Feuille de TP n°1 – Initiation à Matlab

Ce TP porte sur les entrées et sorties, les fonctions et les outils graphiques dont vous disposez sous Matlab.

1 Entrées et sorties

La commande `input` permet de demander à l'utilisateur Matlab d'entrer les valeurs de variables à utiliser. La commande `pause` permet de stopper l'exécution Matlab. Vous pouvez préciser le nombre de secondes de pose ou revenir à Matlab en appuyant sur n'importe quelle touche. La commande `save` permet de sauvegarder dans un fichier, dont le nom par défaut est `matlab.mat`, le contenu de certaines variables dont vous souhaitez garder une trace. Ce fichier peut être appelé par la commande `load` qui restaure toutes les variables que vous avez sauvegardées.

```
n=input('Entrez la valeur de n : ');    % Affectez une valeur à n.
a=input('Precisez la valeur de a : '); % Affectez une valeur à a.
v=a.^[0:n]; A=toeplitz(v); d=det(A);  % Création de la matrice de Toeplitz A
save restoep n a A d; % Sauvegarde de n, a, A, d dans restoep.mat
clear                 % Efface toutes les variables de la session.
load restoep          % Restaure les variables de restoep.mat.
who                   % Vérification.
```

2 Fonctions

Un ensemble de commandes Matlab peut être considéré comme une fonction. On peut voir une fonction comme un sous-programme Matlab dont les paramètres éventuels sont les arguments de la fonction et dont les résultats sont les images de cette fonction. Beaucoup de fonctions Matlab, comme par exemple `mean`, sont déjà écrites en Matlab et le code Matlab correspondant est stocké dans un fichier dont le nom se termine par `.m`. Pour `mean`, il s'agit de `mean.m`. Ajouter de nouvelles fonctions à Matlab revient donc à écrire de nouveaux fichiers de ce type. Il est d'usage d'appeler une fonction du même nom que le fichier correspondant.

Simulation de lois discrètes. Dans votre répertoire personnel, éditer le fichier `probadis.m` suivant dont le code Matlab génère une réalisation aléatoire d'une loi discrète à support fini.

```
function res = probadis(x,p)
%res = PROBADIS(x,p)
%        Input    x     vector of real numbers (support points in IR)
%                 p     vector of probability weights associated to x
%                       i.e. non-negative real numbers such that sum(p) == 1
%        Output   res   random number chosen from the finite discrete
%                       distribution on x(1),...,x(n) with probability weights
%                       p(1),...,p(n) where n == length(x) == length(p)
%
% Renvoie UNE réalisation de la loi discrète à support fini sur IR dont les
% points de support sont les composantes du vecteur x, et les poids sont les
% composantes du vecteur p. Donc x(i) a une probabilité p(i) d'être renvoyé.
%
% La méthode consiste à écrire l'intervalle [0,1] comme une réunion disjointe
% d'intervalles de longueurs p(1),..., p(n) puis à regarder au quel appartient
```

```
% la réalisation d'une loi uniforme obtenue par un appel à rand. Les valeurs
% renvoyées par des appels successifs sont donc pseudo-indépendantes.
% Pour des raisons d'efficacité, les conditions nécessaires suivantes ne sont
% pas contrôlées par cette fonction :
%  - le nombre de paramètres passés est exactement 2
%  - les deux paramètres x et p sont bien des vecteurs et sont de même longueur
%  - les composantes de p sont positives ou nulles et leur somme vaut 1.
%
% See also RDISCR.
%
% ### Copyright (C) D. Chafaï, 2003-12-06.
% ### http://www.lsp.ups-tlse.fr/Chafai/agregation.html
% ### Licence GNU General Public License http://www.gnu.org/copyleft/gpl.html
%

% On pourrait implémenter cette fonction de la façon suivante :
%
%  INDICES = find(cumsum(p) >= rand);
%  res = x(INDICES(1));
%  return;
%
% Cette méthode est correcte mais inefficace car elle ne tient pas compte de
% la monotonie de cumsum(p), ce qui entraine des tests inutiles une fois que
% la valeur critique a été franchie. En jargon, c'est un 'firstmatch' qu'il
% nous faut, pas un 'matchall'. Le nombre de if impliqués dans find est
% toujours égal à la taille de ce qu'on lui passe en paramètre.
%
% Morale de l'histoire : la brièveté d'un code n'assure pas sa performance !
% Et l'abscence de if dans un code ne signifie pas qu'il ne fait pas appel
% indirectement à des if, et encore moins que cela est fait de manière
% optimale ! La fonction find n'est qu'une boucle for contenant un if.
%
% Ci-dessous, nous utilisons une version plus rapide dans le plus pur style
% for-if avec un nombre de if optimal. On pourrait adapter l'ordre des
% intervalles testés (et donc l'arbre associéé) aux poids p(i) de façon à
% tester d'abord les intervalles les plus probables. Est-ce vraiment mieux ?
% Exercice !

n = length(x);      % le nombre d'atomes.
r = rand;           % une réalisation de loi uniforme sur [0,1].
a = 0; b = p(1);    % [a,b] = sous-intervalle de proba p(i) pour l'uniforme.
for i = 1:n-1       % parcours de tous les sous-intervalles juxtaposés.
   if ((r >= a) & (r < b))
      res = x(i);
      return;       % on a trouvé le bon intervalle, on sort.
   end
   a = b; b = b + p(i+1); % on passe à l'intervalle suivant.
end
res = x(n); % le bon intervalle est le dernier.
return;
```

