline{Mod\‘ele fractal de feuille
            \\(exploration r\’ecursive)} etex, 0);
        fi
        graphcolor := black; background := white;
        autograd;
    endfig;
endfor

for flag = true, false:
    beginfig(thisfig);
        setrange(0, 1, 5cm, -0.1, 0.9, 5cm);
        pickup rule.nib;
        usedata(arbre, A);
        if flag:
            fractalpath(0.5, 0, 1/32);
            for i = 0 step epsilon until x.n: gdot(x[i], y[i]); endfor
            gtitle.top(btex \multline{Mod\‘ele fractal d’arbre
            \\(exploration pseudo-al\’eatoire)} etex, 0);
        else:
            fractalset(0.5, 0, 7);
            gtitle.top(btex \multline{Mod\‘ele fractal d’arbre
            \\(exploration r\’ecursive)} etex, 0);
        fi
        autograd;
    endfig;
endfor

for flag = true, false:
    beginfig(thisfig);
        setrange(0, 1, 5cm, 0, 1, 5cm);
        pickup rule.nib;
        usedata(sierpinski, A);
```

```
        if flag:
            fractalpath(0, 0, 1/32);
            for i = 0 step epsilon until x.n: gdot(x[i], y[i]); endfor
            gtitle.top(btex \multline{Triangle de Sierpinski
            \\(exploration pseudo-al\'eatoire)} etex, 0);
        else:
            fractalset(0, 0, 6);
            gtitle.top(btex \multline{Triangle de Sierpinski
            \\(exploration r\'ecursive)} etex, 0);
        fi
        xticks.bot(vmin)0, "1/4", "1/2", "3/4", 1;
        ticklabel.lft(btex $\sqrt 3/4$ etex, gcoord(hmin, sqrt(3)/4));
        ticklabel.lft(btex $\sqrt 3/2$ etex, gcoord(hmin, sqrt(3)/2));
    %    yticks.lft(vmin)0, "sqrt(3)/4", "sqrt(3)/2", 1;
    endfig;
endfor

for flag = true, false:
    beginfig(thisfig);
        setrange(-0.4, 1.2, 12cm, -0.4, 0.8, 9cm);
        pickup rule.nib;
        usedata(heighways, A);
        if flag:
            fractalpath(0, 0, 1/32);
            for i = 0 step epsilon until x.n: gdot(x[i], y[i]); endfor
            gtitle.top(btex Dragon de Heighways
            (exploration pseudo-al\'eatoire) etex, 0);
        else:
            fractalset(0, 0, 10);
            gtitle.top(btex Dragon de Heighways
            (exploration r\'ecursive) etex, 0);
        fi
        autograd;
    endfig;
endfor
```

Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 2. — ALGORITHME DE ROBBINS–MONRO

Cette section reprend — à quelques notations et corrections près — les notes de cours [1].

## 1. Introduction

Cet algorithme aléatoire permet d'étudier les lignes de niveau

$$F_\alpha = \{x \in E : F(x) = \alpha\}, \qquad \alpha \in \mathbb{R}^d,$$

d'une fonction $F : E \to \mathbb{R}^d$ définie sur un espace d'états donné $E$.

*Exemples. — a) Potentiel gradient convexe.* On considère le gradient

$$F(x,y) = \operatorname{grad} V(x,y) = \nabla V(x,y) = \begin{pmatrix} \dfrac{\partial V}{\partial x}(x,y) \\ \dfrac{\partial V}{\partial y}(x,y) \end{pmatrix}, \qquad (x,y) \in \mathbb{R}^2,$$

d'une fonction $V : \mathbb{R}^2 \to \mathbb{R}$, le potentiel, de classe $C^1$ et strictement convexe. Dans ce cas, l'ensemble $F_0 = \{(x,y) \in \mathbb{R}^2 : \nabla F(x,y) = 0\}$ se résume à l'unique minimum $(x_0, y_0)$ de $V$.

*b) Fonction de répartition.* Soit $F$ la fonction de répartition d'une mesure de probabilité $\mu$ sur $\mathbb{R}$, ou de la loi d'une variable aléatoire réelle $Y : (\Omega, \mathcal{A}, \mathbb{P}) \to \mathbb{R}$. Ici $E = \mathbb{R}$ et

$$F : x \in \mathbb{R} \longmapsto F(x) = \mu(]-\infty, x]) = \mathbb{P}\{Y \leqslant x\} = \mathbb{E}\big[\mathbb{1}_{\{Y \leqslant x\}}\big].$$

Dans ce contexte, l'ensemble $F_{1/2} = \{x \in \mathbb{R} : F(x) = 1/2\}$ peut être vide, réduit à un point, ou consister en un intervalle $[a, b[$. Lorsque $F_{1/2}$ est réduit à un point, celui-ci est la médiane $x_{1/2}$, ou $q_{1/2}$, de la distribution de $Y$ ; lorsqu'il est vide, c'est que $F$ évite par un saut la valeur $1/2$, dans ce cas la médiane $x_{1/2}$ est le point où a lieu ce saut ; enfin, lorsque $F_{1/2} = [a, b[$, un bon choix est de poser $x_{1/2} = \alpha$ mais certains conviennent de prendre $x_{1/2} = (a+b)/2$.

## 2. Description du modèle

Supposons désormais qu'il existe un et un seul point $x_\alpha \in E = \mathbb{R}^d$ tel que $F(x_\alpha) = \alpha$, et que pour chaque $x \neq \alpha$, on a

$$\big\langle x - x_\alpha,\, F(x) - F(x_\alpha) \big\rangle > 0. \tag{1}$$

Lorsque $d = 1$, l'inégalité 1 indique que les nombres $x - x_\alpha$ et $F(x) - F(x_\alpha)$ sont de même signe. Dans cette situation la relation 1 se traduit par l'une des équivalences suivantes

$$\begin{aligned} x < x_\alpha &\iff F(x) < F(x_\alpha) \\ x > x_\alpha &\iff F(x) > F(x_\alpha). \end{aligned}$$

Algorithme. — *Un algorithme déterministe naturel de recherche du point $x_\alpha \in \mathbb{R}^d$ est défini en posant*

$$X_{n+1} - X_n = \gamma_n \times \big(F(x_\alpha) - F(X_n)\big) = \gamma_n \times \big(\alpha - F(X_n)\big).$$

*On choisit $X_0 \in \mathbb{R}^d$, et on utilise une suite de pas strictement positifs $\gamma_n \downarrow 0$ afin de stopper l'évolution de la suite $(X_n)_{n\geqslant 0}$. Les schémas de décroissance de $(\gamma_n)_{n\geqslant 0}$ pour lesquels $\lim_{n\to\infty} X_n = x_\alpha$ doivent satisfaire les deux conditions suivantes :*

$$\sum_{n\geqslant 0} \gamma_n = \infty \qquad et \qquad \sum_{n\geqslant 0} \gamma_n^2 < \infty.$$

L'introduction de l'aléatoire dans les algorithmes déterministes précédents ne se justifie que dans les deux situations qui suivront. Dans les deux cas, le choix de schémas de décroissance de $(\gamma_n)_{n\geqslant 0}$ vérifiant les conditions précédentes entraîne que $\lim_{n\to\infty} X_n = x_\alpha$.

## 3. **Applications**

*Évaluation indirecte bruité.* — L'évaluation de la fonction $F : \mathbb{R}^d \to \mathbb{R}^d$ ne peut être faite directement sinon par des mesures approchées. Plus précisément, chaque tentative d'évaluation de $F(x)$ en un point $x \in \mathbb{R}^d$ donne une valeur approximative

$$x \longrightarrow \boxed{\text{capteur/mesure/évaluation}} \longrightarrow F(x) + \varepsilon$$

où $\varepsilon$ représente une perturbation aléatoire à valeurs dans $\mathbb{R}^d$. La nature statistique de ces perturbations de mesures dépendent de l'instrument de mesure utilisé.

Principe. — *Pour construire l'algorithme de Robbins–Monro, on remplace simplement à chaque étape $n$ la valeur inconnue $F(X_n)$ par son observation bruité*

$$Y_{n+1} := F(X_n) + \varepsilon_{n+1}.$$

*La perturbation $\varepsilon_n$ désigne ici une variable aléatoire à valeurs dans $\mathbb{R}^d$ de loi donnée. Cela conduit à l'algorithme stochastique*

$$X_{n+1} - X_n = \gamma_n \times \big(F(x_\alpha) - F(X_n)\big) - \gamma_n \times \varepsilon_{n+1} = \gamma_n \times \big(F(x_\alpha) - Y_{n+1}\big) = \gamma_n \times \big(a - Y_{n+1}\big).$$

*Formulation intégrale.* — La fonction $F : \mathbb{R}^d \to \mathbb{R}^d$ est de forme intégrale

$$F(x) = \int_E \Phi(x, y)\, \mu(\mathrm{d}y)$$

où $\mu$ désigne une mesure de probabilité donnée sur un espace auxiliaire $E$, et $\Phi : \mathbb{R}^d \times E \to \mathbb{R}^d$ est une fonction bornée. De telles intégrales étant bien souvent difficilement calculables analytiquement, il est parfois plus aisé de simuler une variable aléatoire $Z$ de loi $\mu$ sur $E$. On remarquera que $F$ peut alors s'écrire

$$F(x) = \mathbb{E}[\Phi(x, Z)].$$

Pour construire l'algorithme de Robbins–Monro correspondant à cette situation, on remplace à chaque étape $n$ la valeur difficilement accessible $F(X_n)$ par sa « valeur simulée » $\Phi(X_n, Z_{n+1})$, où $(Z_n)_{n\geqslant 1}$ désigne une suite de variables aléatoires indépendantes de même loi que $Z$.

Principe. — *Ceci nous conduit à la chaîne suivante*

$$X_{n+1} - X_n = \gamma_n\big(F(x_\alpha) - \Phi(X_n, Z_{n+1})\big) = \gamma_n \times \big(a - \Phi(X_n, Z_{n+1})\big),$$

*où $(Z_n)_{n\geqslant 1}$ désigne une suite de variables aléatoires indépendantes de même loi que $Z$.*

## 4. **Estimation d'une médiane**

Soit $F : \mathbb{R} \to \mathbb{R}$ la fonction de répartition d'une variable aléatoire réelle $Y$

$$F(x) = \mathbb{P}\{Y \leqslant x\} = \mathbb{E}\big[\mathbb{1}_{\{Y \leqslant x\}}\big]$$

L'objectif de cette étude est d'estimer numériquement la médiane $x_{1/2} \in \mathbb{R}$ de cette distribution. On rappelle que la médiane est le — un — point $x_{1/2} \in \mathbb{R}$ séparant en deux parties les masses de probabilités d'une variable réelle

$$\mathbb{P}\{Y \leqslant x_{1/2}\} = \frac{1}{2} = \mathbb{P}\{Y > x_{1/2}\}.$$

Exercice 1 (*théorique et pratique*). — (i) Donner des conditions suffisantes sur la fonction de répartition pour que la condition (1) soit satisfaite (pour $a = 1/2$ et $x_\alpha = x_{1/2}$).

(ii) Soit $\Phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ la fonction définie par $\Phi(x, y) = \mathbb{1}_{]-\infty,x]}(y)$. Montrer que $F(x) = \mathbb{E}[\Phi(x, Y)]$.

(iii) Construire un algorithme de Robbins–Monro permettant d'approcher cette médiane.

(iv) Tester cet algorithme pour des lois simples où la condition (1) est satisfaire (loi uniforme sur $[0, 1]$, lois exponentielles, lois normales).

(v) Changer le niveau $\alpha = 1/2$ en particulier pour les lois normales (on rappelle que les quantiles d'une loi normale sont calculés avec Scilab par `cdfnor("X", ...)`).

(vi) On peut se demander ce qu'il se passe lorsque lorsqu'il n'existe pas de $x_{1/2}$ tel que $F(x_{1/2}) = 1/2$, ou qu'au contraire ils forment un intervalle non vide. Les lois les plus simples pour faire cet examen sont les lois de Bernoulli $\mathcal{B}(1, p)$ avec $p = 0$ ou $p = 1/2$ par exemple. Commenter les résultats et expliquer.

## Références

[1] Del Moral (Pierre), *Ingénierie Stochastique*,
`http://www.math.u-bordeaux1.fr/~delmoral/eng.html`.

[2] Chancelier (J.-P.), Delebecque (F.), Gomez (C.), Goursat (M.), Nikoukhah (R.), Steer (S.), *Introduction à Scilab, deuxième édition*, Springer-Verlag France (2007).

# TRAVAUX PRATIQUES N° 2. — ALGORITHME DE ROBBINS–MONRO, ÉLÉMENTS D'EXPLICATIONS

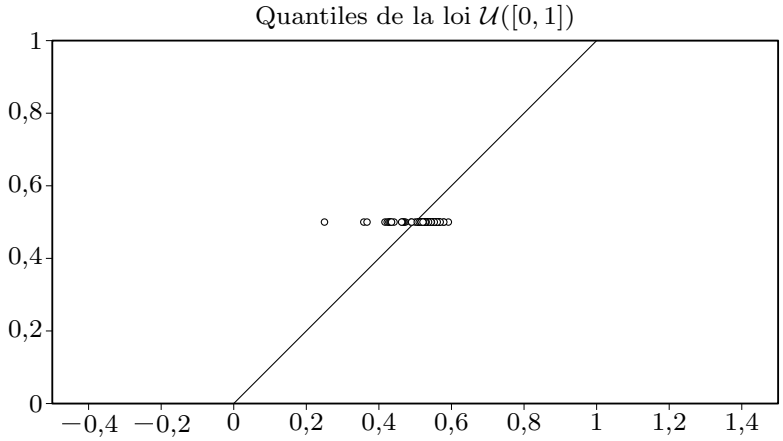
## 3. **Estimation d'une médiane**

EXERCICE 1. — (i) Il faut et suffit que $F$ atteigne $1/2$ en un point de croissance stricte.
(ii) Évident, vu à la section précédente.
(iii) Voici une illustration :

