

Méthodes numériques pour la finance

Rapport de TD n°1

Michaël Sibilleau - Master SAF 2 - ISFA

11 février 2007

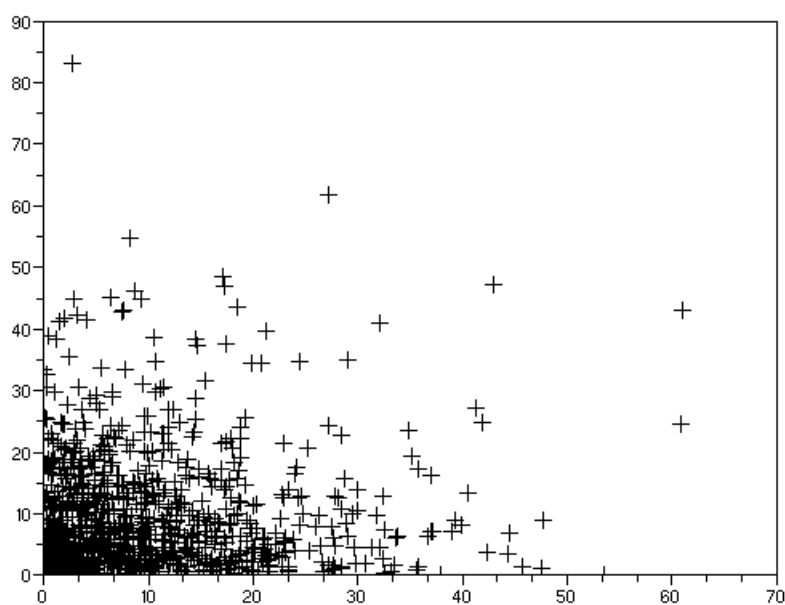


FIG. 1 – simulation d'une t-copule bivariable sur des marginales exponentielles

Nota : Les codes sources (Scilab et R) figurent en annexe. Pour chaque question, il faut utiliser les routines d'exécution en fin de code.

1 Exercice n°1

1.1 Question a

Pour cette question, on utilise les générateurs “mt” et “kiss” à l’aide de l’instruction grand.

1.2 Question b

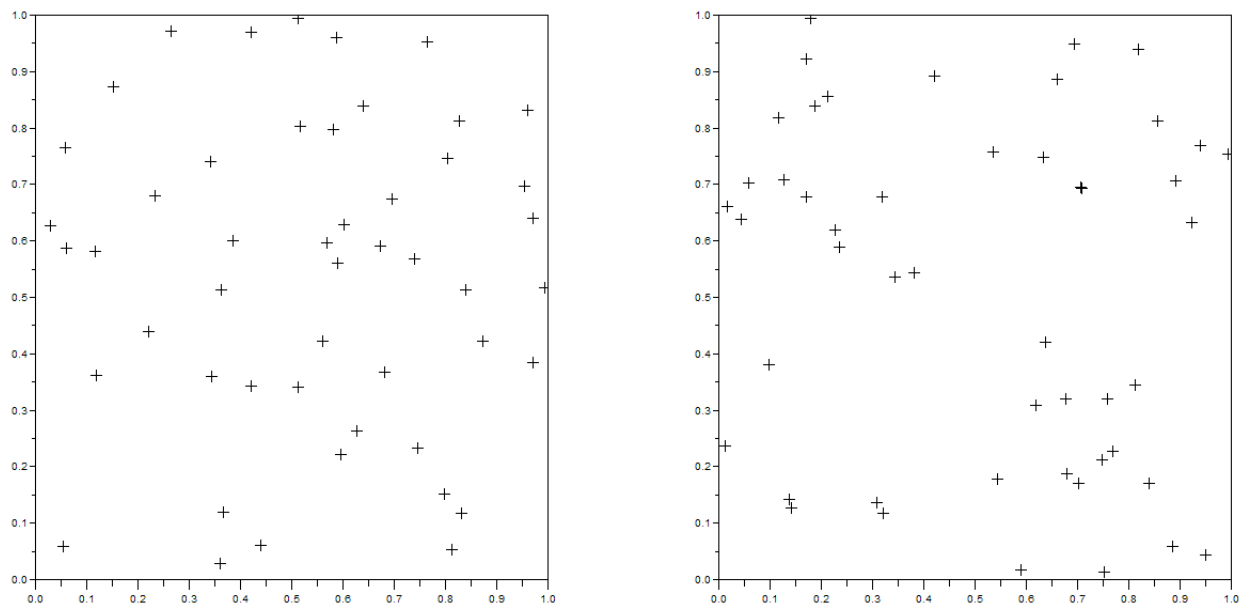


FIG. 2 – Graphes des points ($n = 50$) avec le générateur “mt” (à gauche) et “kiss” (à droite)

On observe que les points générés aléatoirement suivant une loi uniforme remplissent convenablement le carré. Cependant, la répartition est inhomogène (avec un échantillon de taille $n = 50$ seulement, il y a des “trous”). Pour corriger cela, il est possible d’utiliser des méthodes de quasi Monte Carlo.

1.3 Question c

La fonction de répartition empirique est donnée par :

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{\{U_i \leq x\}}$$

et elle tend naturellement vers la fonction de répartition théorique lorsque n est croissant. C’est ce que l’on constate sur le graphe ci-après où on a choisi de représenter les fonctions de répartition empiriques pour les 2 générateurs (“mt” est en bleu) avec $n = 50$ et $n = 150$.

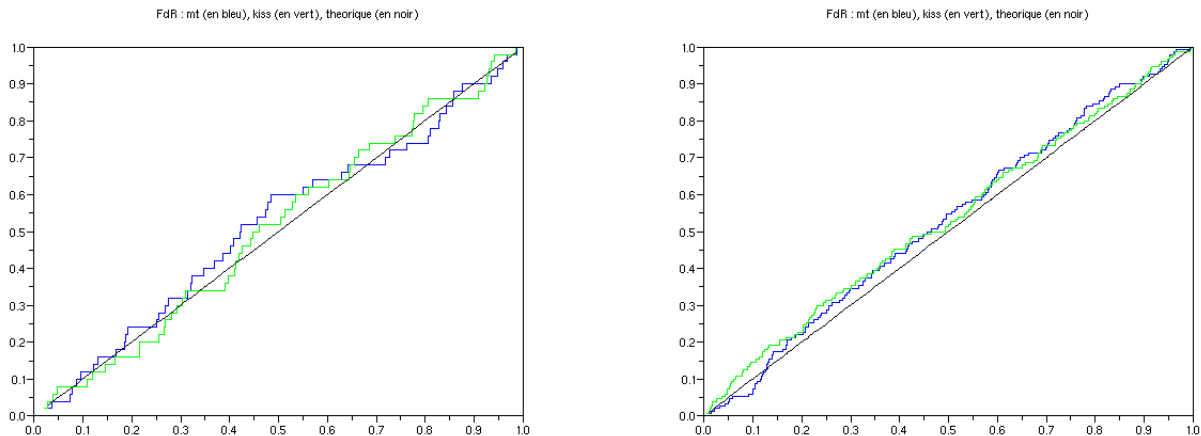


FIG. 3 – FdR empiriques comparées à la FdR théorique ($n = 50$ à gauche et $n = 150$ à droite)

