Loi du premier temps d'occurence d'un motif

Bernard Lapeyre
CERMICS, École des Ponts ParisTech

Mai 2017

Table des matières

1 Propriété de Markov, propriété de Markov forte

Voici deux propriétés proches de la propriété de Markov (simple ou forte).

— pour la "propriété de Markov" : si $(X_1, X_2, \ldots, X_n, \ldots)$ est une suite i.i.d. selon la loi de X_1 alors $(X_{p+1}, X_{p+2}, \ldots, X_{p+n}, \ldots)$ est aussi une suite i.i.d. selon la loi de X_1 indépendante de la tribu $\sigma(X_1, X_2, \ldots, X_p)$.

— pour la "propriété de Markov forte" : si τ est un temps d'arrêt par rapport à la filtration $\mathcal{F}_n = \sigma(X_1, \ldots, X_n)$ alors $(X_{\tau+1}, X_{\tau+2}, \ldots, X_{\tau+n}, \ldots)$ est aussi une suite i.i.d. selon la loi de X_1 indépendante de la tribu $\sigma(X_1, X_2, \ldots, X_\tau)$.

Donnons en quelques applications simples à des tirages indépendants à pile (p) ou face $(1-p)$, $(X_1, X_2, \ldots, X_n, \ldots)$.

Soit $\tau_P = \inf\{n \geq 1, X_n = P\}$. On sait (ou on a oublié ...) que τ_P suit une loi géomètrique de paramètre p. Si on a oublié, voici une façon simple de retrouver la fonction génératrice, on a :

$$\tau_P = \mathbf{1}_{\{X_1 = P\}} \times 1 + \mathbf{1}_{\{X_1 = F\}}(1 + \tau'_P), \tag{1}$$

où τ'_P suit la même loi que τ_P et est indépendante de X_1 (c'est une conséquence de la propriété de Markov). On en déduit que :

$$\mathbf{E}(z^{\tau_P}) = p + (1-p)\mathbf{E}(z^{1+\tau_P}).$$

Ce qui donne la fonction génératrice de τ_P (sans calculs pénibles)

$$\mathbf{E}(z^{\tau_P}) = \frac{p}{1-(1-p)z}.$$

Il est facile de vérifier que c'est bien la fonction génératrice de la loi géométrique.

On obtient aussi, sans calcul, l'espérance de τ_P en prenant l'espérance de l'équation (1) :

$$\mathbf{E}(\tau_P) = p + (1-p)(1+\mathbf{E}(\tau_P)),$$

équation qui se résoud en $\mathbf{E}(\tau_P) = 1/p$. Pour la variance on élève l'équation (1) au carré et l'on prend l'espérance pour obtenir :

$$\mathbf{E}(\tau_P^2) = p + (1-p)(1+2\mathbf{E}(\tau_P)+\mathbf{E}(\tau_P^2)),$$

qui se résoud en $\mathbf{E}(\tau_P^2) = (1+2(1-p)\mathbf{E}(\tau_P))/p$, puis $\mathrm{Var}(\tau_P) = (1-p)/p^2$.

Lorsque $p = 1/2$, on a ainsi $\mathbf{E}(\tau_P) = 2$ et $\mathrm{Var}(\tau_P) = 2$.

Les lois de τ_{PF}, τ_{FP}, τ_{FF}, τ_{PP} On peut aussi calculer les lois de temps d'atteinte de "mots" de longueur 2. Par exemple si $\tau_{PF} = \inf\{n \geq 1, X_{n-1} = P, X_{n-1} = F\}$, il est facile de se convaincre (propriété de Markov forte) que

τ_{PF} est la somme d'une v.a. de loi τ_P et d'une v.a. de loi τ_F indépendantes.

et d'en déduire la moyenne, la variance et la fonction génératrice de τ_{PF} (exercice). On en déduit, par exemple, immédiatement que lorsque $p = 1/2$, $\mathbf{E}(\tau_{PF}) = 2+2 = 4$ et $\mathrm{Var}(\tau_{PF}) = 2+2 = 4$!