```
// Estimation d'une m\'ediane

clear(); mode(0); n = 100;

function quantile = RMquantile(alpha, Z, Xzero)
    // local n i Phi X;
    n = size(Z, 1);
    X = zeros(n+1);
    X(1) = Xzero;
    for i = 1:n
        if X(i) < Z(i) then Phi = 0; else Phi = 1; end
        X(i+1) = X(i)+(1/i)*(alpha-Phi);
        plot(X(i+1), alpha, "o");
    end
    quantile = X(n+1);
endfunction
```

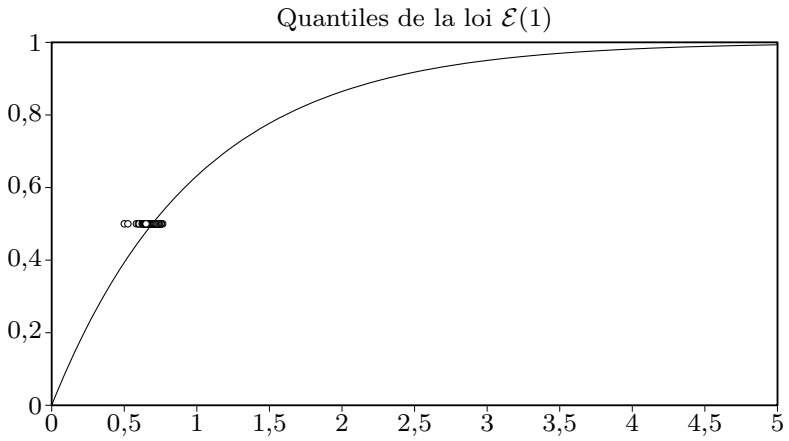
(iv) Pour la loi uniforme sur $[0, 1]$ :

```
// loi uniforme sur [0, 1]

Z = grand(n, 1, "def");

scf(1); clf();
xtitle("Quantiles de la loi U([0, 1])");
plot([-0.5; 0; 1; 1.5], [0; 0; 1; 1], "-");
mprintf("q(0.5) approx %f\n", RMquantile(0.5, Z, 0));
```



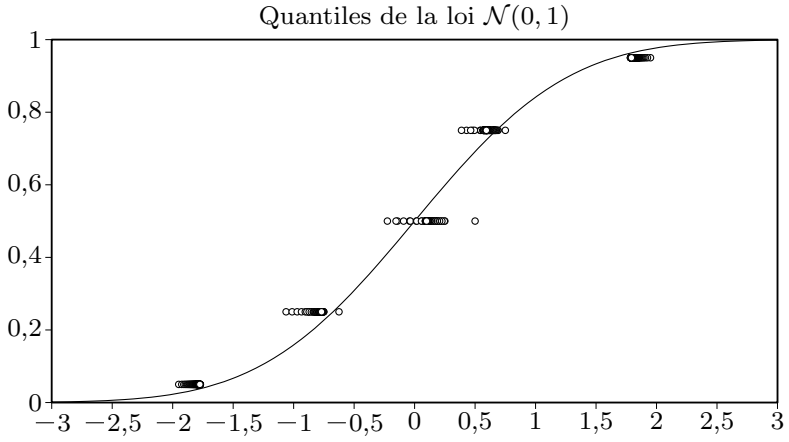

Pour la loi exponentielle de paramètre $\lambda = 1$ :

```
// loi exponentielle de param\‘etre $lambda=1$

lambda = 1; Z = -log(grand(n, 1, "def"))/lambda;

scf(2); clf();
xtitle("Quantiles de la loi E(1)");
x = linspace(0, 5, 20); y = 1-exp(-lambda*x);
plot(x, y, "-");
mprintf("q(0.5) approx %f\n", RMquantile(0.5, Z, 0));
```




Pour la loi normale de paramètres $m = 0$ et $\sigma^2 = 1$ :

```
// loi normale de param\‘etres $m=0$ et $\sigma^2=1$

Z = grand(n, 1, "nor", 0, 1);

scf(3); clf();
xtitle("Quantiles de la loi N(0, 1)");
x = linspace(-3, 3, 20); y = cdfnor("PQ", x, zeros(1, 20), ones(1, 20));
plot(x, y, "-");
mprintf("q(0.5) approx %f\n", RMquantile(0.5, Z, 0));
```

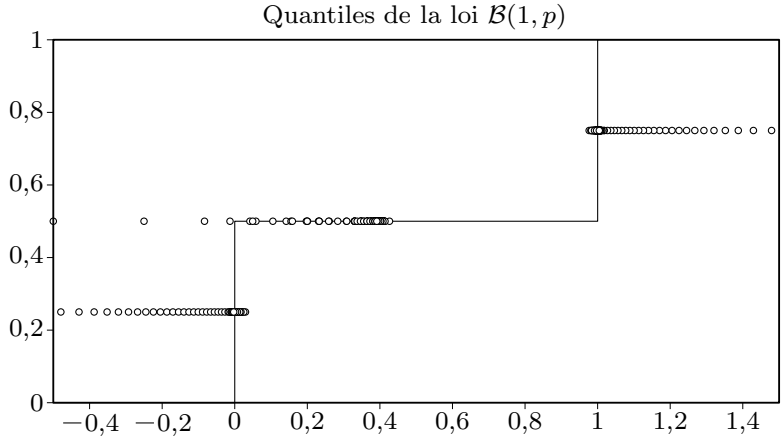



(v) Avec le même tirage $Z$, on peut approcher des quantiles différents. Il est aussi souhaitable d'ajuster la valeur initiale.

```
mprintf("q(0.05) approx %f\n", RMquantile(0.05, Z, -2));
mprintf("q(0.25) approx %f\n", RMquantile(0.25, Z, -1));
mprintf("q(0.5) approx %f\n", RMquantile(0.5, Z, 0));
mprintf("q(0.75) approx %f\n", RMquantile(0.75, Z, 1));
mprintf("q(0.95) approx %f\n", RMquantile(0.95, Z, 2));
```

(vi) Pour la loi de Bernoulli de paramètre $p \in [0, 1]$ :

```
// loi de Bernoulli de param\‘etre $p\in[0, 1]$

p = 0.5; Z = bool2s(grand(n, 1, "def") > 1-p);

scf(4); clf();
xtitle("Quantiles de la loi B(1, 0.5)");
plot([-0.5, 0, 0, 1, 1, 1.5], [0, 0, 1-p, 1-p, 1, 1], "-");
mprintf("q(0.25) approx %f\n", RMquantile(0.25, Z, 0));
mprintf("q(0.5) approx %f\n", RMquantile(0.5, Z, 0));
mprintf("q(0.75) approx %f\n", RMquantile(0.75, Z, 2));
```




### Code MetaPost

```
% Estimation d'une m\'ediane

vardef RMquantile(expr alpha)(suffix Z)(expr Xzero) =
    save n, i, Phi, X;
    n = Z.n;
    declare(X, n+1);
    X[1] = Xzero;
    for i = 1 upto n:
        if X[i] < Z[i]: Phi := 0; else: Phi := 1; fi
        X[i+1] = X[i]+(1/i)*(alpha-Phi);
        if (X[i+1] >= hmin) and (X[i+1] <= hmax):
            gdot(X[i+1], alpha);
        fi
    endfor
    X[n+1]
enddef;

% loi uniforme sur [0, 1]
n := 100;
beginfig(thisfig);
    declare(Z, n);
    for i = 1 upto n: Z[i] = uniformdeviate 1; endfor
    setrange(-0.5, 1.5, 8cm, 0, 1, 4cm);
    pickup rule.nib;
    draw gcoord(hmin, 0)--gcoord(0, 0)--gcoord(1, 1)--gcoord(hmax, 1);
    message "q(0.5) approx "&decimal RMquantile(0.5, Z, 0);
```

```
    gtitle.top(btex Quantiles de la loi $\mathcal U([0, 1])$ etex, 0);
    autograd;
endfig;

% loi exponentielle de param\‘etre 1
beginfig(thisfig);
    lambda := 1;
    declare(Z, n);
    for i = 1 upto n: Z[i] = -ln(1-uniformdeviate 1)/lambda; endfor
    setrange(0, 5, 8cm, 0, 1, 4cm);
    pickup rule.nib;
    plot("1-exp(-lambda*x)", hmin, hmax, 20);
    message "q(0.5) approx "&decimal RMquantile(0.5, Z, 0);
    gtitle.top(btex Quantiles de la loi $\mathcal E(1)$ etex, 0);
    autograd;
endfig;

% loi normale de param\‘etres $m=0$ et $\sigma^2=1$
beginfig(thisfig);
    declare(Z, n);
    for i = 1 upto n: Z[i] = normaldeviate; endfor
    setrange(-3, 3, 8cm, 0, 1, 4cm);
    pickup rule.nib;
    plot("normalcdf(x)", hmin, hmax, 20);
    message "q(0.05) approx "&decimal RMquantile(0.05, Z, -2);
    message "q(0.25) approx "&decimal RMquantile(0.25, Z, -1);
    message "q(0.5) approx "&decimal RMquantile(0.5, Z, 0);
    message "q(0.75) approx "&decimal RMquantile(0.75, Z, 1);
    message "q(0.95) approx "&decimal RMquantile(0.95, Z, 2);
    gtitle.top(btex Quantiles de la loi $\mathcal N(0, 1)$ etex, 0);
    autograd;
endfig;

