Feuille de travaux pratiques # 1

Préambule :

- Scilab est un logiciel libre que l'on peut télécharger à l'adresse http://www.scilab.org.
- Scilab possède une aide en ligne consultable en tapant `help` dans la console. Plus généralement pour obtenir une aide concernant une fonction scilab `bidule`, tapez `help bidule`.
- Pour vous familiariser avec les fonctions de base de Scilab, vous trouverez de nombreux tutoriels sur la toile, par exemple ici et là.

1 Échauffement

1.1 Opération sur les matrices

Quelques opérations basiques sur les matrices à tester/connaître :

- Fabrication manuelle d'une matrice `M=[1 3 5;-2 6 1];` ou `M=[1,3,5;-2,6,1];`
- Extraction de la première ligne ou de la troisième colonne : `l = M(1,:); c = M(:,3);`
- Transposée de la matrice `N=M' ;`
- Fabrication automatique d'une discrétisation d'un intervalle : `[0 : 0.1 : 2];`
- Fabrication automatique d'une matrice de 0 ou de 1 : `P = zeros(4, 5) ; Q = ones(4,5);`
- Suppression de la deuxième colonne : `P(:,2) = [];`
- Fabrication d'une matrice par blocs : `x = [1 : 2]; P = [x 2*x; 3* x 4*x];`
- Que calcule la commande `s= x'*ones(x);` ?
- Si `x` et `y` sont deux vecteurs (colonne), comment calculer leur produit scalaire ?
- D'autres commandes à tester : `sum(x), mean(x), x.^2, M.^2, M^2` (à essayer avec une matrice carrée)

1.2 Opération sur les booléens

Quelques opérations booléennes à tester/connaître :

- `A=[1 2 5 4 ; 3 6 7 8] ; sum(A)>3 ; l=(M>2); k=(M==5);`
- Transformer True/False en 1/0 : `k=bool2s(l);`
- Le ou logique : `O=((A>5)|(A<3));`
- Le et logique : `E=((A>1)&(A<3));`
- Expliquer la commande suivante : `x=[-2 :.5 :2]; l=(x>-1 & x<1); x(l)=3.14;`
- Qu'effectue la commande suivante : `B =bool2s(A<3) ; C=bool2s(A>6); D=bool2s((B+C)>0);`
- Lister des éléments `find(A>4) ; [u,v]=find(A>5) ;`
- Trier des éléments : `X=[ 1 5 4 9 2 ; 12 7 3 5 8] ; Y=gsort(X) ; Z=gsort(X,'g','i') ; T=gsort(X,'c','i') ;`

1.3 Boucles, opérateurs logiques et fonctions

Lorsque les listes de commandes s'allongent, il est recommandé de les regrouper au sein d'une fonction ou d'un petit programme. Par exemple, dans une nouvelle fenêtre de l'éditeur ad-hoc, saisissez les lignes suivantes :

```
function y=mafonction(x)
y=x^2;
endfunction;
```

Pour faire appel à la fonction, sauvez / executez le script dans l'éditeur, puis dans la ligne de commande tapez `mafonction(-3)`. Vous pouvez faire de même avec une fonction plus compliquée :

```
function [y1,y2]=mafonction(x1,x2)
y1=2*x2; //En plus je peux mettre des commentaires
y2=y1+x1; //pour me rappeler le sens de chaque ligne
endfunction;
```

Pourquoi n'a-t-on pas le même résultat en tapant `mafonction(2,3)` et `[u,v]=mafonction(2,3)` ?

Les opérateurs logiques `et` et `ou` sont codés comme suit :

```
x==0 ;      (x<=4)&(x>2) ;      (x<=4)|(x>2);
```

Les syntaxes des boucles `for`, `while` et les instructions conditionnelles sont les suivantes :

```
i = 0; aux=2;
while (i<5)&(aux<10)
   aux=aux*aux;
   i = i + 1;
end
```

```
n=5;
for i = 1:n
   a(i)=1/(i+1);
   a(i)=1/(i+1);
end
```

```
if i == j then
   a(i,j) = 2;
else
   a(i,j) = 0;
end,
```

Application : Encoder la fonction factorielle. Vaut-il mieux utiliser une boucle `for` ou une boucle `while` ?