Pour calculer la loi de τ_{PP}, c'est à peine plus compliqué, on écrit

$$\begin{aligned}\tau_{PP} = \quad & 2 \times \mathbf{1}_{\{X_1 = P, X_2 = P\}} + \\ & \mathbf{1}_{\{X_1 = F\}} \times (1+\tau_{PP}^{(1)}) + \mathbf{1}_{\{X_1 = P, X_2 = F\}} \times (2+\tau_{PP}^{(2)}),\end{aligned} \tag{2}$$

où $\tau_{PP}^{(1)}$ est indépendant de X_1 et de même loi que τ_{PP} et $\tau_{PP}^{(2)}$ est indépendant de (X_1, X_2) et de même loi que τ_{PP} (propriété de Markov simple). On obtient la fonction génératrice, la moyenne ou la variance par les mêmes techniques que précédemment.

Exercice 1. Effectuer les calculs suggérés précédemment pour calculer la moyenne, la variance et la fonction génératrice de τ_{PP}.

Lorsque $p = 1/2$, vérifier $\mathbf{E}(\tau_{PP}) = 6$ et $\mathrm{Var}(\tau_{PP}) = 22$.

Commentaires : on voit que les lois de τ_{PF} et τ_{PP} sont différentes, alors que les lois de τ_{PP} et τ_{FF} sont identiques ainsi que celles de τ_{PF} et τ_{FP} (par un argument de symétrie).

Exercice 2. Montrer que $\mathbf{P}(X_1 = F, \tau_{PPF} > \tau_{FPP}) = \mathbf{P}(X_1 = F) = 1/2$ (autrement dit, si le premier tirage est un face, on a forcément $\tau_{PPF} > \tau_{FPP}$) et que $\mathbf{P}(X_1 = P, \tau_{PPF} > \tau_{FPP}) > 0$. En déduire que $\mathbf{P}(\tau_{PPF} > \tau_{FPP}) > 1/2$.

Nous allons voir que l'on peut systématiser ce genre de technique pour obtenir la loi des temps d'atteinte de tous les mots et ce pour tous les alphabets.

2 Le cadre général

On recherche un pattern $M = (c_1, \dots, c_n)$, où n est la longueur du pattern, dans un chaine de caractères $(S_i, i \geq 1)$ supposé i.i.d. sur un alphabet fini E avec une loi donnée $q = (q_i, i \in E)$, $q_i > 0$ est supposé non nul pour tout i (sinon il suffit de retirer le caractère de l'alphabet considéré).

On appelle τ le temps d'apparition du pattern dans la chaîne

$$\tau = \inf \{i \geq 0, S_{i-n+1} \dots S_i = M\}.$$

Ce temps est fini presque sûrement, car plus petit que le produit de n fois une v.a. de loi géometrique de probabilité de succés $q(M) := \prod_1^n q(c_i) > 0$. Cela prouve aussi qu'il est de moyenne et de variance finie (et, plus généralement, que tous ses moments sont finis).

On va representer τ à l'aide d'une chaine de Markov dont les états sont $(\mathbf{0}, \mathbf{1}, \dots, \mathbf{n})$. On identifie l'état $\mathbf{j}$ à la sous chaine de M, $(c_1, \dots, c_j)$. Le cas $\mathbf{i} = \mathbf{0}$ correspondant à la chaîne vide. $\mathbf{j}$ represente le début du pattern M de longueur j dont la coïncidence a été déjà vérifiée. Lorsque l'on atteint l'état $\mathbf{n}$ on a trouvé la chaîne M : τ est le temps d'atteinte de $\mathbf{n}$ par la chaîne de Markov.

La matrice de transition Lorsque l'on est dans l'état i, à l'instant l, et que l'on considère le caractère suivant de la chaîne $S(l+1)$

1. soit $S(l+1) = M(i+1)$ et l'on passe à l'état $i+1$
2. sinon, on ne revient pas forcèment à l'état $\mathbf{0}$, mais on doit calculer *le plus long postfixe de* $(c(1), \dots, c(i), S(l+1))$ *qui est un prefixe de* M (le résultat peut être vide, bien sûr, auquel cas on revient en $\mathbf{0}$). Cette chaîne correspond à l'un des états $(0, \dots, l)$ et la transition se fait vers cet état.