% loi de Bernoulli de param\‘etre $p\in[0, 1]$
p := 0.5;
beginfig(thisfig);
    declare(Z, n);
    for i = 1 upto n:
        Z[i] = if uniformdeviate 1 < 1-p: 0 else: 1 fi;
    endfor
    setrange(-0.5, 1.5, 8cm, 0, 1, 4cm);
    pickup rule.nib;
    draw gcoord(hmin, 0)--gcoord(0, 0)--gcoord(0, 1-p)
    --gcoord(1, 1-p)--gcoord(1, 1)--gcoord(hmax, 1);
    message "q(0.25) approx "&decimal RMquantile(0.25, Z, 0);
    message "q(0.5) approx "&decimal RMquantile(0.5, Z, 0);
    message "q(0.75) approx "&decimal RMquantile(0.75, Z, 2);
    gtitle.top(btex Quantiles de la loi $\mathcal B(1, p)$ etex, 0);
    autograd;
endfig;
```

Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 3. — ALGORITHMES DE METROPOLIS & CO

Avec cette fiche de travaux pratiques ou dirigés, nous allons aborder une famille d'algorithmes dits de Metropolis, MCMC (Markov Chains Monte Carlo), etc. Ils suivent tous la même idée : une petite particule se déplace sous l'action conjuguée de l'agitation thermique et du gradient d'un potentiel. En laissant le temps s'écouler et la température refroidir, la particule va se figer dans un état particulier de l'espace.

Suivant les applications recherchées (simulation, optimisation, etc.), l'état limite obtenu pourra représenter une valeur d'une variable aléatoire de loi donnée (algorithme d'Hastings–Metropolis), un extremum global de fonction (recuit simulé), et la mise en œuvre de l'algorithme pourra présenter des particularités propres au problème étudié. D'autres différences entre algorithmes de ce type pourront apparaître : implémentation de la fonction de potentiel, schéma de variation (décroissance) de la température au fil du temps, etc. Une seule application devrait suffire pour comprendre l'algorithme, c'est classiquement celle au problème du voyageur de commerce qu'on retient.

Nous conseillons de lire — à ses moments perdus — la courte présentation suivante du recuit simulé

`http://www.lps.ens.fr/~weisbuch/livre/b9.html`

pour sa clarté, l'absence voulue de formalisme mathématique, et l'exposition d'applications en particulier en imagerie. À l'opposé, nous donnons en annexe quelques paragraphes de cours sur les chaînes de Markov à temps discret et espace d'états fini (m2mfa 2007–08), qui complète le cours de première année (2m12). Cette annexe peut servir de petite référence théorique à l'occasion. L'activité proprement dite est basée sur les enseignements de Pierre Del Moral et Bernard Ycart.

## 1. **Le voyageur de commerce**

Ce problème classique d'optimisation combinatoire peut se définir de la façon suivante. On se donne un ensemble fini $V = \{v_1, \dots, v_N\}$ de $N \geqslant 1$ villes ainsi que la distance mutuelle entre chaque ville. Ce qui revient à se donner une distance $d : V \times V \to \mathbb{R}_+$. Le voyageur de commerce partant de la ville $v_1$ doit trouver une permutation

$$\sigma = \begin{pmatrix} 1 & \dots & N \\ \sigma(1) & \dots & \sigma(N) \end{pmatrix}$$

telle que $\sigma(1) = 1$, de sorte à minimiser la distance

$$d(v_{\sigma(1)}, v_{\sigma(2)}) + \cdots + d(v_{\sigma(n-1)}, v_{\sigma(N)}) + d(v_{\sigma(N)}, v_{\sigma(1)})$$

qu'il aura parcourue en effectuant le circuit $v_{\sigma(1)} = v_1 \to v_{\sigma(2)} \to \cdots \to v_{\sigma(N)} \to v_{\sigma(1)} = v_1$. On notera par la suite $\mathfrak{G}_N$ l'ensemble des permutations de $\{1, \dots, N\}$. Pour chaque $\sigma \in \mathfrak{G}_N$, on utilisera la convention $\sigma(N+1) = \sigma(1)$ et on notera

$$U : \sigma \in \mathfrak{G}_N \longmapsto U(\sigma) = \sum_{p=1}^{N} d(v_{\sigma(p)}, v_{\sigma(p+1)}) \in \mathbb{R}_+.$$

Pour chaque $p \geqslant 1$, on désigne par $\mathfrak{G}_N(p)$ et $\mathfrak{G}_N^*(p)$ les sous-ensembles de $\mathfrak{G}_N$ définis par

$$\mathfrak{G}_N(p) = \{\sigma \in \mathfrak{G}_N : \sigma(p) = 1\}$$
$$\mathfrak{G}_N^*(p) = \{\sigma \in \mathfrak{G}_N(p) : U(\sigma) = \min_{\tau \in \mathfrak{G}_N(p)} U(\tau)\}.$$

On prendra comme espace d'état $E = \mathfrak{G}_N$. Il existe sur cet espace diverses façons de spécifier des systèmes de voisinage :

*permutation de deux indices,* $\sigma \sim \tau$ si et seulement si il existe $i \leqslant j$ tels que

$$\sigma = \begin{pmatrix} 1 & \dots & i & \dots & j \dots N & & \\ \tau(1) & \dots & \tau(j) & \dots & \tau(i) & \dots & \tau(N) \end{pmatrix};$$

*inversion d'une suite d'indices,* $\sigma \sim \tau$ si et seulement si il existe $i \leqslant j$ tels que

$$\sigma = \begin{pmatrix} 1 & \dots & i+0 & \dots & i+p & \dots & i+(j-i)=j & \dots & N \\ \tau(1) & \dots & \tau(j-0) & \dots & \tau(j-p) & \dots & \tau(j-(j-i)) = \tau(j) & \dots & \tau(N) \end{pmatrix}$$

Exercice 1 (*théorique*). — (i) Dans les deux cas, montrer que pour chaque $\sigma$ et $\tau \in \mathfrak{G}_N$ il existe une suite $(\sigma_0, \dots, \sigma_N)$ d'éléments de $\mathfrak{G}_N$ telle que $\sigma_0 = \sigma \sim \sigma_1 \sim \cdots \sim \sigma_N = \tau$. Pour réaliser l'algorithme de recuit, on utilisera un noyau d'exploration $K(\sigma, \tau)$ défini par l'un de ces systèmes de voisinage, c'est-à-dire

$$K(\sigma, \tau) = \frac{1}{|V(\sigma)|} \mathbb{1}_{V(\sigma)}(\tau), \qquad V(\sigma) = \{\tau \in \mathfrak{G}_N, \tau \sim \sigma\}.$$

(ii) Vérifier que $\sigma \sim \tau \iff K(\sigma, \tau) > 0$.

(iii) En déduire que pour tous $\sigma$, $\tau \in \mathfrak{G}_N$, $K^N(\sigma, \tau) > 0$.

## 2. Implémentation en Scilab

### 2.1. Pseudo-code

Le pseudo-code suivant met en œuvre le recuit simulé tel que décrit plus haut, en commençant à l'état $\sigma_0$ et continuant jusqu'à un maximum de $k_{\max}$ étapes ou jusqu'à ce qu'un état ayant pour énergie $e_{\max}$ ou moins est trouvé. L'appel `voisin(s)` génère un état voisin aléatoire d'un état $\sigma$. L'appel `aleatoire()` renvoie une valeur aléatoire dans l'intervalle $[0, 1]$. L'appel `temp(r)` renvoie la température à utiliser selon la fraction r du temps total déjà dépensé.

```
s := s0
e := E(s)
k := 0
tant que k < kmax et e > emax
    sn := voisin(s);
    en := E(sn);
    si en < e ou aleatoire() < P(en-e, temp(k/kmax)) alors
        s := sn; e := en
    k := k+1
retourne s
```

### 2.2. RECUIT.SCI

On tapera le code suivant en essayant de comprendre à quoi il sert dans un fichier nommé `recuit.sci`. C'est une version modifiée de

`http://ljk.imag.fr/membres/Bernard.Ycart/codes/scilab/recuit.sci`

```
function [v, dist] = initialise(nv);
    // Tire les coordonn\'ees de nv villes au hasard dans
    // le carr\'e unit\'e. Calcule la matrice des distances.
    // Coordonn\'ees des villes :
    v = rand(nv, 2);
    // Matrice des distances :
    dist = (v(:, 1)*ones(1, nv)-ones(nv, 1)*(v(:, 1))').^2;
    dist = dist+(v(:, 2)*ones(1, nv)-ones(nv, 1)*(v(:, 2))').^2;
    dist = sqrt(dist);
endfunction

function trace_parcours(v);
    // Repr\'esente les villes et le circuit dans le carr\'e unit\'e.
    xbasc(); square(0, 0, 1, 1);
    plotframe([0, 0, 1, 1], [0, 1, 0, 1], [%f, %f]);
    plot2d([v(:, 1); v(1, 1)], [v(:, 2); v(1, 2)], 1, "000");
    // xset("mark", -9, 4);
    plot2d(v(:, 1), v(:, 2), -9, "000");
endfunction

// attention \`a bien passer tous les param\`etres
function e = energie(ordre, dist);
    // fonction d'energie : somme des distances du
    // circuit dans l'ordre donn\'e en entree.
    e = sum(diag(dist(ordre, ..
        [ordre($), ordre([1:size(ordre, "*")])])));//$ colorisation emacs
endfunction

// attention \`a bien passer tous les param\`etres
function o = permuter(ordre, nv);
    // Permute 5 indices cons\'ecutifs d'un vecteur d'ordre
    v = grand(1, 1, "uin", 1, nv);// indice de d\'epart
    o = ordre;// ordre initial
    if v > 1 then o = o([[v:nv], [1:v-1]]); end// d\'ecalage
    o5 = grand(1, "prm", [1:5]');// permutation
    o([1:5]) = o(o5');// ordre modifie
endfunction

// attention \`a bien passer tous les param\`etres
function [o, e] = recuit(ordre, dist, h, unsurT);// unsurT = 1/T
    // Simule nbre pas de l'algorithme de recuit simul\'e
    // pour le probl\`eme du voyageur de commerce.
    // Retourne le nouvel ordre et le vecteur des valeurs
    // successives de la fonction f.
    o = ordre;
    f1 = energie(o, dist);
    e = [f1];
```

```
    n = 0;
    for ut = 1:unsurT;
        palier = int(exp(ut*h));
        while n < palier,
            o2 = permuter(o, nv);
            f2 = energie(o2, dist);
            if f2 < f1 then o = o2; f1 = f2;
            else
                p = exp((f1-f2)*ut);
                if rand(1, 1) < p then o = o2; f1 = f2; end
            end
            e = [e, f1];
            n = n+1;
        end
    end
endfunction
```

### 2.3. RECUIT.SCE

On tapera le code suivant en essayant de comprendre à quoi il sert dans un fichier nommé `essai_recuit.sce`. C'est une version modifiée de

`http://ljk.imag.fr/membres/Bernard.Ycart/codes/scilab/essai_recuit.sce`

```
getf("recuit.sci");

nv = 20;// nombre de villes
[ville, distance] = initialise(nv);

scf(0); clf();
trace_parcours(ville);// circuit initial
h = 0.6; u = 17;// param\‘etres
[o, e] = recuit([1:nv], distance, h, u);// ex\’ecution
trace_parcours(ville(o’, :));// circuit final