1.4 Question d

Il y a plusieurs façons d’écrire dans un fichier texte et nous proposons les entrées sorties à la C pour constituer deux fichiers .dat (correspondant aux deux générateurs). Ces fichiers sont ensuite lus par le logiciel R au moyen des instructions suivantes :

```
# Code R
u <- read.table("u.dat")
v <- read.table("v.dat")
# p valeurs de u et v
ks.test(u, "punif")
ks.test(v, "punif")
```

Voici les p-valeurs obtenues :

$$p - \text{valeur}(u) = 0.3472$$

$$p - \text{valeur}(v) = 0.1790$$

Ces p valeurs sont élevées et correspondent au test suivant :

H_0 (hyp. nulle) : “ les observations proviennent d’une loi uniforme” contre H_1 (hyp. alternative) “les observations ne proviennent pas d’une loi uniforme”

On en déduit que l’on ne peut rejeter H_0 et que donc, on peut avoir confiance dans l’uniformité des observations. Il faut cependant être prudent car les p valeurs ne sont pas très élevées et en exécutant les routines pour des échantillons de plus grande taille, on obtient par exemple avec $n = 1000$: $p - \text{valeur}(u) = 0.4789$ et $p - \text{valeur}(v) = 0.4932$.

Remarque : Dans le code source, une fonction ks est implémentée pour effectuer le test de Kolmogorov Smirnov sous Scilab avec pour argument le seuil α du test.

2 Exercice n°2

2.1 Question a

Pour simuler une variable aléatoire de loi exponentielle de paramètre θ , on utilise la méthode d'inversion avec :

$$F_{\theta}^{-1}(u) = -\frac{1}{\theta} \ln(1-u)$$

La fonction implémentée est la suivante (l'argument "def" de la fonction *grand* garantit $u \neq 1$)

```
function resultat = Ques2A (n,theta)
    grand('setgen','kiss');
    u=grand(1,n,'def'); // le nombre 1 est exclus
    resultat = (-log2(1-u))/theta;
endfunction
```

2.2 Question b

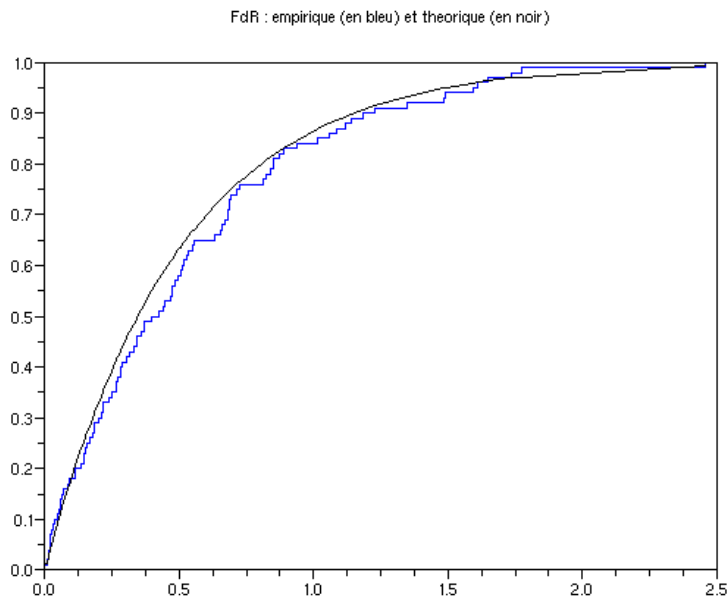


FIG. 4 – FdR empirique et théorique de la loi exponentielle avec $n = 100$ et $\theta = 2$

La figure *supra* donne l'allure des fonctions de répartition théorique et empirique de la loi exponentielle de paramètre $\theta = 2$. Le nombre de réalisations est $n = 100$.

2.3 Question c

Pour vérifier que : $P(\epsilon > 0.75) = P(\epsilon > 0.5) \times P(\epsilon > 0.25)$, il suffit d'exprimer les probabilités sous forme d'espérance d'indicatrices.

On doit donc montrer que : $E(1_{\{\epsilon > 0.75\}}) = E(1_{\{\epsilon > 0.5\}}) \times E(1_{\{\epsilon > 0.25\}})$ et pour ce faire, on utilise l'estimateur du maximum de vraisemblance (tout simplement la moyenne empirique).

La fonction Ques2C comprend deux arguments supplémentaires ie m et α : le premier correspond au nombre de simulations d'un échantillon de taille n et le second à un seuil mesurant le nombre de fois où $P(\epsilon > 0.75)$ s'écarte de $P(\epsilon > 0.5) \times P(\epsilon > 0.25)$.

En d'autres termes, on calcule le nombre de fois où :

$$|P(\epsilon > 0.75) - P(\epsilon > 0.5) \times P(\epsilon > 0.25)| < \alpha$$

ce qui nous donne une confiance dans l'égalité.

Les résultats obtenus sont notifiés dans le tableau *infra* ($m = 100$) :

	$\alpha = 0.1$	$\alpha = 0.05$	$\alpha = 0.01$
$n = 100$	99%	83%	22%
$n = 1000$	100%	85.2%	24.2%

TAB. 1 – part des valeurs relevées pour $|P(\epsilon > 0.75) - P(\epsilon > 0.5) \times P(\epsilon > 0.25)| < \alpha$ avec un échantillon de taille n

Les résultats sont concluants, on peut affirmer que $P(\epsilon > 0.75) = P(\epsilon > 0.5) \times P(\epsilon > 0.25)$ avec une confiance raisonnable.

2.4 Question d

Pour simuler un processus de Poisson d'intensité 2 et dont les sauts sont indépendants et de loi uniforme sur $[0,1]$, on utilise l'algorithme suivant :

```

Debut
  exp <- Simuler n realisations d'une loi exponentielle de parametre 2 // duree inter-occurrence
  u <- Simuler n realisations d'une loi uniforme sur [0,1] // intensite des sauts
  exp <- Somme cumulee (exp) // on prend les sommes cumulees
  u <- Somme cumulee (u) // idem
  Tracer (exp,u)
Fin
    
```

Pour $n = 100$, on obtient par exemple les résultats de la figure 5.

3 Exercice n°3

Schématiquement, il s'agit à porter en ordonnée les valeurs de la fonction de répartition de X puis à simuler une loi uniforme sur $[0,1]$ pour enfin établir les valeurs de x qui satisfont à la fonction de répartition. On a donc la fonction *resultat* ci-après.