Pour l'implémentation, il faudra savoir calculer le plus long postfixe d'une chaîne y qui est un suffixe d'une autre chaîne x : le code `ScicosLab` page 7 fait ce travail.

Remarque 2.1. — Noter que l'on ne revient pas forcement en 0.
- — On a $P(k, l) = 0$, si $l \geq k+2$ et, pour tout $l \geq 1$, $P(l-1, l) > 0$. On utilisera par la suite que $P(n-1, n) > 0$.
- — On peut traiter les deux cas en même temps, puisque dans le cas 1, le plus long prefixe correspond bien à $i+1$. Ceci permet de simplifier la programmation.
- — Lorsque la suite S n'est plus iid mais est markovienne, on peut étendre le modèle markovien mais l'espace d'état de la chaîne doit être augmenté (il faut rajouter le caractère précedent à l'état lorsque la chaine revient en 0). La taille de l'espace d'état est alors $n + \text{card}(E)$ et il faut adapter le calcul de la matrice de transition.

On notera $P(M, E, q)$ la matrice de transition ainsi obtenue et P lorsque le contexte n'exige pas plus de précisions. P est une matrice de taille $(n+1) \times (n+1)$, si n est la longueur du pattern. P est une généralisation de la matrice de la chaîne en "dents de scie".

Voir le code page 7 pour le code qui réalise le calcul de la matrice de la chaîne à partir des données (la chaîne M, l'alphabet E et sa fréquence q).

Calcul de la fonction génératrice de la loi de τ Soit $\phi_k(z) = \mathbf{E}_k\left(z^\tau\right)$, la fonction génératrice de la loi de τ, lorsque la chaîne de Markov démarre en $k \in \{0, 1, \ldots, n\}$. En utilisant la propriété de Markov, on vérifie que, pour tout $k \in \{0, 1, \ldots, n-1\}$

$$\phi_k(z) = \sum_{l=0}^{n} zP(k,l)\phi_l(z). \tag{3}$$

Exercice 3. Démontrer cette équation.

En tenant compte du fait que si $y = n$, $\tau = 0$ et $\phi_n(z) = 1$, on obtient l'équation, pour $x \in \{0, 1, \ldots, n-1\}$

$$\phi_x(z) = \sum_{y=0}^{n-1} zP(x,y)\phi_y(z) + zP(x,n).$$

On peut réécrire matriciellement cette équation sous la forme

$$(I - zQ)\phi_\bullet(z) = zP(\bullet, n), \tag{4}$$

où Q est la matrice $n \times n$, $Q = (P(i,j), 0 \leq i \leq n-1, 0 \leq j \leq n-1)$; $\phi_\bullet(z)$ est le vecteur colonne $(\phi_i(z), 0 \leq i \leq n-1)'$ et $P(\bullet, n)$ le vecteur colonne $(P(i,n), 0 \leq i \leq n-1)'$.

Nous allons voir que cette équation a une solution unique sur le corps de fractions rationnelles d'inconnu z. En effet, si l'on a une solution à $(I - zQ)u(z) = 0$ avec u vecteur de fractions rationnelles, on peut en multipliant $u(z)$ par le produit des dénominateurs du vecteur $u(z)$, obtenir un vecteur de polynôme $v(z)$ solution de $v(z) = zQv(z)$. Supposons un tel vecteur de polynome v de la forme $v(z) = v_0 + v_1 z + \cdots + v_n z^n$ alors $zPv(z) = zQv_0 + z^2 Qv_1 + \cdots + z^{n+1} Qv_n$, et donc par récurrence $v_0 = \cdots = v_n = 0$. $(I - zQ)$ est donc injectif et donc surjectif si on le considère comme une matrice sur le corps des fractions rationnelles. Ce qui prouve l'existence et l'unicité d'une fraction rationnelle solution à l'équation (4). Ceci prouve aussi que $\phi_i(z)$ est une fraction rationelle pour tout $i \in \{0, \ldots, n-1\}$.