scf(1); clf();
x = [1:size(e, "*")];
plot2d(x, e);// energie
```

Exécuter `essai_recuit.sce`. Analyser les résultats graphiques.

## Annexe : compléments de cours, 2m12

### 2.1. Simulation de lois, l'algorithme de Hastings–Metropolis

Soit $E$ un espace d'états au plus dénombrable et $\mu$ une mesure finie non nulle sur $E$. On souhaite simuler la loi de probabilité $\pi = \mu/\mu(E)$. Deux problèmes peuvent se poser. Le premier est le calcul de $\mu(E)$ qui est dans certains cas pratiques difficile, le second est qu'on veut disposer d'une méthode qui ne nécessite pas trop de calculs. Pour notre propos, on supposera $\mu\{x\} > 0$ pour tout $x \in E$, ce qui revient à ignorer les états de mesure nulle.

*Exemple.* — Soit $E$ l'ensemble des permutations $\sigma$ de $\{1,\dots,10\}$ vérifiant $\sum_{j=1}^{10} j\times\sigma(j) \geqslant 250$. Cet ensemble est fini, et non vide puisqu'il contient notamment l'ensemble des permutations laissant fixe $\{7,8,9,10\}$. Nous souhaitons choisir uniformément une permutation dans $E$. Ici $\mu$ est la mesure uniforme sur $E$ et $\mu(E) = \mathrm{Card}(E)$, ce cardinal semblant difficile à expliciter par des méthodes combinatoires ou autres.

Idée. — *Construire une matrice de transition $P$ sur $E$, irréductible et apériodique et telle que à $\mu$ en soit une mesure réversible. En effet, on aura alors :*

(i) *La mesure de probabilité $\pi = \mu/\mu(E)$ est invariante par $P$.*

(ii) *Il y a convergence vers l'équilibre : si $(X_n)_{n\geqslant 0}$ est une chaîne de Markov $(\lambda, P)$, alors, pour tout $x \in E$, $\mathbb{P}\{X_n = x\} \to \pi\{x\}$ quand $n$ tend vers l'infini.*

*Construction.* — On considère une matrice de transition $K$ sur $E$, irréductible et apériodique telle que $K(x,y) = 0$ implique $K(y,x) = 0$ — construite par exemple à partir d'un graphe de sommets $E$.

Soit $D = \{(x,y) \in E\times E : K(x,y) > 0\}$ et soit $\alpha : D \to ]0,1]$ défini par

$$\alpha(x,y) = \min\left(\frac{K(y,x)\times\mu\{y\}}{K(x,y)\times\mu\{x\}}, 1\right),$$

où on notera l'importance de l'hypothèse de stricte positivité de $\mu$ pour cette définition. On définit la matrice $P$ par

$$x \neq y \in E, \qquad P(x,y) = \begin{cases} K(x,y)\times\alpha(x,y) & \text{si } (x,y)\in D,\\ 0 & \text{sinon,}\end{cases}$$

et

$$x \in E, \qquad P(x,x) = 1 - \sum_{y\neq x} P(x,y).$$

Ceci définit bien une matrice stochastique : les sommes sur les lignes valent 1 ; les coefficients non diagonaux sont positifs ; et pour les coefficients diagonaux on a $0 \leqslant \sum_{y\neq x} P(x,y) \leqslant \sum_{y\neq x} K(x,y) \leqslant 1$, et donc $P(x,x) \geqslant 0$.

L'irréductibilité de $P$ est une conséquence immédiate de celle de $K$ puisque si on a, pour $x_0, x_1, \dots, x_{n-1}, x_n \in E$,

$$K(x_0,x_1)\times\cdots\times K(x_{n-1},x_n) > 0 \qquad \text{alors} \qquad P(x_0,x_1)\times\cdots\times P(x_{n-1},x_n) > 0$$

et réciproquement. L'apériodicité de $P$ résulte aussi de celle de $K$ puisqu'en utilisant ce qui précède constate que $K^n(x,y) > 0$ si et seulement si $P^n(x,y) > 0$. Nous allons voir que la mesure $\mu$ est réversible pour $P$. Soient $(x,y) \in D$, $x \neq y$. Si par exemple $\alpha(x,y) \leqslant 1$, alors $\alpha(y,x) = 1$ et on a

$$\begin{aligned}\mu\{x\}\times P(x,y) = \mu\{x\}\times K(x,y)\times\alpha(x,y) &= \mu\{x\}\times K(x,y)\times\frac{K(y,x)\times\mu\{y\}}{K(x,y)\times\mu\{x\}}\\ &= K(y,x)\times\mu\{y\} = \mu\{y\}\times K(y,x)\times\alpha(y,x) = \mu\{y\}\times P(y,x).\end{aligned}$$

Pour $(x,y) \notin D$, $x \neq y$, on a $\mu\{x\}\times P(x,y) = 0 = \mu\{y\}\times P(y,x)$. Enfin, il n'y a rien à dire pour le cas $x = y$. La matrice $P$ ainsi construite vérifie les conditions imposées.

## 2.2. Le recuit simulé

La méthode du recuit provient de la métallurgie et de la verrerie : afin d'éliminer dans la structure d'un produit (métal, verre) des inhomogénéités ou fractures qui pourraient le rendre cassant, on le chauffe puis le laisse lentement refroidir afin que sa structure interne se réorganise de manière plus stable. Ceci a inspiré une méthode probabiliste de recherche de

minima globaux de fonction. Elle utilise des chaînes de Markov inhomogènes, l'inhomogénéité provenant de l'abaissement progressif de la température, et donc la modification au fil du temps des règles d'évolution de la chaîne.

Soit $U : E \to \mathbb{R}_+$ une fonction positive bornée. Cette fonction sera appelée *énergie* ou *potentiel* sur l'espace des états $E$, espace toujours supposé au plus dénombrable muni de sa tribu discrète. On cherche à déterminer l'ensemble $E_U$ de ses minima globaux, ensemble qui peut être éventuellement vide lorsque $E$ est infini dénombrable.

On explore aléatoirement l'espace des états selon une matrice de transition $K$ sur $E$ vérifiant

$$\begin{cases} \text{(i) il existe une mesure } \lambda \text{ sur } E \text{ réversible pour } K \ ; \\ \text{(ii) il existe } p \in \mathbb{N} \text{ tel que } K^p \text{ ait tous ses coefficients positifs.} \end{cases} \qquad (\mathcal{P})$$

La matrice $K$ permettra de déterminer un nouvel état vers lequel on pourra choisir ou non de transiter.

*Remarque.* — La condition (i) implique que $\lambda$ est une mesure invariante de $K$. La condition (ii) implique que pour tout $n \geqslant 0$, $K^{p+n}$ a tous ses coefficients strictement positifs ; en particulier, $K$ est apériodique. La réciproque est vraie lorsque $E$ est fini.

*Exemple.* — Lorsque $E$ est un ensemble fini, on peut construire $K$ à partir d'un graphe connexe de sommets $E$, comme il a été vu précédemment. Lorsque $E$ est infini, on peut, par exemple identifier $E$ à $\mathbb{Z}$, se donner une mesure de probabilité $\mu$ sur $\mathbb{Z}$ partout strictement positive, symétrique par rapport à 0, et définir $K(x, y) = \mu\{y - x\}$. La condition (i) est alors satisfaite avec $\lambda$ la mesure uniforme sur $\mathbb{Z}$ et (ii) est vérifiée avec $p = 1$.

La température est notée $(T_n)_{n\geqslant 0}$ ; c'est une suite numérique, décroissante de nombres strictements positifs tendant vers 0. Pour tout $n \geqslant 0$, on note $\beta_n = 1/T_n$ l'inverse de la température. La suite $(\beta_n)_{n\geqslant 0}$ est croissante et tend vers l'infini. On définit de manière algorithmique une chaîne de Markov inhomogène $(X_n)_{n\geqslant 0}$ qui se stabilise lorsque $n$ tend vers l'infini dans l'ensemble des minima globaux $E_U$ de $U$.

Heuristique[1]. — *Supposons que pour $n \geqslant 0$, l'algorithme soit dans un état $X_n = x \in E$ d'énergie $U(x)$. Puisqu'on ignore si c'est un minimum global, on choisit un nouvel état $y$ selon la loi $K(x, .)$. Alors*

— *si $U(y) \leqslant U(x)$, on pose $X_{n+1} = y$ ;*

— *si $U(y) > U(x)$, alors on choisit*

$$X_{n+1} = \begin{cases} x & \text{avec probabilité } 1 - \mathrm{e}^{-\beta_n(U(y)-U(x))}, \\ y & \text{avec probabilité } \mathrm{e}^{-\beta_n(U(y)-U(x))}. \end{cases}$$

*Remarques.* — *a*) On constate que le passage d'un état $x$ à un état $y$ d'énergie strictement supérieure est d'autant plus difficile que la différence des énergies est grande.

*b*) Pour des températures proches de 0, c'est-à-dire des $\beta_n$ grands, il est de plus en plus difficile de passer à un état d'énergie supérieure, il est donc plus fortement probable de passer dans un état d'énergie inférieure ou de rester dans le même état.

---

1. En optimisation combinatoire, Théorie des graphes et Théorie de la complexité, une heuristique est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile. Une heuristique, où méthode approximative, est donc le contraire d'un algorithme exact qui trouve une solution optimale pour un problème donné. L'intérêt de l'heuristique étant que pour les problèmes NP-difficiles, les algorithmes exacts connus sont tous de complexité exponentielle et donc sans aucun intérêt en pratique (ni en théorie d'ailleurs). On utilise presque systématiquement une heuristique pour obtenir une première solution réalisable dans un processus de résolution exacte. (Wikipédia)

La difficulté est de choisir la suite de température $(T_n)_{n\geqslant 0}$ pour que $(X_n)_{n\geqslant 0}$ explore suffisamment d'états et converge vers un minimum global, autrement dit

— $(X_n)_{n\geqslant 0}$ est une chaîne de Markov (inhomogène) récurrente ;

— $(X_n)_{n\geqslant 0}$ converge en loi vers une mesure portée par l'ensemble des minima globaux de $U$.

*Cas de la tempéture constante.* — On suppose la suite $(T_n)_{n\geqslant 0}$ constante, il en est donc de même pour $(\beta_n)_{n\geqslant 0}$. Ainsi, soit $\beta > 0$. On considère la chaîne de Markov homogène $(X_n)_{n\geqslant 0}$ de matrice de transition $P_\beta$ définie, pour $x$ et $y \in E$, par

$$P_\beta(x,y) = K(x,y)\,\mathrm{e}^{-\beta\max(U(y)-U(x),0)} = \begin{cases} K(x,y) & \text{si } U(y) \leqslant U(x), \\ K(x,y)\,\mathrm{e}^{-\beta(U(y)-U(x))} & \text{si } U(y) > U(x). \end{cases}$$

Cette matrice correspond à la description faite dans l'heuristique pour une température constante.

Soit

$$Z_\beta = \sum_{y\in E} \lambda\{y\}\,\mathrm{e}^{-\beta U(y)}.$$

Si $Z_\beta < \infty$, on définit la *mesure de Gibbs*, ou *mesure de Gibbs–Boltzmann*, par

$$\mu_\beta\{x\} = \frac{\lambda\{x\}}{Z_\beta}\,\mathrm{e}^{-\beta U(x)}, \qquad x \in E.$$

C'est une mesure de probabilité associée à la fonction d'énergie $U$ et à la température $T = 1/\beta$. Elle dépend évidemment de l'espace mesuré $(E, \lambda)$.

*Remarque.* — La constante $Z_\beta$ est généralement difficile à calculer. C'est pourquoi les physiciens pour simuler une mesure de Gibbs se servent de la méthode de Hastings–Metropolis, méthode qui a d'ailleurs inspiré celle du recuit simulé.

Proposition 1. — *On suppose que les propriétés $(\mathcal{P})$ sont vérifiées, et que $Z_\beta$ est fini. Alors la chaîne de Markov $(X_n)_{n\geqslant 0}$ de matrice de transition $P_\beta$ est irréductible, apériodique, de loi de probabilité réversible — et donc invariante — la mesure de Gibbs $\mu_\beta$.*

*Démonstration.* — Puisque $K$ est irréductible et apériodique, et que le facteur

$$\exp\bigl(-\beta\max(U(y)-U(x),0)\bigr) \qquad x, y \in E,$$

est toujours strictement positif, $P_\beta$ est aussi irréductible et apériodique (voir le raisonnement semblable pour l'algorithme de Hastings–Metropolis). Il reste simplement à voir que $\mu_\beta$ est réversible pour $P_\beta$. Pour $x$, $y \in E$, on a

$$\begin{aligned} Z_\beta \times \mu_\beta\{x\} \times P_\beta(x,y) &= \bigl(\lambda\{x\}\,\mathrm{e}^{-\beta U(x)}\bigr) \times \bigl(K(x,y)\,\mathrm{e}^{-\beta\max(U(y)-U(x),0)}\bigr) \\ &= \bigl(\lambda\{x\}K(x,y)\bigr) \times \bigl(\mathrm{e}^{-\beta(\max(U(y)-U(x),0)+U(x))}\bigr) \\ &= \bigl(\lambda\{y\}K(y,x)\bigr) \times \bigl(\mathrm{e}^{-\beta(\max(U(y),U(x))}\bigr) \\ &= \bigl(\lambda\{y\}K(y,x)\bigr) \times \bigl(K(y,x)\,\mathrm{e}^{-\beta(\max(0,U(x)-U(y))+U(y))}\bigr) \\ &= \bigl(\lambda\{y\}\,\mathrm{e}^{-\beta U(y)}\bigr) \times \bigl(K(y,x)\,\mathrm{e}^{-\beta\max(U(x)-U(y),0)}\bigr) \\ &= Z_\beta \times \mu_\beta\{y\} \times P_\beta(y,x) \end{aligned}$$

du fait de la réversibilité de $\lambda$ pour $K$. □

Corollaire 1. — *En conservant les mêmes hypothèses qu'à la proposition précédente, pour toute loi initiale $\eta$ et tout $\beta > 0$, on a*

$$\lim_{n\to\infty}\left(\sup_{A\subset E}\bigl|\eta P_\beta^n(A) - \mu_\beta(A)\bigr|\right) = 0.$$