La commande Matlab `type` permet de lister le contenu d'un fichier. Ainsi, `type probadis` vous montrera le code source Matlab de la fonction `probadis`. Le commentaire ajouté à partir de la seconde ligne constituera l'aide affiché lorsque l'utilisateur tapera `help probadis`. Finalement, la commande `what` liste les fichiers Matlab du répertoire courant.

Voici une autre fonction de simulation de loi discrète, qui peut renvoyer une matrice de réalisations.

```
function X = rdiscr(num,x,p)
%X = RDISCR(num,x,p)
%        Input    num    positive integer or a vector [lig,col] of integers
%                 x      vector of real numbers (support points in IR)
%                 p      vector of probability weights associated to x
%                        i.e. non-negative real numbers such that sum(p) == 1
%        Output   X      num-vector or a num-matrix of random numbers
%                        chosen from the finite discrete distribution on
%                        x(1),...,x(n) with probability weights p(1),...,p(n)
%                        where n == length(x) == length(p)
%
% Renvoie num réalisations ou une matrice [lig,col] de réalisations
% pseudo-i.i.d de la loi discrète à support fini sur IR dont les
% points de support sont les composantes du vecteur x, et les poids sont les
% composantes du vecteur p. Donc x(i) a une probabilité p(i) d'être renvoyé.
%
% La méthode consiste à écrire l'intervalle [0,1] comme une réunion disjointe
% d'intervalles de longueurs p(1),..., p(n) puis à regarder au quel appartient
% la réalisation d'une loi uniforme obtenue par un appel à rand. Les valeurs
% renvoyées par des appels successifs sont donc pseudo-indépendantes.
% Pour des raisons d'efficacité, les conditions nécessaires suivantes ne sont
% pas contrôlées par cette fonction :
%  - le nombre de paramètres passés est exactement 3
%  - les deux paramètres x et p sont bien des vecteurs et sont de même longueur
%  - les composantes de p sont positives ou nulles et leur somme vaut 1
%  - le paramètre num est un entier positif non nul une un couple de ce type
%
% See also PROBADIS.
%
% ### Copyright (C) D. Chafaï, 2003-12-06.
% ### http://www.lsp.ups-tlse.fr/Chafai/agregation.html
% ### Licence GNU General Public License http://www.gnu.org/copyleft/gpl.html
%

% Voir les commentaires dans le code de la fonction probadis.
% Faire 'type probadis' pour cela.
% Le code qui suit pourrait beaucoup gagner en rapidité sur une machine //
% Il est possible de l'améliorer en stockant un arbre construit avec les
% tests utilisés par les valeurs déjà générées. Il est surtout aussi possible
% d'imiter le code matriciel de la fonction de répartition inverse binomiale
% de la fonction qbinom de Stixbox, appelée par rbinom. Exercice !
%
% D'autres algorithmes sont possibles. Par exemple, on pourrait procéder
% avec des réalisations de Bernoulli i.i.d. (obtenues facilement avec rand)
% pour choisir l'intervalle :
% La probabilité d'être <= x(1) est p(1)
% Sinon, la probabilité d'être <= x(2) est p(2)/sum(p(2:n))
% etc. Nul besoin de commencer par x(1), et l'on peut adapter l'arbre utilisé
% aux poids p(i) de façon à faire un nombre de tests optimal. Exercice !
```

```
if length(num) == 1
   num = [num 1];
else
   num = reshape(num,1,2);
end

n = length(x);     % le nombre d'atomes.
U = rand(num);     % réalisations de loi uniforme sur [0,1].
X = repmat(x(n),num(1),num(2)); % par défaut, la valeur est la plus grande
for l = 1:num(1) % lignes
for c = 1:num(2) % colonnes
        a = 0; b = p(1);  % [a,b] = sous-inter. de proba p(i) pour l'uniforme.
        for i = 1:n-1 % parcours de tous les sous-intervalles juxtaposés.
                if ((U(l,c) >= a) & (U(l,c) < b))
                        X(l,c) = x(i);
                        break; % on a trouvé le bon intervalle, on sort.
                end
                a = b; b = b + p(i+1); % on passe à l'intervalle suivant.
        end
end
end
return;
```