Voir le code page 8 pour l'implementation de ce calcul en `Scilab` (qui permet de faire du calcul sur les fractions rationnelles sur $\mathbf{R}$) et page 10 pour le même code en `Sage` (qui permet de faire de l'arithmétique en précision illimitée à la différence de `Scilab`).

3 Calcul des moments

Par dérivation il est facile d'obtenir une équation pour le calcul de la moyenne et de la variance.

Calcul de la moyenne Noter que $\phi_i(z)$ est un fraction rationelle donc peut toujours être dérivée comme fraction rationelle autant de fois que souhaité. Par ailleurs, si $\phi_i(z)$ est représenté sous la forme :

$$\phi_i(z) = \frac{P_i(z)}{Q_i(z)},$$

P_i et Q_i étant deux polynômes sans facteurs communs, $Q_i(1) \neq 0$ (car si $Q_i(1) = 0$, $P_i(1) \neq 0$ et $\phi_i(1) = 1$ est alors impossible). Il est facile d'en déduire que toutes les dérivées de $\phi_i(z)$ calculé en $z = 1$ sont finies. Ce qui prouve que τ a des moments de tout ordre (ce que l'on a déjà obtenu en majorant par un temps géométrique).

On dérive l'équation 1 pour obtenir

$$(I - zQ)\phi'_\bullet(z) = P(\bullet, n) + Q\phi_\bullet(z).$$

où $\phi'_\bullet(z)$ est le vecteur colonne $(\phi'_0(z), \dots, \phi'_{n-1}(z))$. En prenant $z = 1$, on obtient en tenant compte de $\phi_i(1) = 1$

$$(I - Q)\phi'_\bullet(1) = \mathbf{1}.$$

où $\mathbf{1}$ désigne le vecteur colonne $(1, \dots, 1)'$. Ce que l'on peut réécrire (il faut toutefois vérifier que 1 n'est pas valeur propre de Q [1])

$$\mathbf{E}_\bullet(\tau) = (I - Q)^{-1}\mathbf{1}.$$

Calcul de la variance En dérivant une deuxième fois l'équation (1) on obtient, pour $i \in \{0, \dots, n-1\}$

$$(I - zQ)\phi''_\bullet(z) = 2Q\phi'_\bullet(z).$$

où $\phi''_\bullet(z)$ le vecteur colonne $(\phi''_0(z), \dots, \phi''_{n-1}(z))'$. Et en $z = 1$, on obtient

$$[(I - Q)\phi''_\bullet(1)] = 2Q\phi'_\bullet(1) = 2Q(I - Q)^{-1}\mathbf{1},$$

et donc (en tenant compte du fait que Q et $(I - Q)^{-1}$ commutent

$$\mathbf{E}_\bullet(\tau(\tau - 1)) = 2(I - Q)^{-1}\left(Q(I - Q)^{-1}\mathbf{1}\right) = 2(I - Q)^{-2}Q\mathbf{1}.$$

Noter que $(Q\mathbf{1})_i = 1$ pour $i = 0, \dots, n-2$ et que $(Q\mathbf{1})_{n-1} = 1 - P(n-1, n)$.

Pour les programmes qui suivent, on utilise plutôt la forme

$$\mathbf{E}_\bullet(\tau(\tau - 1)) = 2Q\mathbf{E}_\bullet(\tau).$$

Lorsque P est à coefficient rationnel on peut faire des calculs exacts à l'aide d'un logiciel de calcul formel. $(I - P)$ étant souvent difficile à inverser numériquement, ça peut être utile.

Voici page 7 pour le code correspondant en `ScicosLab` ainsi que pour quelques exemples d'utilisation.