*Démonstration.* — Par convergence vers l'équilibre, on sait que pour tout $x \in E$,

$$\eta P_\beta^n\{x\} - \mu_\beta\{x\} \longrightarrow 0 \qquad \text{quand } n \text{ tend vers l'infini.}$$

Soient $\varepsilon > 0$ et $F \subset E$ fini tel que $\mu_\beta(F) \geqslant 1 - \varepsilon/4$, ou encore $\mu_\beta(F^c) \leqslant \varepsilon/4$. Soit $N \geqslant 0$ tel que pour tout $n \geqslant N$ et tout $x \in F$,

$$\left|\eta P_\beta^n\{x\} - \mu_\beta\{x\}\right| \leqslant \frac{\varepsilon}{4\,\mathrm{Card}(F)}\,.$$

On a alors, par l'inégalité triangulaire,

$$\begin{aligned}\left|1 - \eta P_\beta^n(F)\right| &\leqslant \left|1 - \mu_\beta(F)\right| + \left|\mu_\beta(F) - \eta P_\beta^n(F)\right| \\ &\leqslant \left|1 - \mu_\beta(F)\right| + \sum_{x\in F}\left|\mu_\beta\{x\} - \eta P_\beta^n\{x\}\right| \leqslant \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{\varepsilon}{2}\,,\end{aligned}$$

et ainsi $\eta P_\beta^n(F) \geqslant 1 - \varepsilon/2$, ou encore $\eta P_\beta^n(F^c) \leqslant \varepsilon/2$. Soit maintenant $A \subset E$ une partie de $E$. On a

$$\begin{aligned}\left|\eta P_\beta^n(A) - \mu_\beta(A)\right| &\leqslant \left|\eta P_\beta^n(A\cap F) - \mu_\beta(A\cap F)\right| + \left|\eta P_\beta^n(A\cap F^c) - \mu_\beta(A\cap F^c)\right| \\ &\leqslant \sum_{x\in A\cap F}\left|\eta P_\beta^n\{x\} - \mu_\beta\{x\}\right| + \eta P_\beta^n(A\cap F^c) + \mu_\beta(A\cap F^c) \\ &\leqslant \sum_{x\in A\cap F}\left|\eta P_\beta^n\{x\} - \mu_\beta\{x\}\right| + \eta P_\beta^n(F^c) + \mu_\beta(F^c) \\ &\leqslant \frac{\varepsilon}{4\,\mathrm{Card}\,F} \times \mathrm{Card}(A\cap F) + \frac{\varepsilon}{2} + \frac{\varepsilon}{4} \leqslant \frac{\varepsilon}{4} + \frac{\varepsilon}{2} + \frac{\varepsilon}{4} = \varepsilon.\end{aligned}$$

Ainsi, pour tout $\varepsilon > 0$, il existe $N \geqslant 0$ tel que pour tout $n \geqslant N$ et $A \subset E$, on a $\left|\eta P_\beta^n(A) - \mu_\beta(A)\right| \leqslant \varepsilon$, ce qui est la propriété annoncée. □

*Remarque.* — On munit généralement un ensemble $E$ au plus dénombrable de sa tribu discrète ainsi que de sa topologie discrète. Dans ce contexte, on peut montrer que la convergence d'une suite de mesures de probabilité $(\mu_n)_{n\geqslant 0}$ vers une mesure de probabilité $\mu$ sur $E$ équivaut à la convergence des suites $(\mu_n(A))_{n\geqslant 0}$ vers $\mu(A)$ pour tout $A \subset E$ (ou encore celle de $(\mu_n\{x\})_{n\geqslant 0}$ vers $\mu\{x\}$ pour tout $x \in E$). Le type de convergence apparaissant dans ce corollaire est donc la *convergence uniforme* sur l'ensemble des parties de $E$, ce qui est un résultat fort.

Avec l'hypothèse $(\mathcal{P})$, on sait que la mesure $\lambda$ charge tous les points, c'est-à-dire $\lambda\{x\} > 0$ pour tout $x \in E$. Nous notons

$$m = \inf\{U(x) : x \in E\} \qquad \text{et rappelons que} \qquad E_U = \{x \in E : U(x) = m\}.$$

En multipliant le numérateur et le dénominateur par $\exp(\beta m)$, on peut alors réécrire la mesure de Gibbs sous la forme

$$\mu_\beta\{x\} = \frac{\lambda\{x\}\,\mathrm{e}^{-\beta(U(x)-m)}}{\lambda(E_U) + \sum_{y\notin E_U}\lambda\{y\}\,\mathrm{e}^{-\beta(U(y)-m)}}\,, \qquad x \in E.$$

Si on considère cette expression, le comportement de $\mu_\beta$ lorsque $\beta$ tend vers l'infini est immédiat. Ceci est précisé dans les deux propositions qui suivent.

Proposition 2. — *On suppose $E_U \neq \varnothing$. Alors $\mu_\beta$ tend vers la mesure de probabilité déduite de la restriction de $\lambda$ à $E_U$ lorsque $\beta$ tend vers l'infini :*

$$\lim_{\beta\to\infty}\mu_\beta\{x\} = \frac{1}{\lambda(E_U)} \times \mathbb{1}_{E_U}(x)\,\lambda\{x\}, \qquad \textit{pour tout } x \in \mathbb{E}.$$

*Démonstration.* — S'il est besoin d'une démonstration, celle-ci est laissée en exercice. □

Proposition 3. — *Soient $\varepsilon > 0$ et $E_U^\varepsilon = \{x \in E : U(x) \leqslant m + \varepsilon\}$. Alors*

$$\lim_{\beta\to\infty} \mu_\beta(E_U^\varepsilon) = 1.$$

*Démonstration.* — S'il est besoin d'une démonstration, celle-ci est laissée en exercice. □

*Cas de la température variable.* — On montre que le réglage optimal de la décroissance de la température au cours du temps est inverse logarithmique

$$\beta_n = \frac{1}{C}\ln(n + \mathrm{e}), \qquad n \geqslant 0,$$

où $\mathrm{e} = \mathrm{e}^1$ est la constante de Neper[2] dont le rôle est simplement de partir d'une température strictement positive. Si $C$ est proche de 0, c'est-à-dire $(\beta_n)_{n\geqslant 0}$ croissant rapidement, la loi limite pourrait ne pas être portée par $E_U$ et n'être portée que par des minima locaux au sens de la matrice de transition $K$, et ainsi ignorer d'autres minima, en particulier des minima globaux. Si $C$ est grand, la convergence peut être très lente, ce qui est pénalisant d'un point de vue pratique. On définit l'oscillation de $U$ selon la matrice de transition $K$ par

$$\mathrm{osc}_K(U) = \max\{U(y) - U(x) : x, y \in I \text{ tels que } K(x, y) > 0\}.$$

Théorème 1. — *On suppose la condition $(\mathcal{P})$ réalisée et $U$ bornée. Soit $(X_n)_{n\geqslant 0}$ une chaîne de Markov inhomogène de loi initiale $\eta$ définie par l'algorithme de recuit associé à la suite d'inverses de températures*

$$\beta_n = \frac{1}{C}\ln(n + \mathrm{e}), \quad n \geqslant 0, \qquad \text{avec} \quad C > p \times \mathrm{osc}_K(U),$$

*où $p \geqslant 0$ est tel que $K^p$ ait tous ses coefficients strictement positifs. Alors, on a*

$$\text{pour tout } \varepsilon > 0, \qquad \lim_{n\to\infty} \mathbb{P}\{X_n \in E_U^\varepsilon\} = 1.$$

*En particulier, si $E$ est fini,*

$$\lim_{n\to\infty} \mathbb{P}\{X_n \in E_U\} = 1.$$

*Démonstration.* — Admise. □

*Remarque.* — Il n'a été nulle part question dans ces énoncés de vitesse de convergence. Les questions de ce type sont pourtant fondamentales pour les applications numériques.

*Exemple (le problème du voyageur de commerce).* — Un voyageur de commerce doit visiter $N$ villes en partant de l'une d'entre elle et en y revenant à la fin. Le problème est de trouver un ou des trajets minimisant la distance totale (le point de départ et d'arrivée n'a pas vraiment d'importance puisque le voyageur fait une boucle). Nous considérons donc $N$ points dans un certain espace métrique de distance $d$ (ce qui compte est de pouvoir mesurer la distance entre deux villes). L'espace des états $E$ est l'ensemble des arrangements de ces $N$ villes, il s'identifie donc à l'ensemble $\mathcal{S}_N$ des permutations de $\{1, \ldots, N\}$ qui est fini de cardinal $N\,!$ ce qui peut être trop grand pour examiner chaque état. La fonction d'énergie s'écrit alors

$$U(\sigma) = \sum_{x=1}^{N-1} d(\sigma(x), \sigma(x + 1)) + d(\sigma(N), \sigma(1)), \qquad \sigma \in \mathcal{S}_N,$$

le dernier terme étant la distance à parcourir pour retourner à son point de départ.

Une structure de graphe connexe simple de sommets $\mathcal{S}_N$ est obtenue en considérant les transpositions $\mathcal{T}_N$ de $\{1, \ldots, N\}$ qui est un ensemble de cardinal $N(N-1)/2$. En effet, on

2. John Napier, plus connu sous son nom francisé Neper, né à Édimbourg en 1550 et mort le 4 avril 1617 au château de Merchiston, est un théologien, physicien, astronome et mathématicien écossais. (Wikipédia)

sait que toute permutation est la composée d'au plus $N-1$ transpositions. En considérant $\mathcal{T}'_N = \mathcal{T}_N \cup \{\mathrm{Id}\}$ qui est de cardinal $N(N-1)/2+1$, on constate que toutes permutations $\sigma$ et $\sigma'$ peuvent être conjuguées par exactement $N-1$ éléments de $\mathcal{T}'_N$ :

$$\sigma' = \tau_{N-1} \circ \cdots \circ \tau_1 \circ \sigma, \qquad \text{avec } \tau_1, \ldots, \tau_{N-1} \in \mathcal{T}'_{N-1}.$$

Ceci suggère de définir la matrice de transition $K$ par

$$K(\sigma, \sigma') = \begin{cases} \dfrac{1}{\operatorname{Card} \mathcal{T}'_N} = \dfrac{1}{N(N-1)/2+1} & \text{si } \sigma' = \tau \circ \sigma \text{ pour } \tau \in \mathcal{T}_N, \text{ ou } \sigma' = \sigma, \\ 0 & \text{sinon.} \end{cases}$$

La matrice $K$ a pour mesure réversible la mesure uniforme $\lambda$ sur $E$ et $K^{N-1}$ a tous ses coefficients strictement positifs. Il ne reste alors qu'à choisir convenablement la constante $C$, puis de lancer une chaîne de Markov $(X_n)_{n\geqslant 0}$ depuis un état quelconque, et enfin attendre en espérant observer assez rapidement des états d'énergie minimale.

Le choix de $K$ dans ce problème n'est peut-être pas le meilleur envisageable. D'une permutation donnée, on peut passer à $N(N-1)/2$ autres permutations ou conserver celle de départ, alors qu'il y a en tout $N!$ permutations. Ceci suggère que la transition à l'aide de $K$ est très locale et qu'il faut beaucoup de temps pour explorer l'ensemble des cas possibles, ou, au moins, un sous-ensemble significatif de celui-ci. De plus, le réglage de la constante $C$ s'avère assez délicat. Elle est très fortement liée au choix de $K$ et peut s'avérer complexe à choisir.

Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 3. — ALGORITHME DE METROPOLIS & CO, ÉLÉMENTS D'EXPLICATIONS

Nous allons traduire le code Scilab en code MetaPost, et commenter au passage les programmes.

Dans la fonction d'initialisation d'origine, ce qui est le plus mystérieux est le calcul des distances. En notant $(x_i, y_i)_{i=1}^n$ les coordonnées des villes, on considère tout d'abord une différence de « produits » matriciels

$$\begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} \otimes (1 \ \dots \ 1 \ \dots \ 1) - \begin{pmatrix} 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{pmatrix} \otimes (x_1 \ \dots \ x_j \ \dots \ x_n)$$

$$= \begin{pmatrix} x_1 & \dots & x_1 & \dots & x_1 \\ \vdots & & \vdots & & \vdots \\ x_i & \dots & x_i & \dots & x_i \\ \vdots & & \vdots & & \vdots \\ x_n & \dots & x_n & \dots & x_n \end{pmatrix} - \begin{pmatrix} x_1 & \dots & x_j & \dots & x_n \\ \vdots & & \vdots & & \vdots \\ x_1 & \dots & x_j & \dots & x_n \\ \vdots & & \vdots & & \vdots \\ x_1 & \dots & x_j & \dots & x_n \end{pmatrix}$$

$$= \begin{pmatrix} 0 & \dots & x_1 - x_j & \dots & x_1 - x_n \\ \vdots & & \vdots & & \vdots \\ x_i - x_1 & \dots & x_i - x_j & \dots & x_i - x_n \\ \vdots & & \vdots & & \vdots \\ x_n - x_1 & \dots & x_n - x_j & \dots & 0 \end{pmatrix}.$$

Ces produits s'apparentent donc à des produits tensoriels. Ensuite, la matrice `dist` est définie comme le carré terme à terme de ce qui précède :

$$\texttt{dist} = \begin{pmatrix} 0 & \dots & (x_1 - x_j)^2 & \dots & (x_1 - x_n)^2 \\ \vdots & & \vdots & & \vdots \\ (x_i - x_1)^2 & \dots & (x_i - x_j)^2 & \dots & (x_i - x_n)^2 \\ \vdots & & \vdots & & \vdots \\ (x_n - x_1)^2 & \dots & (x_n - x_j)^2 & \dots & 0 \end{pmatrix}.$$

On lui rajoute ensuite la matrice correspondant aux secondes coordonnées :

$$\texttt{dist} := \texttt{dist} + \begin{pmatrix} 0 & \dots & (y_1 - y_j)^2 & \dots & (y_1 - y_n)^2 \\ \vdots & & \vdots & & \vdots \\ (y_i - y_1)^2 & \dots & (y_i - y_j)^2 & \dots & (y_i - y_n)^2 \\ \vdots & & \vdots & & \vdots \\ (y_n - y_1)^2 & \dots & (y_n - y_j)^2 & \dots & 0 \end{pmatrix}.$$

Et finalement, `dist` := $\sqrt{\texttt{dist}}$, racine carrée terme à terme, qui est donc bien la matrice des distances euclidiennes entre les différents points. Puisqu'il n'y a pas toutes ces facilités de calcul en MetaPost, on procède plus naïvement.