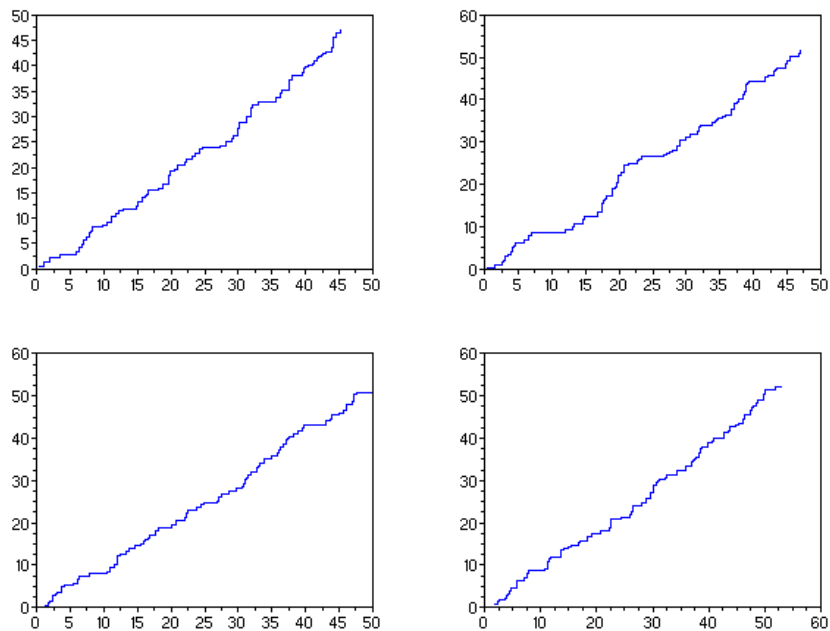


FIG. 5 – trajectoires d'un processus de Poisson ($\theta = 2$, $n = 100$) dont les sauts sont indépendants et de loi uniforme sur $[0,1]$

```

function resultat = loi(n,x,p)
    if (length(x)<>length(p))
        disp("les vecteurs doivent etre de meme taille");
        resultat = 0;
        break;
    else
        FdR = cumsum(p/sum(p));
        grand('setgen','kiss');
        u = grand(1,n,'def');
        for i = 1 :n
            resultat(i) = x(sum(u(i)>FdR)+1);
        end
    end
endfunction

```

Pour $n = 10$ nous obtenons par exemple :

```

-----
resultat de la simulation :
-----
7.
3.
7.
6.
9.
5.

```

- 6.
- 9.
- 10.
- 7.

4 Exercice n°4

4.1 Algorithme de Box Muller

4.1.1 Question a (démonstration de l'algorithme de Box Muller)

Rappelons un résultat du cours :

Théorème : Soit un vecteur aléatoire $X = {}^t (X_1, \dots, X_n)$ de densité f_X et D le domaine supposé ouvert $D = \{x \in \mathbb{R}^n ; f_X(x) > 0\}$. Soit φ une fonction définie sur le domaine D par : $Z = \varphi(X)$ et bijective sur son image $\Delta = \varphi(D)$. On suppose φ et φ^{-1} de classe C^1 sur D et Δ respectivement.

On considère le déterminant :

$$Jac(\varphi)(x) = \begin{vmatrix} \frac{\partial \varphi_1}{\partial x_1} & \dots & \dots & \frac{\partial \varphi_1}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial \varphi_n}{\partial x_1} & \dots & \dots & \frac{\partial \varphi_n}{\partial x_n} \end{vmatrix} \neq 0$$

alors la densité du vecteur Z est donnée par : $f_Z(z) = |Jac(\varphi^{-1})|(z) \times f_X(\varphi^{-1}(z))$

Application : Soient U et V deux variables iid de loi uniforme sur $[0,1]$ et X, Y définies par :

$$\begin{cases} X & := R \cos(\Theta) \\ Y & := R \sin(\Theta) \end{cases} \quad (1)$$

$$\text{avec : } \begin{cases} R & := \sqrt{-2 \ln U} \\ \Theta & := 2\pi V \end{cases} \quad (2)$$

L'image du domaine $D = [0, 1]^2$ est : $\Delta = \{(r, \theta) : 0 < r < +\infty, 0 < \theta < 2\pi\}$

et nous avons : $\begin{cases} u & = \exp(-r^2/2) \\ v & = \frac{\theta}{2\pi} \end{cases}$ d'après (2)

on en déduit donc que :

$$|Jac(\varphi^{-1})| = \begin{vmatrix} \frac{\partial u}{\partial r} & \frac{\partial u}{\partial \theta} \\ \frac{\partial v}{\partial r} & \frac{\partial v}{\partial \theta} \end{vmatrix} = \begin{vmatrix} -re^{-r^2/2} & 0 \\ 0 & \frac{1}{2\pi} \end{vmatrix} = \frac{r}{2\pi} \exp(-r^2/2)$$

Par suite : $f_{R,\Theta}(r, \theta) = \frac{r}{2\pi} e^{-r^2/2}$ puisque $f_{U,V}$ est la densité bivariable de deux lois uniformes indépendantes

Comme de plus $R \perp \Theta$ (car $U \perp V$), on a :

$$\begin{cases} f_R(r) & = re^{-r^2/2} \\ f_\Theta(\theta) & = \frac{1}{2\pi} \end{cases}$$

D'après (1), il est encore possible d'écrire :

$$\begin{cases} R &= \sqrt{X^2 + Y^2} \\ \Theta &= \text{Arctan}\left(\frac{Y}{X}\right) \end{cases}$$

On peut écrire le déterminant jacobien de la façon suivante :

$$|Jac(\varphi^{-1})| = \begin{vmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} \end{vmatrix} = \begin{vmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} \\ \frac{-y}{\sqrt{x^2+y^2}} & \frac{x}{\sqrt{x^2+y^2}} \end{vmatrix} = \frac{1}{\sqrt{x^2+y^2}}$$

ce qui nous permet d'exprimer la densité du couple de variables aléatoires (X, Y) par :

$$f_{X,Y}(x, y) = f_{R,\Theta}\left(\sqrt{x^2 + y^2}, \text{Arctan}\left(\frac{x}{y}\right)\right) \times \frac{1}{\sqrt{x^2+y^2}}$$

$$\text{soit : } f_{X,Y}(x, y) = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) = \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\right) \times \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}\right)$$

ce qui termine la démonstration.

4.2 Question b

On utilise l'algorithme de Box Muller pour simuler $2n$ échantillons d'une loi normale (fonction Ques4B).

Une représentation des points de coordonnées (X, Y) est donnée par le graphique *infra* pour des simulations d'une loi normale de moyenne 5 et de variance 10.