4 Vers un algorithme efficace de recherche de pattern (Knuth-Morris-Pratt)

En s'inspirant de l'approche précédente, on peut concevoir un algorithme (proche de l'algorithme KMP) qui permet de rechercher une chaîne de caractère dans un texte en parcourant le texte séquentiellement (c'est à dire en avancant toujours de 1 caractère sans jamais avoir à revenir en arrière ou à sauter en avant dans le texte, ce qui est commode voire indispensable

1. Si 1 est valeur propre de Q, 1 est aussi valeur propre de ${}^tQ$, mais dans ce cas, si u est un vecteur propre de ${}^tQ$,

$$|{}^tQu|_1 = \sum_{i=0}^{n-1}\left|\sum_{j=0}^{n-1} P(j,i)u(j)\right| \leq \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} P(j,i)|u(j)| = \sum_{j=0}^{n-1}(\sum_{i=0}^{n-1} P(j,i))|u(j)|.$$

Comme $\sum_{i=0}^{n-1} P(j,i) = 1$ pour $j = 0, \dots, n-2$ (car $P(j,n) = 0$) et $\sum_{i=0}^{n-1} P(n-1,i) = 1 - P(n-1,n) < 1$, on a $|{}^tPu|_1 < |u|_1$. Ce qui interdit à 1 d'être valeur propre de ${}^tP$ donc de P.

pour certain type de hardware). le programme qui suit n'est pas exactement l'algorithme KMP, mais il en donne l'idée essentielle.

```
function res=search_pattern(M,Text)
  indice_partial_match=0;
  res=0;
  for i = 1:length(Text)
    if part(M,indice_partial_match+1:indice_partial_match+1) == part(Text,i) then
       indice_partial_match=indice_partial_match+1;
       if indice_partial_match == length(M) then
          // printf("position=%d, indice=%d\n",i,indice_partial_match);
          res=i; return;
       end
      else
       proposition=part(M,1:indice_partial_match)+part(Text,i);
       // on calcule le nouvel indice de matching partiel
       indice_partial_match=length(overlap(M,proposition));
    end
    //printf("position=%d, indice=%d\n",i,indice_partial_match);
  end
endfunction
```

Pour rendre ce schéma d'algorithme efficace et obtenir l'algorithme KMP, il faut être plus précis voir par exemple `https://fr.wikipedia.org/wiki/Algorithme_de_Knuth-Morris-Pratt` pour les détails.

5 TP en Scilab

— Ecrire la fonction qui calcule le prefixe de longueur maximale de x qui est un suffixe de y (ou de façon équivalente le suffixe de longueur maximale de y qui est un préfixe de x).

```
function res=overlap(x,y)
  // Calcule un prefixe de longueur maximale de x qui est un suffixe de y
  for i=length(x):-1:1
    //QUESTION: prefixe = le prefixe de x de longueur i
    //REPONSE: prefixe = part(x,1:i);
    //QUESTION: suffixe = le suffixe de j de longueur i
    //REPONSE: suffixe = part(y,max(1,length(y)-i+1):length(y));
    if (prefixe == suffixe) then
      res=prefixe;return;
    end
  end
  res='';
endfunction
```