```
% correspond \‘a ‘‘recuit.sci’’

def initialise(suffix v, dist)(expr nv) =
    % Tire les coordonn\’ees de nv villes au hasard dans
    % le carr\’e unit\’e. Calcule la matrice des distances.
    % Coordonn\’ees des villes :
    pair v[]; v.n := nv;
    for i = 1 upto nv:
        v[i] = (uniformdeviate1, uniformdeviate1);
    endfor
    % Matrice des distances :
    numeric dist[][]; dist.n := nv;
    for i = 1 upto nv:
        dist[i][i] = 0;
        for j = i+1 upto nv:
            dist[i][j] = dist[j][i] = abs(v[i]-v[j]);
        endfor
    endfor
enddef;
```

Pour le tracé du parcours, il n'y a rien de sorcier hormis l'option $-9$ qui implique un tracé point à point avec des cercles.

```
def trace_parcours(suffix v)(expr titre) =
    % Repr\’esente les villes et le circuit dans le carr\’e unit\’e.
    beginfig(thisfig);
        setrange(0, 1, 5cm, 0, 1, 5cm);
        draw for i = 1 upto v.n: gcoord(v[i])-- endfor cycle;
        for i = 1 upto v.n: gdot v[i]; endfor
        gtitle.top(titre, 0);
    endfig;
enddef;
```

Le code d'origine pour le calcul de l'énergie est un peu compliqué...

```
vardef energie(suffix ordre, dist) =
    % fonction d’energie : somme des distances du
    % circuit dans l’ordre donn\’e en entree.
    dist[ordre[dist.n]][ordre[1]]
    for i = 2 upto dist.n: +dist[ordre[i-1]][ordre[i]] endfor
enddef;
```

Les manipulations de vecteurs sont moins efficaces en MetaPost. Le tirage d'une permutation aléatoire est un peu lourd (méthode de type rejet). Notons que le paramètre `nv` est ici supposé contenu dans la définition de `ordre`.

```
def randomperm(suffix o)(expr k) =
    numeric o[]; o.n := k;
    begingroup; save flag, x; boolean flag[];
        for i = 1 upto k: flag[i] = true; endfor
        for i = 1 upto k-1:
```

```
            forever:
                x := min(k, floor(1+uniformdeviate k));
                exitif flag[x];% emplacement libre
            endfor
            o[i] = x; flag[x] := false;
        endfor
        % il reste un unique emplagement libre
        x := 1; forever: exitif flag[x]; x := x+1; endfor
        o[k] = x;
    endgroup;
    % message "{"&decimal(o[1]) for i = 2 upto k: &", "&decimal(o[i]) endfor &"}";
enddef;

vardef permuter(suffix ordre, o) =
    % Permute 5 indices cons\'ecutifs d'un vecteur d'ordre
    save x, oo;
    numeric o[]; o.n := ordre.n;
    x = min(o.n, floor(1+uniformdeviate o.n));% indice de d\'epart
    for i = 1 upto o.n: o[i] = oo[i] = ordre[i]; endfor% ordre initial
    if x > 1:
        for i = 0 upto o.n-x: o[i+1] := oo[x+i]; endfor
        for i = 1 upto x-1: o[i+o.n-x] := oo[i]; endfor
    fi
    % d\'efinir une permutation al\'eatoire sur $\{1,\dots,5\}$
    randomperm(ofive, 5);
    for i = 1 upto 5: oo[i] := o[ofive[i]]; endfor
    for i = 1 upto 5: o[i] := oo[i]; endfor% ordre modifie
enddef;
```

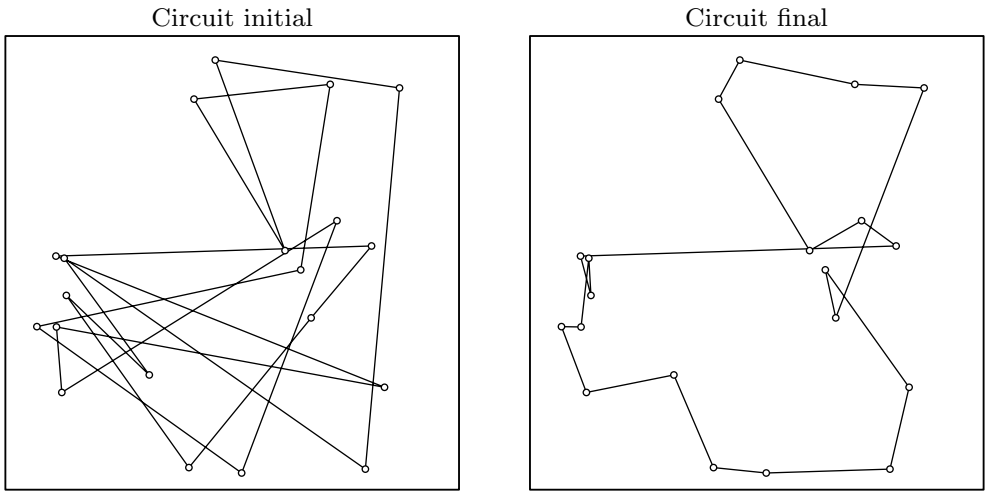
Pour le recuit proprement dit, ça semble aller. Simplement, les conditions d'arrêt sont un peu difficiles à comprendre. La variable de boucle `ut` est l'inverse de la température qui est destinée à croître (donc la température décroît) mais reste constante sur des périodes de temps (ce qu'il se passe dans la boucle `while` ou `forever`), périodes pendant lesquelles le recuit s'effectue à température constante.

```
def recuit(suffix ordre, dist)(expr h, unsurT)(suffix o, e) =
    % Simule nbre pas de l'algorithme de recuit simul\'e
    % pour le probl\'eme du voyageur de commerce.
    % Retourne le nouvel ordre et le vecteur des valeurs
    % successives de la fonction f.
    numeric o[], e[];
    o.n := ordre.n; for i = 1 upto o.n: o[i] = ordre[i]; endfor
    f1 := energie(o, dist);
    e[0] = e.min = e.max = f1;
    n := 0;
    for ut = 1 upto unsurT:
        palier := round(exp(ut*h));
        forever: exitunless n < palier;
            permuter(o, otwo);
            f2 := energie(otwo, dist);
            if f2 < f1:
                for i = 1 upto o.n: o[i] := otwo[i]; endfor
```

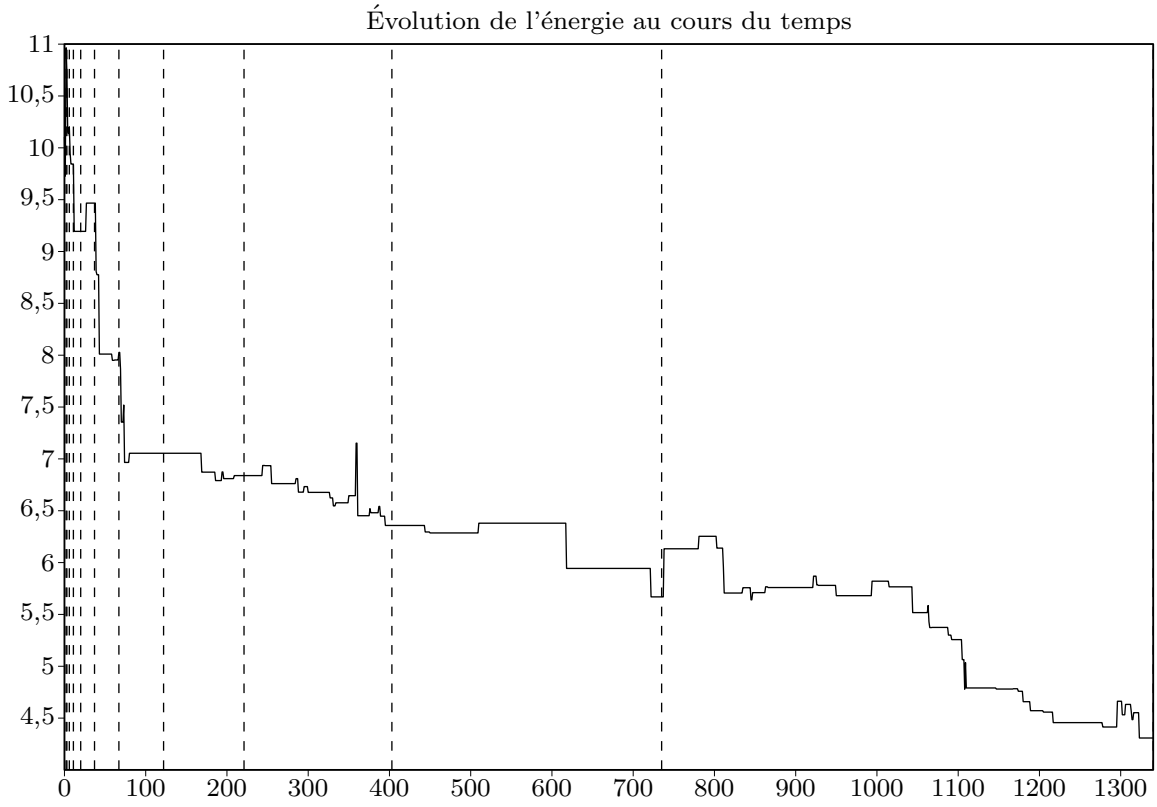
```
                f1 := f2;
            else:
                p := exp((f1-f2)*ut);
                if uniformdeviate1 < p:
                    for i = 1 upto o.n: o[i] := otwo[i]; endfor
                    f1 := f2;
                fi
            fi
            e[incr n] := f1;
            e.min := min(e.min, f1);
            e.max := max(e.max, f1);
        endfor
    endfor
    e.n := n;
enddef;
```

Passons maintenant à l'essai en lui-même. Les paramètres $h$ (pseudo-constante de Planck) et $u$ (inverse de la température minimale) permettent de régler le nombre total de pas et les durées des périodes où la température est constante.

```
% correspond \‘a ‘‘recuit.sce’’

nv := 20;% nombre de villes
initialise(ville, distance, nv);

trace_parcours(ville, btex Circuit initial etex);
h := 0.6; u := 12;% param\‘etres
ordre.n := nv; for i = 1 upto nv: ordre[i] := i; endfor
recuit(ordre, distance, h, u, o, e);% ex\’ecution
pair neuville[]; neuville.n := nv;
for i = 1 upto nv: neuville[i] = ville[o[i]]; endfor
trace_parcours(neuville, btex Circuit final etex);