Pour vous entraîner à manipuler ces commandes, quelques fonctions/algorithmes simples à essayer : somme des inverses des carrés d'entiers jusqu'à n, crible d'Ératostène pour les premiers inférieurs à n, etc.

2 Générer des variables aléatoires avec Scilab

2.1 Fonctions prédéfinies

La fonction la plus simple pour générer des variables aléatoires avec Scilab s'appelle `rand`. Elle permet de simuler des nombres pseudo-aléatoires distribués selon la loi uniforme sur $[0, 1]$. La fonction `grand` permet de simuler des échantillons suivant toutes les lois de probabilité classiques (on renvoie à l'aide en ligne pour plus de précisions).

1. Tester les commandes `rand(1,1)`, `rand(2,4)`, `grand(3,1,'nor',1,2)` ;
2. Que font les commandes suivantes :
   a. `plot2d(cumsum(rand(1,1000))./[1:1000])` ?
   b. `plot2d([1 :1000],cumsum(rand(1,1000))./[1:1000],4)` ?

c. `clf; plot2d(cumsum(-log(rand(1,1000)))./[1:1000])` ?

d. `plot2d([1 :100]',cumsum(grand(100,5,'nor',0,1),'r')./([1 :100]'*ones(1,5)))` ?

e. `histplot(100,grand(1,1000,'exp',2))` ? `xtitle('Histogramme expo')` ?

3. Générer un $n-$échantillon de variables gaussiennes $\mathcal{N}(0,1)$, tracer l'histogramme correspondant sur lequel on superposera la densité gaussienne.

2.2 Simulation par fonction de répartition inverse

Proposition 1 *Soit X une variable aléatoire réelle de fonction de répartition F. Pour $u \in [0,1]$, on désigne par $F^{-1}(u) := \inf\{x \in \mathbb{R},\, F(x) \geq u\}$ l'inverse généralisée de la fonction de répartition F. Si $U \sim \mathcal{U}_{[0,1]}$ alors $F^{-1}(U)$ a pour fonction de répartition F.*

1. Que vaut la fonction de répartition de la loi de Bernoulli $\mathcal{B}(1/2)$. À partir de variables uniformes sur $[0,1]$, générer un $n-$échantillon de variables de loi $\mathcal{B}(1/2)$ à valeurs dans $E = \{0,1\}$. Même question lorsque $E = \{-1,1\}$.
2. Simuler une variable aléatoire Y de loi uniforme à valeurs dans $\{0,1,2,3\}$.
3. À partir de variables uniformes, simuler un $n-$échantillon $(X_1,\ldots,X_n)$ de variables aléatoires de loi exponentielle de paramètre 2. Mettre en évidence la loi des grands nombres en traçant la fonction $k \mapsto (X_1 + \ldots + X_k)/k$ pour k allant de 1 à n.

2.3 Simulation par rejet

Proposition 2 *Soient A, $D \in \mathcal{B}(\mathbb{R}^d)$ tels que $A \subset D$. Soit $(X_i)_{i\in\mathbb{N}}$ une suite i.i.d. de variables aléatoires définies sur un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$ uniformes sur D. Introduisons le temps $\tau := \inf\{i \in \mathbb{N}^*,\, X_i \in A\}$, alors X_τ est uniformément distribuée dans A, i.e. pour $B \in \mathcal{B}(\mathbb{R}^d)$, $B \subset A$, $\mathbb{P}(X_\tau \in B) = |B|/|A|$. Le nombre de tirages avant l'obtention d'une réalisation de la loi uniforme sur A par ce procédé suit une loi géométrique $\mathcal{G}(p)$, avec $p = |A|/|D|$. En particulier le nombre moyen de tirages nécessaires est $|D|/|A|$.*

De la proposition, on déduit la méthode de simulation suivante. Soit X une variable aléatoire réelle de loi μ possèdant une densité continue f à support compact inclus dans $[a,b] \subset \mathbb{R}$ et telle que son graphe soit inclus dans $[a,b] \times [0,M]$, pour un certain $M \in \mathbb{R}^+$. On considère une suite $(Z_i)_{i\in\mathbb{N}} = (X_i, Y_i)_{i\in\mathbb{N}}$ de variables aléatoires uniformes dans $[a,b] \times [0,M]$ et l'on définit $\tau = \inf\{i \in \mathbb{N},\, f(X_i) > Y_i\}$. Alors X_τ a pour loi μ.