Exercice 2.1 (Loi binomiale). Créer un code Matlab permettant de générer un vecteur aléatoire X contenant N réalisations i.i.d. de loi binomiale $\mathcal{B}(n,p)$ où les valeurs $N, n \geqslant 1$ et $0 < p < 1$ sont affectées par l'utilisateur. Pour N assez grand, vérifier la LGN sur les moyennes empiriques successives de X. Voici un exemple de programme qui fait l'affaire, et donc la sortie graphique se trouve en page 6.

```
%
% ### Copyright (C) D. Chafaï, 2003-12-06.
% ### http://www.lsp.ups-tlse.fr/Chafai/agregation.html
% ### Licence GNU General Public License http://www.gnu.org/copyleft/gpl.html
%
% Ce bout de code permet de simuler une loi binomiale B(n,p) et de comparer
% graphiquement les moyennes empiriques avec la moyenne théorique.
% Tous les vecteurs sont des vecteurs ligne.
%
% Nota bene : la bibliothèque Stixbox fournit la fonction rbinom qui permet
% de s'affranchir des calculs détaillés ici, cf. fin du présent fichier.
%
clear r n p q P Q C B X
clf
%
r = input('Nombre maximal de réalisations ? = ');
n = input('Taille n de la loi binomiale B(n,p), qui a n+1 atomes ? = ');
p = input('Valeur du paramètre p de la loi binomiale B(n,p) ? = ');
% p = proba de gagner à pile ou face = proba de 1 dans la Bernoulli sur {0,1}
% q = proba de perdre à pile ou face = proba de 0 dans la Bernoulli sur {0,1}
% B(n,p) est la loi de la somme de n v.a. i.i.d. de Bernoulli de ce type,
% i.e. la nième puissance de convolution de cette loi de Bernoulli. Elle
% représente la loi du nombre de succés à pile ou face en n lancés.
%
disp(sprintf('Calcul de la loi binomiale B(%d,%f)',n,p))
```

```
q = 1 - p;
P = [1 , cumprod(p * ones(1,n))];                % puissances croissantes de p
Q = [fliplr(cumprod(q * ones(1,n))) , 1];        % puissances décroissantes de q
C = [1 , cumprod([n:-1:1]) ./ cumprod([1:n])];   % coef. du binôme de i = 1 à n
B = C .* P .* Q;                                 % vecteur des poids de B(n,p)
%
disp(sprintf('Génération aléatoire de %d réalisations de B(%d,%f)',r,n,p))
X = rdiscr([1,r],[0:n],B);                       % échantillonnage
% alternative :
%X=[]
%for i=1:r
%       X = [X probadis([0:n],B)];
%end
%
disp(sprintf('Tracé des graphique.'))
plot(cumsum(X) ./ [1:length(X)], 'b')            % tracé des moyennes empiriques
title('Loi des Grands Nombres')
xlabel('Nombre de realisations')
ylabel('Moyennes empiriques')
hold on
plot(n * p * ones(1,r), 'r--')                   % tracé de la moyenne théorique
legend('Empirique','Theorique')
hold off
%
% Avec Stixbox, inutile de calculer B et d'appeler rdiscr([1,r],[0:n],B)
% puisque qu'un simple rbinom([1,r],n,p) suffit.
% Même si l'on décide d'utiliser quand même prodadis, les coefficients
% binomiaux nécessaires au calcul de B peuvent se calculer beaucoup plus vite
% en utilisant la fonction Stixbox bincoef qui fait appel à la fonction gamma.
%
```