beginfig(thisfig);
    interim rounding := 10;
    setrange(0, e.n, 12cm, floor(e.min), ceiling(e.max), 8cm);
    draw gcoord(0, e[0]) for i = 1 upto e.n:
        --gcoord(i, e[i]) endfor;% energie
    for ut = 1 upto u:
        palier := round(exp(ut*h));
        draw gcoord(palier, vmin)..gcoord(palier, vmax) dashed evenly;
%        label(blank(catpict(btex palier\ etex, decimal ut)),
%            gcoord(palier, 0.5[vmin, vmax]));
    endfor
    gtitle.top(btex \’Evolution de l’\’energie  au cours du temps etex, 0);
    autograd;
endfig;
```




Les traits verticaux en pointillés délimitent les périodes de temps au cours desquelles la température est constante.



Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 4. — FILTRE DE KALMAN–BUCY

## 1. **Filtrage optimal, position du problème**

Les problèmes de filtrage apparaissent aussi bien en temps discret (Filtrage et Séries temporelles, 3m22) qu'en temps continu (Processus à temps continu, 3m21). Le problème concret est le même : on cherche à estimer un processus $X$, pouvant être déterministe ou non, qu'on ne perçoit qu'à travers un signal bruité $Y$. On veut à partir de $Y$ obtenir une estimation $\widehat{X}$ satisfaisante de $X$.

Pour attaquer ce problème général, il faut faire des hypothèses, c'est-à-dire se donner un modèle : quel est le temps, quels sont les espaces d'états, à quelle classe de processus appartient $X$, comment $Y$ s'exprime-t-il en fonction de $X$ et d'un bruit supplémentaire, en quel sens un processus $\widehat{X}$ sera-t-il considéré comme satisfaisant ?

À chaque modèle, sa, ou ses, solutions. Ce que l'on sait est qu'une solution $\widehat{X}$ doit être $\mathcal{G}$-adaptée, où $\mathcal{G}$ est la filtration engendrée par $Y$, car on se donne pour contrainte que $\widehat{X}$ est construit de manière *progressive* par rapport à $Y$ (penser à un contrôleur aérien suivant un airbus au radar dans un orage).

Une solution évidente est de projeter $X$ sur la filtration $\mathcal{G}$ en posant $\widehat{X}_t = \mathbb{E}[X_t \,|\, \mathcal{G}_t]$ pour tout $t \geqslant 0$. Si les variables sont de carré intégrable, les espérances conditionnelles apparaissent comme des projections orthogonales, et on obtient ainsi *la solution* au sens des moindres carrés.

Oui, mais... Sait-on toujours expliciter ces projections ? Non, évidemment. Mais des solutions explicites existent pour des modèles simples, voire trop simples.

## 2. **Signaux linéaires gaussiens**

Nous nous plaçons en temps discret $\mathbb{T} = \mathbb{N}$, ou bien parce que c'est la nature même du problème, ou bien parce que le problème réel est en temps continu et qu'on discrétise le temps pour l'attaquer sur ordinateur.

Le processus qui nous intéresse est $X = (X_n)_{n\geqslant 0}$ et nous savons qu'il vérifie (c'est dans le modèle)

$$X_n = aX_{n-1} + U_n, \qquad n \geqslant 1,$$

où $U = (U_n)_{n\geqslant 1}$ est un bruit blanc gaussien centré de variance $\sigma^2$ (les $(U_n)_{n\geqslant 0}$ sont des variables aléatoires indépendantes identiquement distribuées de loi $\mathcal{N}(0, \sigma^2)$) que nous supposons indépendant de $X_0$. C'est donc un processus MA(1) dans le langage des séries temporelles. Le processus observé $Y = (Y_n)_{n\geqslant 0}$ est supposé satisfaire

$$Y_n = cX_n + V_n, \qquad n \geqslant 0,$$

où $V = (V_n)_{n\geqslant 1}$ est un bruit blanc gaussien centré de variance $\tau^2$ indépendant de $(X_0, U)$. On a donc le système

$$\begin{cases} X_n = aX_{n-1} + U_n, & n \geqslant 1, \\ Y_n = cX_n + V_n, & n \geqslant 0. \end{cases}$$

Exercice 1. — Écrire une procédure de simulation de paramètres $a$, $c$, $\sigma$, $\tau$, $x_0$ (`Xzero`) et $T$ l'horizon temporel retournant deux vecteurs colonnes $X$ et $Y$ à $T+1$ composantes ($n = 0, \dots, T$). On prendra $a = 1/2$, $c = 1$, $\sigma = \tau = 1$. Des couleurs particulières seront choisies pour chaque type de signal (réel, observé). On préfèrera tracer $Y/c$ plutôt que $Y$ afin que les deux processus soient dans la même échelle (unité, si on préfère).

## 3. **Le filtre de Kalman–Bucy**

On constate que le vecteur aléatoire $(X_0, \dots, X_T, Y_0, \dots, Y_T)$ est un vecteur gaussien : quitte à prendre $X_0$ déterministe ou de loi normale, alors $(X_0, U_1, \dots, U_T, V_0, \dots V_T)$ est un vecteur fait de $T+2$ composantes indépendantes toutes de lois normales et est alors gaussien. Le vecteur précédent est une fonction linéaire de ce dernier (*voir* les calculs mis en place), il est donc gaussien.

Pour $0 \leqslant n \leqslant T$, nous désirons connaître la loi de $X_n$ conditionnellement à $(Y_0, \dots, Y_n)$, sa moyenne conditionnellement à $(Y_0, \dots, Y_n)$ étant précisément l'espérance conditionnelle cherchée. Lorsque $X_0$ est normal centré, on montre le

Résultat. — *La loi de $X_n$ sachant $Y_0, \dots, Y_n$ est la loi normale $\mathcal{N}(\widehat{X}_n, P_n)$ où*

$$\widehat{X}_n = a\widehat{X}_{n-1} + \frac{P_n}{\tau^2}\big(Y_n - a\widehat{X}_{n-1}\big) \qquad \textit{et} \qquad P_n = \frac{a^2\tau^2 P_{n-1} + \sigma^2\tau^2}{P_{n-1} + \sigma^2 + \tau^2}$$

*pour $n \geqslant 1$, et*

$$\widehat{X}_0 = \frac{\sigma^2}{\sigma^2 + \tau^2}\, Y_0 \qquad \textit{et} \qquad P_0 = \frac{\sigma^2\tau^2}{\sigma^2 + \tau^2}\,.$$

*Démonstration.* — La preuve nous importe peu ici. Elle repose uniquement sur le conditionnement dans les vecteurs gaussiens (*voir* le cours de 1m02). Ce qui compte ce sont les coefficients qui sont donnés ici sous une forme récurrente convenant parfaitement à la détermination progressive de $\widehat{X}$.

Exercice 2. — Implémenter le calcul du signal $\widehat{X}$ (`Xhat`). On pourra tracer les différents signaux sur un même graphique.

## 4. **Estimation de certains paramètres du modèle**

L'observateur — celui qui perçoit le signal $Y$ — connaît ou devrait connaître les paramètres $c$ et $\tau$ de ce modèle : ce sont souvent des caractéristiques de son appareil de mesure. En revanche, les paramètres $a$ et $\sigma$ sont ceux du signal $X$ qu'on ne connaît que mal. On peut n'avoir accès qu'à des estimations de ceux-ci. Nous avons le

Résultat. — *Les estimateurs de $a$ et de $\sigma^2$ obtenu par le principe du maximum de vraisemblance sont*

$$\hat{a}_n = \frac{\sum_{k=1}^{n} X_{k-1}X_k}{\sum_{k=1}^{n} X_{k-1}^2} \qquad \textit{et} \qquad \hat{\sigma}_n^2 = \frac{1}{n}\sum_{k=1}^{n}(X_k - \hat{a}_n X_{k-1})^2.$$

*Ils sont de plus fortement consistants et sans biais.*

Exercice 3. — (i) En se servant de l'équation d'évolution de $X$ constater que la forme de ces estimateurs tombe sous le sens.

(ii) Utiliser ces estimateurs pour calculer l'estimation correspondante $\breve{X}$ (`Xcheck`) du processus $\widehat{X}$…

Université de Poitiers
Département de Mathématiques

3M24–Algorithmes stochastiques
Master MMAS 2, année 2009–10

# TRAVAUX PRATIQUES N° 4. — FILTRE DE KALMAN–BUCY, ÉLÉMENTS D'EXPLICATIONS

## 2. **Signaux linéaires gaussiens**

EXERCICE 1. — Rappellons rapidement et incomplètement les notations : le signal initial $X = (X_n)_{n\geqslant 0}$ est un processus MA(1) avec un paramètre de variance $\sigma^2$

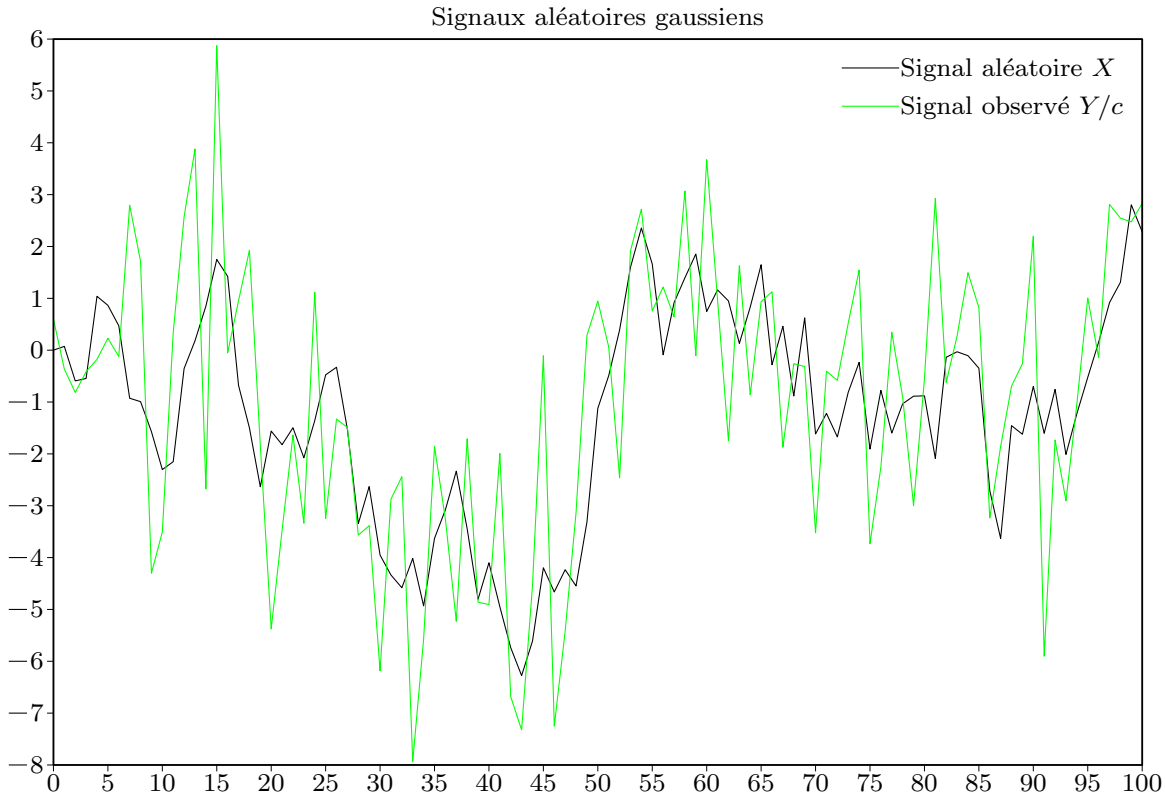
$$X_n = aX_{n-1} + U_n, \qquad n \geqslant 1,$$

le signal observé $Y = (Y_n)_{n\geqslant 0}$ vérifie

$$Y_n = cX_n + V_n, \qquad n \geqslant 1,$$

où le bruit blanc $V$ a une variance $\tau^2$. On simule.

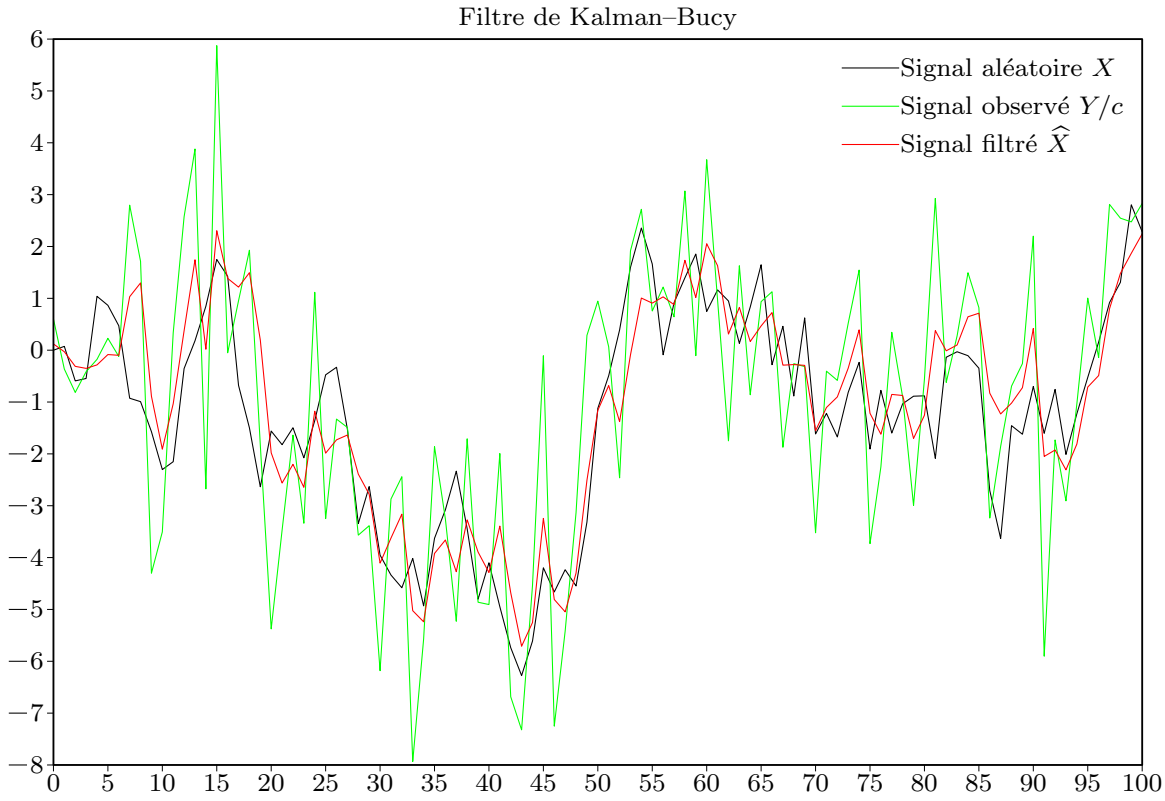
```
// Signaux lin\'eaires gaussiens

function [X, Y] = signals(a, c, sigma, tau, Xzero, T);
    // local U V t;
    U = grand(T, 1, 'nor', 0, sigma);
    V = grand(T+1, 1, 'nor', 0, tau);
    X = [Xzero];
    for t = 1:T;
        X = [X; a*X(t)+U(t)];
    end
    Y = c*X+V;
endfunction

a = 1; c = 1; sigma = 1; tau = 2;

T = 100;  time = [0:T]';
[X, Y] = signals(a, c, sigma, tau, 0, T);

scf(0); clf();
plot2d(time, [X, Y/c], [1, 3]);
xtitle("Signaux lineaires gaussiens");
legends(["Signal aleatoire X"; "Signal observe Y/c"], [1, 3], opt="ur");
```




## 3. **Filtre de Kalman–Bucy**

Exercice 2. — On implémente le calcul du signal $\widehat{X}$ qui est grossièrement le processus des espérances conditionnelles de $X$ prises par rapport à la tribu engendrée par $Y$.