1. Utiliser la méthode de simulation ci-dessus pour générer une variable aléatoire de densité $x \mapsto \frac{\pi}{2}\sin(\pi x)$ sur l'intervalle $[0,1]$.

On peut généraliser la méthode de rejet de la façon suivante. Soit X une variable aléatoire réelle de loi μ qui possède une densité continue f majorée uniformément par une densité g, i.e. il existe une constante $C \geq 1$ telle que

$$\forall x \in \mathbb{R},\, f(x) \leq Cg(x), \quad \text{avec} \quad \int g(y)dy = 1.$$

On suppose que l'on sait facilement simuler des variables aléatoires de loi de densité g. Soient donc $(X_i)_{i\in\mathbb{N}}$ une suite i.i.d. de variables aléatoires de loi de densité g et $(U_i)_{i\in\mathbb{N}}$ une suite i.i.d.

de variables aléatoires uniformes sur l'intervalle $[0,1]$, indépendantes de $(X_i)_{i\in\mathbb{N}}$. Si l'on considère le temps $\tau := \inf\{i \geq 1,\ f(X_i) > Cg(X_i)U_i\}$, alors X_τ a pour loi μ.

Application : on souhaite simuler une variable aléatoire X de loi $\mathcal{N}(0,1)$. On montre sans trop de difficulté la majoration suivante :

$$\forall x \in \mathbb{R},\ \underbrace{\left(\frac{e^{-x^2/2}}{\sqrt{2\pi}}\right)}_{:=f_X(x)} \leq \underbrace{\sqrt{\frac{2e}{\pi}}}_{:=C} \times \underbrace{\left(\frac{1}{2}e^{-|x|}\right)}_{:=g(x)}.$$

2. a. Montrer que si ε suit une loi de Bernoulli de paramètre $1/2$ dans $\{-1,1\}$ et si Z est une variable exponentielle de paramètre 1 indépendante de ε, alors εZ a pour densité la fonction g sur $\mathbb{R}$;

   b. Implémenter un algorithme qui simule une variable gaussienne via la méthode de rejet ;

   c. Simuler 1000 variables selon cet algorithme et vérifier que l'histogramme correspondant approche bien la densité gaussienne.

2.4 Méthode de Monte-Carlo

La méthode de Monte-Carlo est basée sur la loi des grands nombres. Elle permet par exemple de calculer des valeurs approchées d'intégrales ou d'espérances, en utilisant des réalisations i.i.d. d'une loi que l'on sait simuler. Par exemple, si $(X_n)_{n\geq 1}$ est une suite de variables aléatoires i.i.d. de loi uniforme sur $[0,1]^m$ et si $f : [0,1]^m \to \mathbb{R}$ est une fonction intégrable par rapport à la mesure de Lebesgue sur $[0,1]^m$, alors la loi des grands nombres entraîne la convergence presque-sûre suivante :

$$\frac{1}{n}\left(f(X_1) + \ldots + f(X_n)\right) \to \mathbb{E}(f(X_1)) = \int_{[0,1]^m} f(x)dx \ \ p.s..$$

Si l'on sait majorer la variance de $f(X_1)$, on est de plus en mesure de fournir des intervalles de confiance pour contrôler l'erreur commise dans l'approximation. La vitesse de convergence de cette méthode (de l'ordre de $\sqrt{n}$) est lente par rapport à des méthodes déterministes. Cependant cette vitesse ne dépend pas de la régularité de l'intégrande f et dépend plus faiblement de la dimension m que les méthodes déterministes.

1. Calculer les approximations des intégrales suivantes.

$$\int_0^1 4\sqrt{1-x^2}dx \ \text{ (aire du disque unité)}, \quad \int_{[-1,1]^3} \mathbb{1}_{\{x^2+y^2+z^2\leq 1\}}dxdydz, \ \text{ (volume de la boule unité)}$$