Exercice 2.2. Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires i.i.d. de loi exponentielle $\mathcal{E}(\lambda)$ avec $\lambda > 0$. Si $S_n = \sum_{k=1}^{n} X_k$, $N_0 = 0$ et pour tout $t > 0$, $N_t = \sum_{n=1}^{\infty} \mathrm{I}_{\{S_n \leqslant t\}}$, $(N_t)_{t\geqslant 0}$ est un processus de Poisson d'intensité λ. Montrer que, pour tout $t > 0$, N_t suit la loi de Poisson $\mathcal{P}(\lambda t)$. En déduire un code Matlab permettant de générer un vecteur aléatoire Y contenant N réalisations i.i.d. de loi $\mathcal{P}(\lambda)$ où les valeurs $N \geqslant 1$ et $\lambda > 0$ sont affectées par l'utilisateur. Pour N assez grand, vérifier la LGN sur les moyennes empiriques successives de Y.

Exercice 2.3. Pour N, N_1 et $n \geqslant 1$ avec $N_1, n \leqslant N$, la loi hypergéométrique $\mathcal{H}(N, N_1, n)$ est donnée, pour tout $k \in \mathbb{N}$ avec $0 \leqslant k \leqslant n$, par $\mathbb{P}(X = k) = C_{N_1}^{k} C_{N-N_1}^{n-k} / C_N^n$. Créer un code Matlab permettant de générer un vecteur aléatoire Z contenant M réalisations i.i.d. de loi $\mathcal{H}(N, N_1, n)$ où les valeurs $M, N \geqslant 1$ et $N_1, n \leqslant N$ sont affectées par l'utilisateur.

1. Si N tend vers l'infini et le rapport N_1/N tend vers p avec $0 < p < 1$, montrer que X converge en loi vers la loi Binomiale $\mathcal{B}(n,p)$. Pour M, N assez grand et $N_1 = pN$ avec $0 < p < 1$, tracer l'histogramme de Z et comparer le à la loi $\mathcal{B}(n,p)$.
2. Si N, N_1 et n tendent vers l'infini et le produit nN_1/N tend vers $\lambda > 0$, montrer que X converge en loi vers la loi de Poisson $\mathcal{P}(\lambda)$. Pour M, N assez grand, $N_1 = \lambda\sqrt{N}$ et $n = \sqrt{N}$, tracer l'histogramme de Z et comparer le à la loi $\mathcal{P}(\lambda)$.