```
// Filtre de Kalman--Bucy

function [Xhat, P] = KalmanBucy(a, c, sigma, tau, X, Y);
    // local t T;
    T = size(X, 1)-1;
    Xhat = sigma^2/(sigma^2+tau^2)*Y(1);
    P = (sigma^2*tau^2)/(sigma^2+tau^2);
    for t = 1:T;
        P = [P; (a^2*tau^2*P(t)+sigma^2*tau^2)/(P(t)+sigma^2+tau^2)];
        Xhat = [Xhat; a*Xhat(t)+P(t+1)/tau^2*(Y(t+1)-a*Xhat(t))];
    end
endfunction

[Xhat, P] = KalmanBucy(a, c, sigma, tau, X, Y);

scf(1); clf();
plot2d(time, [X, Y/c, Xhat], [1, 3, 5]);
xtitle("Filtre de Kalman-Bucy");
legends(["Signal aleatoire X"; "Signal observe Y/c"; "Signal filtre Xhat"], ..
         [1, 3, 5], opt="ur");
```




## 4. **Estimation de certains paramètres du modèle**

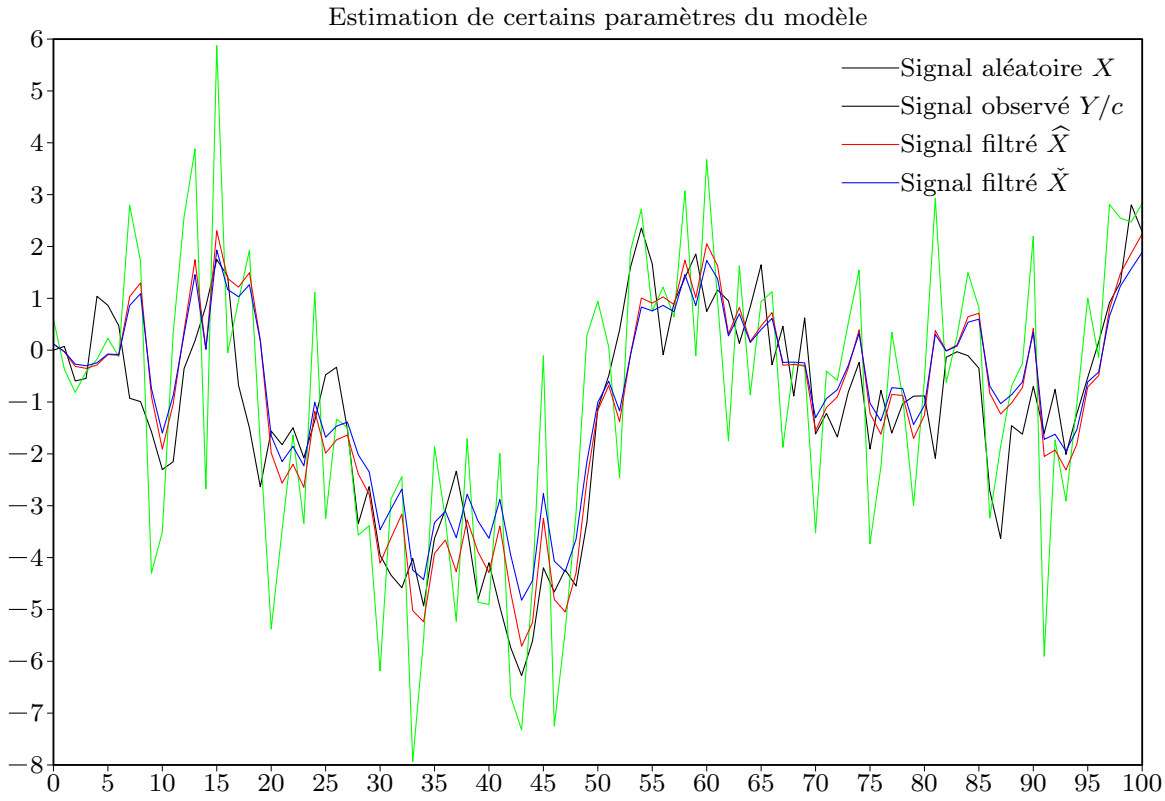
Exercice 3. — (i) Si ça tombe sous le sens...

$$\hat{a}_n = \frac{\sum_{k=1}^{n} X_{k-1}X_k}{\sum_{k=1}^{n} X_{k-1}^2} \qquad \text{et} \qquad \hat{\sigma}_n^2 = \frac{1}{n}\sum_{k=1}^{n}(X_k - \hat{a}_n X_{k-1})^2.$$

(ii)

Le calcul est assez simple.

```
// Estimation de certains param\‘etres du mod\‘ele

function [ahat, sigmahat] = estimation(X);
    T = size(X, 1)-1;
    ahat = (X(1:T)'*X(2:T+1))/(X(1:T)'*X(1:T));
    sigmahat = (X(2:T+1)-ahat*X(1:T))'*(X(2:T+1)-ahat*X(1:T))/T;
endfunction

[ahat, sigmahat] = estimation(X);
[Xcheck, P] = KalmanBucy(ahat, c, sigmahat, tau, X, Y);

scf(2); clf();
plot2d(time, [X, Y/c, Xhat, Xcheck], [1, 3, 5, 2]);
xtitle("Filtre de Kalman-Bucy");
legends(["Signal aleatoire X"; "Signal observe Y/c"; "Signal filtre Xhat"; "Signal fil
         [1, 3, 5, 2], opt="ur");
```




## Code MetaPost

```
% Signaux lin\'eaires gaussiens

vardef signals(expr a, c, sigma, tau, Xzero, T)(suffix X, Y) =
    save U, V; numeric U[], V[], X[], Y[];
    X.n = Y.n = T+1;
    X[1] = Xzero;
    for t = 1 upto T:
        U[t] = sigma*normaldeviate;
        V[t] = tau*normaldeviate;
        X[t+1] = a*X[t]+U[t];
        Y[t] = c*X[t]+V[t];
    endfor
    V[T+1] = tau*normaldeviate;
    Y[T+1] = c*X[T+1]+V[T+1];
enddef;

a := 1; c := 1; sigma := 1; tau := 2;

T = 100;
signals(a, c, sigma, tau, 0, T, X, Y);

beginfig(thisfig);
    x.min = min(X[1], Y[1]/c);
    x.max = max(X[1], Y[1]/c);
    for t = 2 upto T+1:
        x.min := min(x.min, X[t], Y[t]/c);
        x.max := max(x.max, X[t], Y[t]/c);
    endfor
```

```
    setrange(0, T, 12cm, floor x.min, ceiling x.max, 8cm);
    pickup rule.nib;
    draw gcoord(0, X[1]) for t = 1 upto T: --gcoord(t, X[t+1]) endfor;
    addtolegend legendline black, btex Signal al\'eatoire $X$ etex;
    draw gcoord(0, Y[1]/c) for t = 1 upto T: --gcoord(t, Y[t+1]/c) endfor
    withcolor green;
    addtolegend legendline green, btex Signal observ\'e $Y/c$ etex;
    gtitle.top(btex Signaux al\'eatoires gaussiens etex, 0);
    thelegend.urt;
    autograd;
endfig;

% Filtre de Kalman--Bucy

vardef KalmanBucy(expr a, c, sigma, tau)(suffix X, Y, Xhat, P) =
    save T; T = X.n-1;
    numeric Xhat[], P[]; Xhat.n := X.n;
    Xhat[1] = (sigma**2)/((sigma**2)+(tau**2))*Y[1];
    P[1] = ((sigma**2)*(tau**2))/((sigma**2)+(tau**2));
    for t = 1 upto T:
        P[t+1] = ((a**2)*(tau**2)*P[t]+(sigma**2)*(tau**2))
            /(P[t]+(sigma**2)+(tau**2));
        Xhat[t+1] = a*Xhat[t]+P[t+1]/(tau**2)*(Y[t+1]-a*Xhat[t]);
    endfor
enddef;

KalmanBucy(a, c, sigma, tau, X, Y, Xhat, P);

beginfig(thisfig);
    x.min = min(X[1], Y[1]/c, Xhat[1]);
    x.max = max(X[1], Y[1]/c, Xhat[1]);
    for t = 2 upto T+1:
        x.min := min(x.min, X[t], Y[t], Xhat[t]);
        x.max := max(x.max, X[t], Y[t], Xhat[t]);
    endfor
    setrange(0, T, 12cm, floor x.min, ceiling x.max, 8cm);
    pickup rule.nib;
    draw gcoord(0, X[1]) for t = 1 upto T: --gcoord(t, X[t+1]) endfor;
    addtolegend legendline black, btex Signal al\'eatoire $X$ etex;
    draw gcoord(0, Y[1]/c) for t = 1 upto T: --gcoord(t, Y[t+1]/c) endfor
    withcolor green;
    addtolegend legendline green, btex Signal observ\'e $Y/c$ etex;
    draw gcoord(0, Xhat[1]) for t = 1 upto T: --gcoord(t, Xhat[t+1]) endfor
    withcolor red;
    addtolegend legendline red, btex Signal filtr\'e $\widehat X$ etex;
    gtitle.top(btex Filtre de Kalman--Bucy etex, 0);
    thelegend.urt;
    autograd;
endfig;

% Estimation de certains param\‘etres du mod\‘ele
```

```
vardef estimation(suffix X, ahat, sigmahat) =
    save T; T = X.n-1;
    ahat := (0 for t = 1 upto T: +X[t]*X[t+1] endfor)
        /(0 for t = 1 upto T: +X[t]*X[t] endfor);
    sigmahat := 0 for t = 1 upto T: +((X[t+1]-ahat*X[t])**2)/T endfor;
enddef;

estimation(X, ahat, sigmahat);
KalmanBucy(ahat, c, sigmahat, tau, X, Y, Xcheck, P);

beginfig(thisfig);
    x.min = min(X[1], Y[1]/c, Xhat[1], Xcheck[1]);
    x.max = max(X[1], Y[1]/c, Xhat[1], Xcheck[1]);
    for t = 2 upto T+1:
        x.min := min(x.min, X[t], Y[t], Xhat[t], Xcheck[t]);
        x.max := max(x.max, X[t], Y[t], Xhat[t], Xcheck[t]);
    endfor
    setrange(0, T, 12cm, floor x.min, ceiling x.max, 8cm);
    pickup rule.nib;
    draw gcoord(0, X[1]) for t = 1 upto T: --gcoord(t, X[t+1]) endfor;
    addtolegend legendline black, btex Signal al\'eatoire $X$ etex;
    draw gcoord(0, Y[1]/c) for t = 1 upto T: --gcoord(t, Y[t+1]/c) endfor
    withcolor green;
    addtolegend legendline black, btex Signal observ\'e $Y/c$ etex;
    draw gcoord(0, Xhat[1]) for t = 1 upto T: --gcoord(t, Xhat[t+1]) endfor
    withcolor red;
    addtolegend legendline red, btex Signal filtr\'e $\widehat X$ etex;
    draw gcoord(0, Xcheck[1]) for t = 1 upto T: --gcoord(t, Xcheck[t+1]) endfor
    withcolor blue;
    addtolegend legendline blue, btex Signal filtr\'e $\check X$ etex;
    gtitle.top(btex Estimation de certains param\`etres du mod\`ele etex, 0);
    thelegend.urt;
    autograd;
endfig;
```