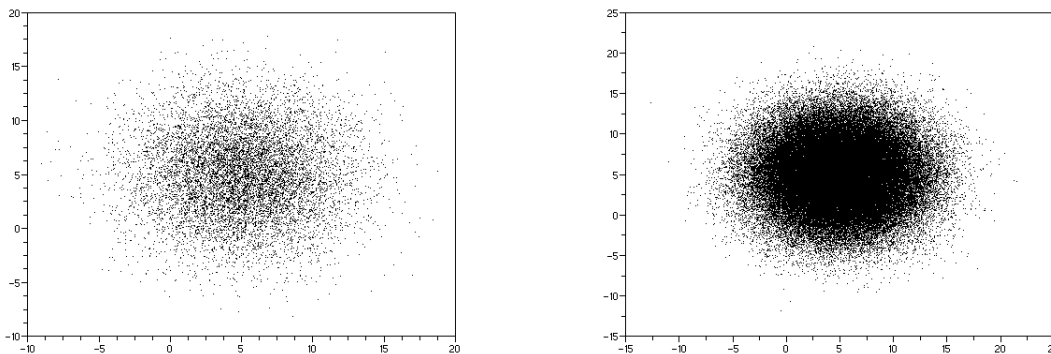


FIG. 6 – simulation de 10.000 points (à gauche) et 100.000 (à droite) d'une loi normale avec $m = 5$ et $\sigma^2 = 10$

4.3 Algorithme de Marsaglia-Bray

4.3.1 Question a

Pour simuler des variables uniformes sur le disque unité, on utilise l'algorithme suivant :

```

Repeter
  U := 2*random()-1
  V := 2*random()-1
  S := U^2+V^2
Jusqu'a
  S <= 1 // rejet
    
```

En Scilab, c'est la fonction SimuDisque qui réalise cet algorithme (voir codes source). La seule difficulté en terme de programmation, c'est qu'il n'existe pas d'instruction du type "do ... until". Pour substituer cette fonctionnalité, nous utilisons simplement un booléen (bol). La simulation de variables uniformes sur le cercle unité s'obtient à partir de la fonction SimuDisque en projetant les réalisations sur le cercle de centre 0 et de rayon 1.

A titre de remarque, on obtient les graphes suivants (attention aux échelles) :

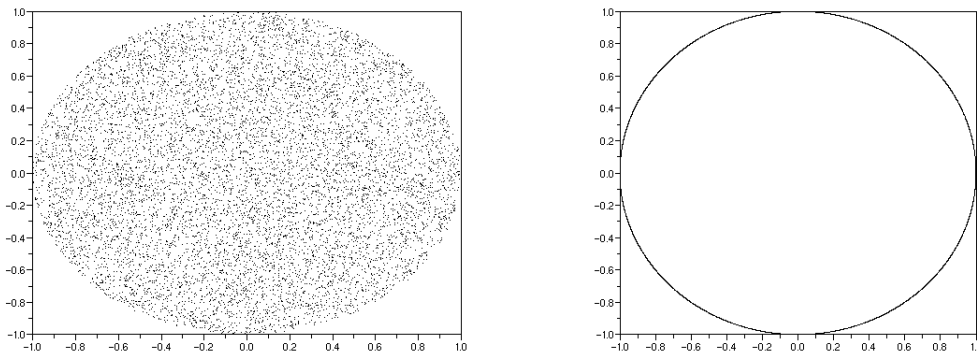


FIG. 7 – échantillons de 10.000 simulations de la loi uniforme sur le disque et sur le cercle unité par la méthode du rejet

4.3.2 Question b

L'algorithme de Box Muller permet de simuler des variables normales centrées réduites à partir de :

$$\begin{cases} X &= \sqrt{-2\ln U} \cos(2\pi V) \\ Y &= \sqrt{-2\ln U} \sin(2\pi V) \end{cases}$$

avec U et V des lois iid uniformes sur $(0,1)$. Nous avons donc :

$$X^2 + Y^2 = -2\ln U \quad (1)$$

et si l'on considère maintenant le couple (U', V') uniformément distribués sur le disque, d'après l'algorithme 4.3.1, on sait que $U'^2 + V'^2$ est de loi uniforme sur $(0,1)$.

On peut donc réécrire (1) sous la forme suivante :

$$X^2 + Y^2 = -2\ln(U'^2 + V'^2) \times \frac{U'^2 + V'^2}{U'^2 + V'^2}$$

en posant $S := U'^2 + V'^2$ il vient : $X^2 + Y^2 = -2\ln S \times \frac{U'^2 + V'^2}{S}$

Nous pouvons donc extraire :

$$\begin{cases} X' &= \sqrt{(-2\ln S)/S} \times U' \\ Y' &= \sqrt{(-2\ln S)/S} \times V' \end{cases}$$

On peut donc proposer l'algorithme suivant (*algorithme de Marsaglia Bray*) :

```

Debut
  Repeter
    U := 2*random()-1
    V := 2*random()-1
    S := U^2+V^2
  Jusqu'a
    S <= 1 // rejet
    Z := sqrt(-2*log(S)/S)
    X1 = Z*U
    X2 = Z*V
Fin

```

4.3.3 Question c

L'avantage de l'algorithme de Marsaglia Bray sur celui de Box Muller est qu'il est plus rapide (facteur 2) car il utilise moins de fonctions coûteuses comme les fonctions trigonométriques. Il présente le seul inconvénient d'utiliser une boucle pour le rejet (instruction "répéter ... jusqu'à").

4.3.4 Questions d et e

La fonction `ExporterVersR` assure l'exportation vers les fichiers MB et BM des simulations respectivement obtenues avec les algorithmes de Marsaglia Bray et de Box Muller. Sous R, la fonction `Exo4` récupère ces données, trace les graphes `qqplot` et fournit les résultats des tests de Shapiro-Wilks.

Pour un échantillon de $n = 1000$ simulations d'une loi normale centrée réduite, on obtient les résultats de la figure 8

Les résultats des tests sont :

```

Resultats pour Box Muller
-----
statistic p.value method data.name
[1,] 0.9983896 0.4846599 "Shapiro-Wilk normality test" "BM1"
[2,] 0.9983882 0.4838206 "Shapiro-Wilk normality test" "BM2"

Resultats pour Marasaglia Bray
-----
statistic p.value method data.name
[1,] 0.9986226 0.6362549 "Shapiro-Wilk normality test" "MB1"
[2,] 0.9988237 0.7701876 "Shapiro-Wilk normality test" "MB2"