3 Représentations graphiques

```
{Illustration de la LGN pour la loi exponentielle}
 clear; n=1000; lambda=0.5; X=-log(rand(n,1))/lambda;
 figure;                                 % Création d'une nouvelle fenêtre graphique.
 plot(cumsum(X)'./[1:length(X)],'b')     % Trace les moy. emp. successives de X.
 title('Loi des Grands Nombres')         % Titre de la figure.
 xlabel('Nombre de réalisations')        % Titre des abscisses.
 ylabel('Moyennes empiriques')           % Titre des ordonnées.
 hold on                                 % Garde la fenêtre graphique.
 plot(1/lambda*ones(n,1),'r--');         % Trace la limite théorique.
 legend('Empirique','Theorique');        % Légende.
```

Exercice 3.1. Ajouter à vos codes Matlab les représentations graphiques rencontrées ci-dessus.

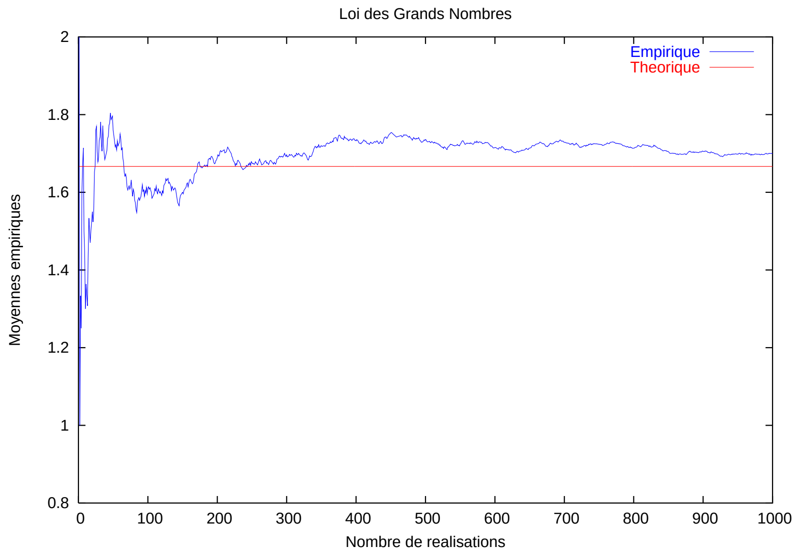

Fig. 1 – Sortie graphique du programme 2 5.

Voici une fonction Matlab qui permet de simuler la loi uniforme discrète finie de façon rapide et simple.

```
function X = rint(num,k);
% X = RINT(num,k)
%        Input   num     positive integer or a vector [lig,col] of integers
%                k       positive integer
%        Output  X       num-vector or a num-matrix of random numbers
%                        chosen from uniform distribution on {1,...,k}
%
% Renvoie num réalisations ou une matrice [lig,col] de réalisations
% pseudo-indépendantes de la loi uniforme discrète sur les k premiers
% entiers non nuls {1,...,k}.
%
% La méthode consiste à considérer la partie entière de 1+kU où U suit une loi
```

```
% uniforme sur [0,1]. Cette dernière s'obtient via la fonction rand, et les
% valeurs renvoyées par des appels successifs sont donc pseudo-indépendantes.
%
% ### Copyright (C) D. Chafaï, 2003-12-06.
% ### http://www.lsp.ups-tlse.fr/Chafai/agregation.html
% ### Licence GNU General Public License http://www.gnu.org/copyleft/gpl.html
%

if length(num) == 1
  num = [num 1];
end
X = ceil(k * rand(num));
return;
```

Références

[BL98] Ph. Barbe and M. Ledoux, *Probabilités*, De la licence à l'agrégation, Belin, 1998.

[Bou86] N. Bouleau, *Probabilités de l'ingénieur, variables aléatoires et simulation*, Hermann, 1986.

[DCD82] D. Dacunha-Castelle and M. Duflo, *Probabilités et statistiques. Tome 1*, Masson, Paris, 1982, Problèmes à temps fixe.

[Yca02] B. Ycart, *Modèles et Algorithmes Markoviens*, Mathématiques et Applications, vol. 39, Springer, 2002.