— Calculer la matrice de transition de la chaîne de markov en fonction du pattern M, de l'alphabet E et de la loi de probabilité de cet alphabet q.

```
function res=inc(n)
// sert à décaler de 1 les indices des matrices commencant en 0
  res=n+1;
endfunction

function Matrice=markov_chain(M, E, q)
// Calcule la matrice de transtion de la chaine de Markov
//   M : la chaîne recherchée,
//   E: l'aphabet,
//   q: la probabité de chaque lettre supposée i.i.d. selon cette loi
  card_alphabet=length(E);

  // Construction de la matrice de transition
  Matrice=zeros(length(M)+1,length(M)+1);
  for i=0:length(M)-1
    for j=1:card_alphabet
      // On rajoute la lettre E(j) à l'état courant
      proposition=part(M,1:i)+part(E,j);
      // on calcule le nouvel etat grâce à la fonction overlap
      to=overlap(M,proposition);
      indice_etat=length(to);// l'indice de l'etat, c'est sa longueur
      // rajout de la probabilité q(j) à la transition de i -> "to"
      //QUESTION: Matrice(inc(i), inc(indice_etat)) = <À COMPLÉTER>
      //REPONSE: Matrice(inc(i), inc(indice_etat)) =  ...
      //REPONSE:             Matrice(inc(i), inc(indice_etat)) + q(j);
    end
  end
  // Lorsque l'on a atteint l'etat "length(M)", c'est gagné, on s'arrête.
  Matrice(inc(length(M)),inc(length(M))) = 1;
endfunction
```

```
function [moyenne,V]=moyenne_variance_tau(M,E,q)
// Calcule la moyenne et la variance de τ
   P = markov_chain(M,E,q);
   N = max(size(P))-1;
   Q = P(1:N,1:N);

   M=eye(N,N) - Q;
   M_moins_1=M^(-1);// calcule l'inverse la matrice M

   // Calcul de la moyenne E(τ)
   moyennes=M_moins_1 * ones(N,1);
   //QUESTION: moyenne=<À COMPLÉTER>;
   //REPONSE: moyenne=moyennes(1,1);

   // Calcul de la variance
   //QUESTION: VV = <À COMPLÉTER>;// Calcul de E(τ(τ − 1))
   //REPONSE: VV = (M^(-1)) * 2 * Q * moyennes;// Calcul de E(τ(τ − 1))
   V=VV(1,1)+moyenne-moyenne^2;// On en déduit la variance.
endfunction;
```

```
// Tirage à Pile  ou Face
alphabet="PF";proba_alphabet=[1/2,1/2];

W='P';
// loi géomètrique on doit trouver 2 et 2
[m,v]=moyenne_variance_tau(W,alphabet, proba_alphabet)

W='FP';
// on doit trouver 4 et 4
[m,v]=moyenne_variance_tau(W,alphabet, proba_alphabet)

W='PP';
// on doit trouver 6 et 22
[m,v]=moyenne_variance_tau(W,alphabet, proba_alphabet)
// Cette loi est différente de la précedente
```

```
// Le génome, 4 bases CATG
alphabet="CATG";proba_alphabet=[1/4,1/4,1/4,1/4];

W='CCTAAGGA'
[m,v]=moyenne_variance_tau(W,alphabet, proba_alphabet)
```

```
// Alphabet de 27 lettres
alphabet="abcdefghijklmnopqrstuvwwyz"+' ';
N=length(alphabet);
proba_alphabet=ones(1,N)/N;
W='bonjour';
[m,v]=moyenne_variance_tau(W,alphabet, proba_alphabet)
```

```
function phi=fonction_generatrice(M,E,q)
// Calcule la fonction génératrice de la loi de τ
// P est la matrice de transition de taille N+1×N+1
  P=markov_chain(M, E, q);
  N=max(size(P))-1;

  // On resoud l'equation dans le corps des fractions rationnelles
  Q=P(1:N,1:N);
  Id=eye(N,N);
  z=poly(0,"z");
  M=Id - z * Q;
  A=(1/M) * z * P(1:N,N+1);// 1/M calcule l'inverse la matrice M
  phi=A(1,1);
endfunction;
```

6 Listing Sage

Malheureusement, on ne peut traiter que des chaînes de caractères longueur petite (≈ 10) si l'on se contente de l'arithmétique en précision limitée standard. Pour traiter des chaînes plus longues il faut faire du calcul en précision illimitée dans $\mathbf{Q}$. Voici une implementation en `sage` (on peut télécharger librement `sage` pour linux, macos ou windows sur le site `http://www.sagemath.org`) pour un cas où M est plus long (longueur $l = 43$, on néglige accents et majuscules mais on rajoute le `' '` à l'alphabet, de 27 lettres donc).