```

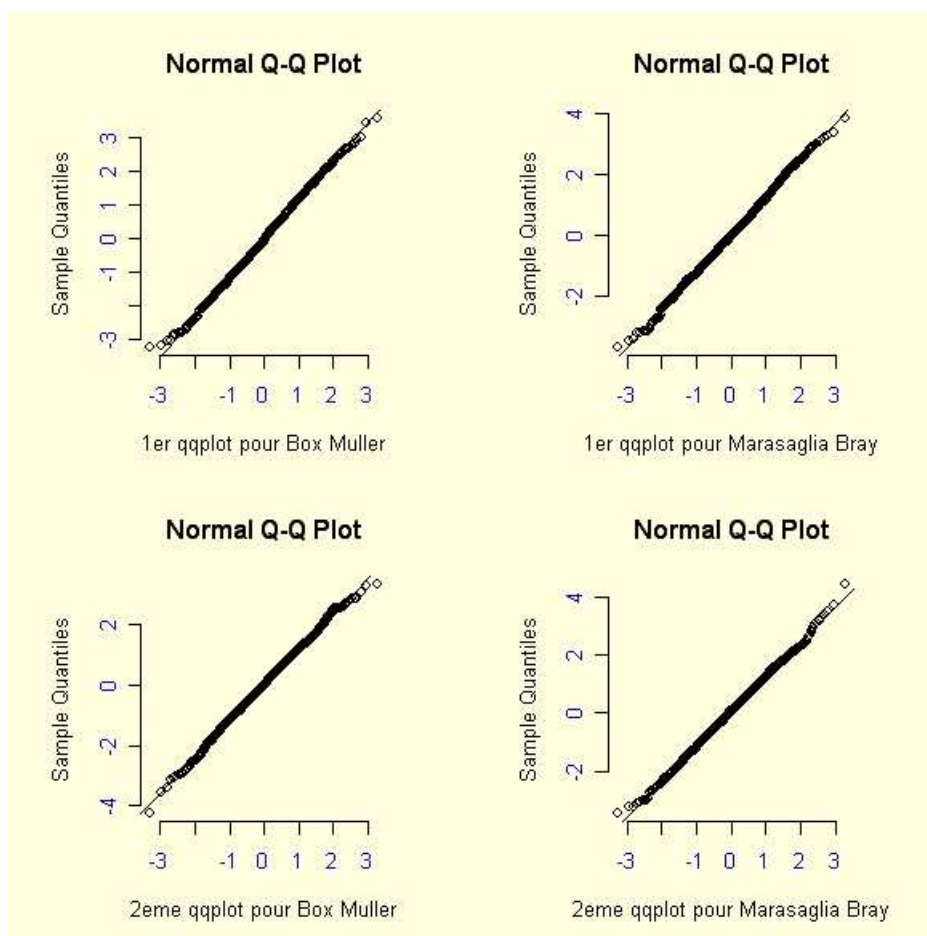


FIG. 8 – qqplot obtenus pour une loi normale centrée réduite avec $n = 1000$

On rappelle que pour un test de Shapiro-Wilks¹, l’hypothèse nulle est ici : “ les observations proviennent d’une loi normale” contre H1 (hyp. alternative) “les observations ne proviennent pas d’une loi normale”.

On en déduit que l’on ne peut rejeter H0 et que donc, on peut avoir confiance dans la normalité. On observe que :

1. les résultats obtenus avec l’algorithme de Marsaglia Bray sont meilleurs
2. les graphes obtenus confirment les résultats des tests (droites)

5 Exercice n°5

5.1 Question a

Pour établir la densité de la loi $Beta(3,2)$, commençons par calculer : $\beta(3,2) = \int_0^1 x^2(1-x)dx = \frac{1}{12}$

Il vient : $f_{Beta(3,2)}(x) = 12x^2(1-x)\mathbf{1}_{[0,1]}(x)$

A titre indicatif, la représentation graphique de $f_{Beta(3,2)}$ est illustrée par le graphe suivant :

¹comme pour un test de Kolmogorov Smirnov

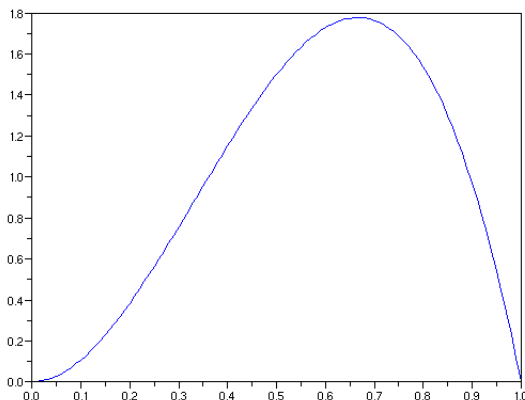


FIG. 9 – Représentation graphique de la densité de la loi $Beta(3, 2)$

Nous choisissons pour g la densité de la loi uniforme sur $[0,1]$ ie : $g(x) = \mathbf{1}_{[0,1]}(x)$

Nous avons donc $\forall x \in [0, 1] \quad h(x) = \frac{f(x)}{g(x)} = 12x^2(1 - x)$

Et par suite : $c = \sup_{[0,1]} \left(\frac{f(x)}{g(x)} \right) = h\left(\frac{2}{3}\right) = \frac{16}{9}$

5.2 Question b

On considère à présent la densité de la loi normale centrée réduite ie $f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ et $g(x) = \frac{1}{2}e^{-|x|} \neq 0$ ($\forall x \in \mathbb{R}$)

On a cette fois : $h(x) = \frac{f(x)}{g(x)} = \frac{2}{\sqrt{2\pi}}e^{-x^2/2+|x|}$

Une rapide étude de cette fonction conduit à $c = \sup_{\mathbb{R}} \left(\frac{f(x)}{g(x)} \right) = h(1) = \frac{2}{\sqrt{2\pi}}e^{1/2}$

Le graphe 10 donne une idée des représentations graphiques de f et g (partie gauche) et de l'efficacité dans le choix du facteur c (à droite). En multipliant la densité de g par ce facteur, on obtient une fonction au-dessus (mais suffisamment proche) de f en tout point de \mathbb{R} .

5.3 Remarques sur les simulations

La simulation de $Beta(3,2)$ ne pose pas de difficulté particulière (voir la fonction Ques5A).

Celle de la loi normale appelle une observation quant à la simulation de la fonction $g(x) = \frac{1}{2}e^{-|x|}$.

Pour simuler une telle variable, on remarque que g est paire et que l'on peut écrire $g(x) = \frac{1}{2}e^{\frac{1}{2} \times (-2 \times |x|)}$

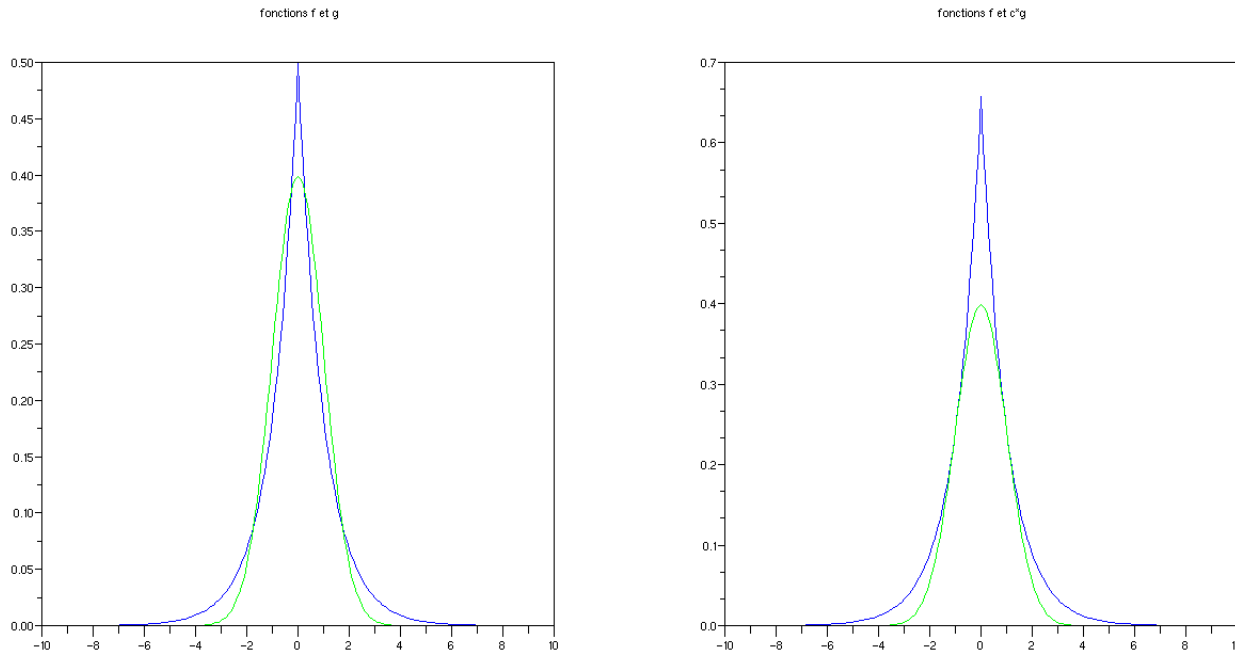


FIG. 10 – densités de f (vert) et g (bleu) à gauche et de f et $c \times g$ à droite

D'où l'algorithme suivant :

Debut

```

expo <- Simuler n realisations d'une loi exponentielle de parametre 1/2
u <- Simuler n realisations d'une loi uniforme sur [0,1]
epsilon <- (1_{u<0.5}) * 2 - 1 // epsilon vaut equiprobablement -1 ou +1
renvoyer 0.5 * (expo * epsilon)

```

Fin.

L'implémentation de cet algorithme correspond à la fonction SimuG.

S'agissant de l'adéquation aux lois simulées, on exporte les données dans un fichier .dat (Beta.dat et Normale.dat). Un test de Kolmogorov Smirnov donne les résultats suivants (pour $n = 1000$) :

statistic	p.value	alternative	method	data.name
[1,]	0.02278056	0.677088	"two-sided"	"One-sample Kolmogorov-Smirnov test" "LoiBeta"
[2,]	0.01532554	0.9729307	"two-sided"	"One-sample Kolmogorov-Smirnov test" "LoiNormale"

Les p valeurs obtenues sont très satisfaisantes.

A titre d'exemple, le graphe qq plot obtenu pour la loi normale est le suivant :

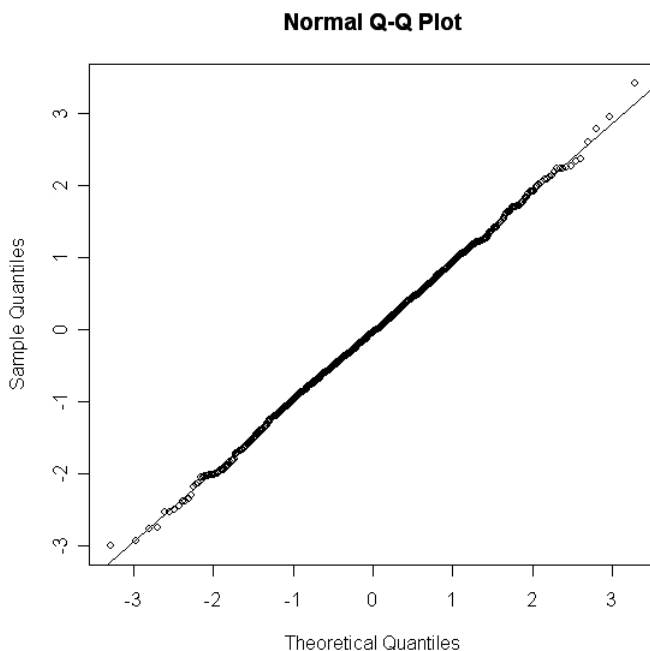


FIG. 11 – Simulation de la loi normale centrée réduite par la méthode du rejet (échantillon $n = 1000$)

6 Exercice n°6

6.1 Question a

On rappelle que la loi log-normale est la loi d'une variable aléatoire X telle que son logarithme népérien suive une loi normale ie :

$$\ln X \sim N(m, \sigma^2)$$

Pour simuler une telle loi, il suffit tout simplement de simuler une variable aléatoire de loi normale (par l'algorithme de Marsaglia-Bray qui est plus efficace) et de prendre l'exponentielle.

6.2 Question b

Soit un actif dont la dynamique de prix est donnée par :

$$\frac{dS_t}{S_t} = \mu dt + \sigma dB_t \quad (1)$$

Considérons maintenant le processus d'Itô suivant :

$$dX_t = \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma dB_t$$

qui s'écrit encore :

$$X_t = X_0 + \left(\mu - \frac{\sigma^2}{2}\right)t + \sigma B_t$$

On applique la formule d'Itô avec $f(t, x) = e^x$ pour trouver la dynamique de $S_t = exp(X_t) = S_0 exp \left[\left(\mu - \frac{\sigma^2}{2} \right) t + \sigma B_t \right]$

Il vient :

$$dS_t = \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial x} dX_t + \frac{1}{2} \frac{\partial^2 f}{\partial x^2} d \langle X \rangle_t$$

$$\text{soit : } dS_t = 0 + e^{X_t} dX_t + \frac{1}{2} e^{X_t} \sigma^2 dt = S_t \left(\mu - \frac{\sigma^2}{2} \right) dt + \sigma dB_t + \frac{1}{2} \sigma^2 S_t dt$$

ce qui nous donne l'équation (1).

Conclusion :

$$S_T = S_0 exp \left[\left(\mu - \frac{\sigma^2}{2} \right) T + \sigma B_T \right] \quad (2)$$

6.3 Question c

Pour cette question, on remarque que $Z_T = S_T/S_0$ est log-normale avec :

$$E(\ln Z_T) = \left(\mu - \frac{\sigma^2}{2} \right) T$$

$$\text{et } Var(\ln Z_T) = \sigma^2 T$$

L'algorithme est donc le suivant :

Debut

`S_T <- Simuler n realisations d'une loi log-normale avec $\left(\mu - \frac{\sigma^2}{2} \right) T$ et $\sigma^2 T$ pour parametres`

`S_T <- $S_T \times S_0$`

Fin

6.4 Question d

Dans le modèle de Black-Scholes, le prix d'un Call européen peut s'écrire sous la forme d'une espérance sous probabilité risque neutre \mathbb{Q} de la manière suivante :

$$Call(S_T, r, \sigma^2, T) = E_{\mathbb{Q}} [e^{-rT} (S_T - K)^+]$$

Les méthodes de Monte Carlo approchent ce calcul d'espérance par une moyenne pour construire un estimateur :

$$Call_n(S_T, r, \sigma^2, T, K) = \frac{e^{-rT}}{n} \sum_{i=1}^n max(S_i - K, 0) \quad (3)$$

On simule donc sous \mathbb{Q} n réalisations du prix du sous jacent et on utilise (3) avec n suffisamment grand puisque :

$$Call_n(S_T, r, \sigma^2, T, K) \rightarrow Call(S_T, r, \sigma^2, T, K) \text{ pour } n \rightarrow +\infty$$

L'intervalle de confiance de niveau 95% correspondant est donné par (cours) :

$$Call(S_T, r, \sigma^2, T) \in \left[\bar{X}_n - 1.96 \frac{\bar{\sigma}_n}{\sqrt{n}}, \bar{X}_n + 1.96 \frac{\bar{\sigma}_n}{\sqrt{n}} \right]$$

où \bar{X}_n et $\bar{\sigma}_n$ désignent respectivement les moyenne et écart-type empiriques de $e^{-rT} \max(S_i - K, 0)$.