`"Longtemps je me suis couché de bonne heure".`

Cette chaîne est plutôt simple, puisque, lorsque $l > 0$ soit on passe de l à $l+1$ avec proba $1/27$, on retourne en 1 avec proba $1/27$ (cas d'échec mais avec comme nouvelle lettre `'l'`), sinon on retourne en 0 avec proba $25/27$. Le cas $l = 0$ est particulier, la probabilité de rester en 0 est de $26/27$ et d'aller en 1 de $1/27$.

```
def overlap(x,y):
# Calcule le plus long prefixe
# de x qui est un suffixe de y
    l_y=len(y)
    i=len(x)
    while ((i>0) and (x[0:i] <> y[max(0,l_y-i):l_y])):
        i=i-1
    return x[0:i]
```

```
def markov_chain(M, E, q):
# Calcule la matrice de transition de la chaine de Markov
    card_alphabet=len(E);
    N=len(M);
    MS=MatrixSpace(QQ,N+1,N+1)
    Matrice=MS()
    for i in [0..N-1]:
        for j in [0..card_alphabet-1]:
            # On rajoute la lettre E(j) a l'etat de la chaine
            proposition=M[0:i]+E[j:j+1];
            # on calcule le nouvel etat grace a la fonction overlap
            to=overlap(M,proposition);
            indice_etat=len(to)  # l'indice de l'etat, c'est sa longueur
            # on rajoute la probabilite de alphabet(j)
            # a la  transition de i vers "to"
            Matrice[i, indice_etat] =  Matrice[i,indice_etat] + q[j];
    # Lorsque l'on a atteint l'etat "M=len(M)", c'est gagne, on s'arrete.
    Matrice[N,N] = 1
    return Matrice
```

```
def moyenne_variance(M, E, q):
    # calcul la moyenne et la variance du temps d'atteinte de N
    # par la chaine partant de 0 de matrice de transition P
    P=markov_chain(M, E, q)
    N=P.nrows()
    Q=P[0:N-1,0:N-1]
    #
    MS=MatrixSpace(QQ,N-1,N-1)
    Id=MS.one()
    M=Id-Q;
    #
    VS=VectorSpace(QQ,N-1)
    v=VS()
    for i in [0..N-2]: v[i]=1
    A=M.solve_right(v)
    moyenne=A[0]
    #
    vv=2*Q*A
    B=M.solve_right(vv)
    variance=B[0]+moyenne-moyenne*moyenne
    return [moyenne, variance]
```

```
load("overlap.sage")
load("markov.sage")
load("gene.sage")
load("moments.sage")

# tirages à pile ou face
E='PF'
q=[1/2, 1/2]
moyenne_variance('P', E, q)
moyenne_variance('PF', E, q)
moyenne_variance('PP', E, q)
moyenne_variance('PFPFPFPFPF', E, q)

# alphabet classique
E='abcdefghijklmnopqrstuvwxyz'+' '
M='bonjour et bonjour'
N=len(M)
size=len(E);
VS=VectorSpace(QQ,size);
q=VS();
for i in [0..size-1]: q[i]=1/size;
[moyenne,variance]=moyenne_variance(M, E, q)
```

```
# Un exemple de très grande taille : 1000 caractères
def log10(x):
    return ln(x)/ln(10)

N=1000
p=1/27
# Construit la matrice de transition d'une chaine generique qui :
#    -> 0 avec proba 1-2*p, -> 1 avec proba p, n -> n+1 avec proba p
E='PFQ';q=[p,p,1-2*p];
# P suivi de (N-1) F
M='P';
for i in [1..N-1]: M=M+'F'
[moyenne,variance]=moyenne_variance(M, E, q)
log10(moyenne).n(digits=10)
# Esp ≈ 10^1431
# que l'on peut comparer à la loi geometrique majorante
pp=(1/27)^N
moy_geom=N/pp
log10(moy_geom).n(digits=10)
# Esp ≈ 10^1434
```

```
def fonction_gene(M, E, q):
    # calcule la fonction generatrice
    # qui est de la forme z^N/Pol(z)

    P=markov_chain(M, E, q)
    N=P.nrows()
    Q=P[0:N-1,0:N-1]
    R.<z>=QQ['z']
    MS=MatrixSpace(R,N-1,N-1)
    Id=MS.one()
    v=P[0:N-1,N-1]
    M=Id - z * Q
    A=M.solve_right(z * v)
    # return A[0,0]
    temp=z^(N-1) / A[0,0]
    return z^(N-1) / temp
```