La fonction Call permet d'obtenir le graphe de haut de la figure 12 (cette figure est en fin de rapport). On constate bien que plus le nombre de simulations est élevé, meilleure est la précision.

6.5 Question e

On reprend les démarches détaillées *supra*, mais cette fois on utilise une suite à discrétance faible (on utilise la fonction Faure fourni).

On obtient le graphe du bas de la figure 12 sur lequel on remarque une convergence plus rapide vers la valeur limite ainsi qu'un "lissage" des bornes up et down.

7 Exercice n°7

7.1 Question a

Pour simuler un vecteur gaussien $X \sim N(m, K)$ avec K une matrice de covariance symétrique positive, nous utilisons le fait que :

$$N(0, K) = B \times N(0, \mathbf{I})$$

où B est la décomposition triangulaire inférieure de Cholesky qui vérifie : $K = BB'$

D'où l'algorithme suivant :

```

Debut
  Y <- Simuler n variables aleatoires normales centrees reduites independantes
  X <- m+B*Y // B est tel que BB'=K
Fin
    
```

7.2 Question b

7.2.1 préambule

Tout d'abord, rappelons que pour simuler $\mathbf{U} = (U_1, \dots, U_n)$ vérifiant : $C(U_1, \dots, U_n) = \mathbf{F}(\mathbf{F}_1^{-1}(U_1), \dots, \mathbf{F}_n^{-1}(U_n))$, nous pouvons simuler le vecteur aléatoire $\mathbf{X} = (X_1, \dots, X_n)$ de distribution \mathbf{F} et appliquer la transformation $\mathbf{U} = (\mathbf{F}_1(X_1), \dots, \mathbf{F}_n(X_n))$.

Cette méthode est intéressante si la distribution de \mathbf{F} est plus facile à simuler que la copule C , ce qui est le cas de la copule normale.

7.2.2 exercice

On souhaite simuler un vecteur dont chaque composante est de loi exponentielle et dont la relation de dépendance est définie par une copule gaussienne de matrice de corrélation Σ . Compte tenu de 7.2.1, on utilise l'algorithme suivant :

```
Fonction SimuVecteur(theta,K)
Debut
  Y <- Simuler n variables aleatoires normales centrees reduites independantes
  Z <- B*Y // B est tel que BB'=K
  UNIF <- F(Z) // F est la fonction de repartition de la loi normale centree reduite
// application de la methode inverse pour la loi exponentielle de parametre theta
  EXPO <- -log(UNIF)/theta
  retourner EXPO
Fin
```

7.3 Question c

Pour cette question, le principe est le même. On utilise l'algorithme suivant (ddl est le degré de liberté de la copule) :

```
Fonction t_SimuVecteur(theta,ddl,K)
Debut
  Y <- Simuler n variables aleatoires normales centrees reduites independantes
  Z <- B*Y // B est tel que BB'=K; Z est un vecteur de loi normal multivarie
  W <- Simuler 1 variable aleatoire du Khi deux avec ddl degre de liberte
  Z <- Z/sqrt(W/ddl)
// F est la fonction de repartition de la loi de Student a ddl degres de liberte
  UNIF <- F(Z)
// application de la methode inverse pour la loi exponentielle de parametre theta
  EXPO <- -log(UNIF)/theta
  retourner EXPO
Fin
```

7.4 Résultats obtenus sur des simulations de copules bivariées

Nous reportons (voir *infra*) les résultats graphiques des simulations portant sur des échantillons de taille $n = 1000$ avec $ddl = 100$.

Nous pouvons remarquer que :

- l'introduction d'une dépendance forte ($\rho = 0.9$) influence considérablement la disposition des points
- l'accroissement du paramètre θ de la loi exponentielle concentre très fortement les points (voir les figures 14 et 17)

GRAPHIQUES ET CODES

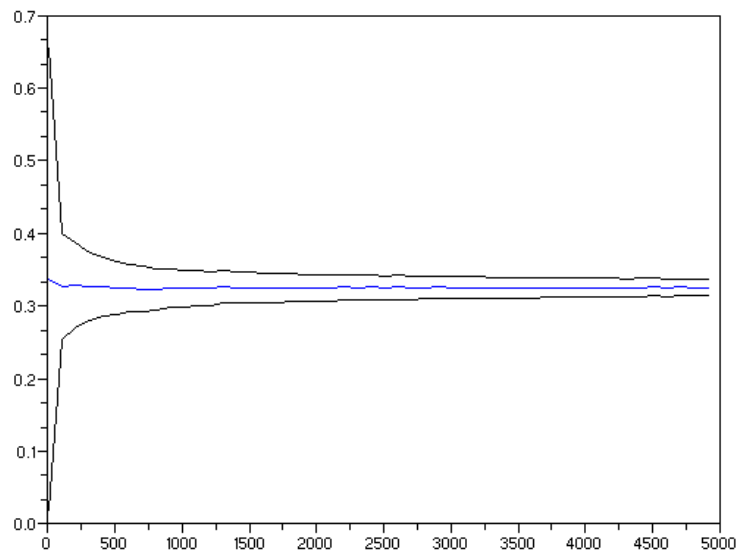
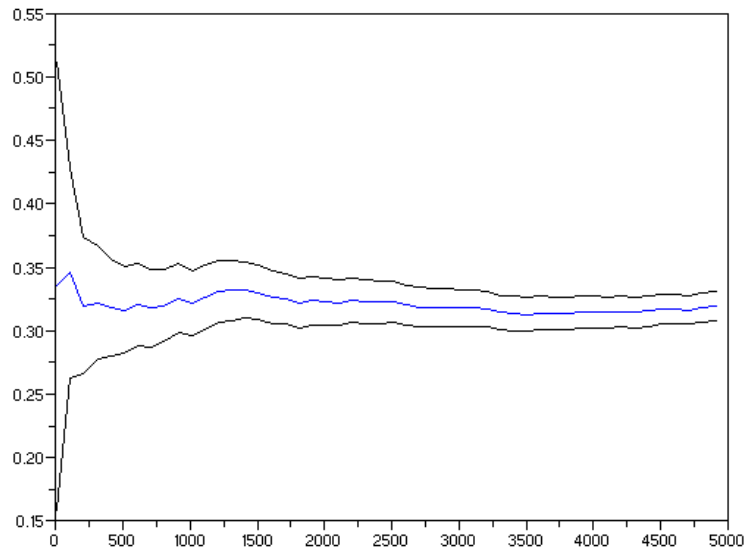


FIG. 12 – Intervalle de confiance à 95% par Monte Carlo (en haut) et Quasi Monte Carlo (en bas) en fonction du nombre de simulations

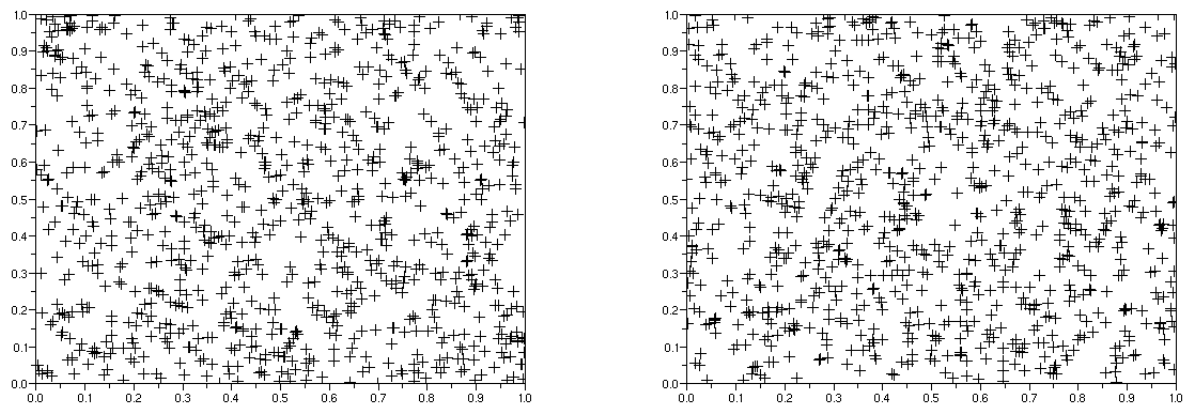


FIG. 13 – marginales uniformes **indépendantes** sur une copule normale (à gauche) et une t copule (à droite)

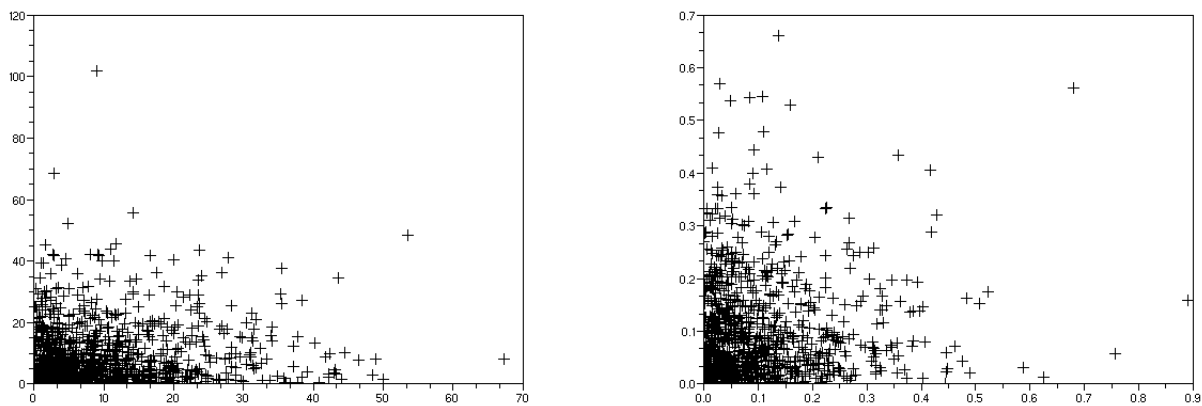


FIG. 14 – marginales exponentielles **indépendantes** avec $\theta = 0.1$ (à gauche) et $\theta = 10$ (à droite) pour une copule normale

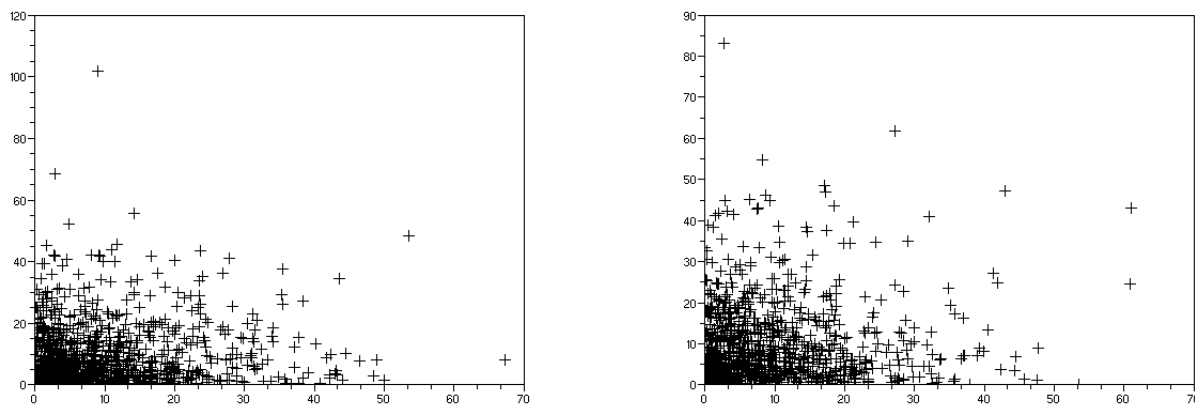


FIG. 15 – marginales exponentielles **indépendantes** avec $\theta = 0.1$ sur une copule normale (à gauche) et de Student (à droite)

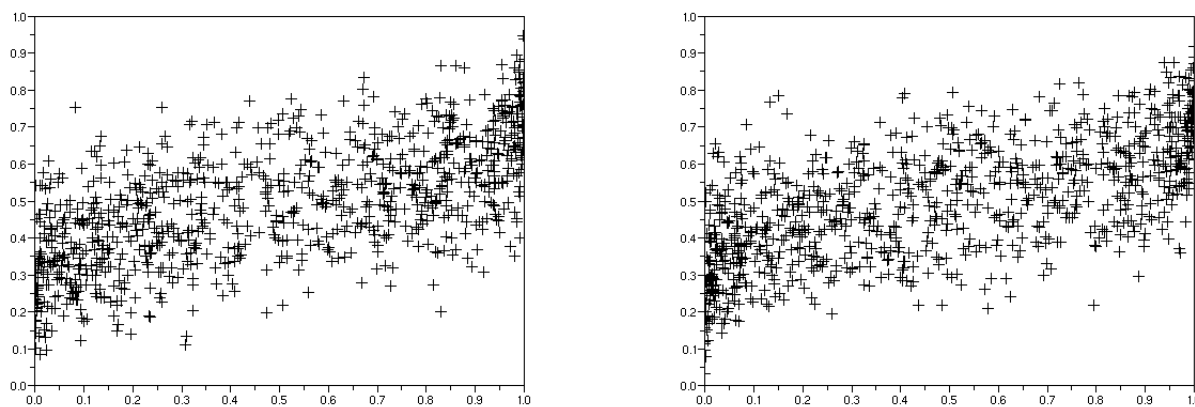


FIG. 16 – marginales uniformes avec **dépendance forte** sur une copule normale (à gauche) et de Student (à droite)

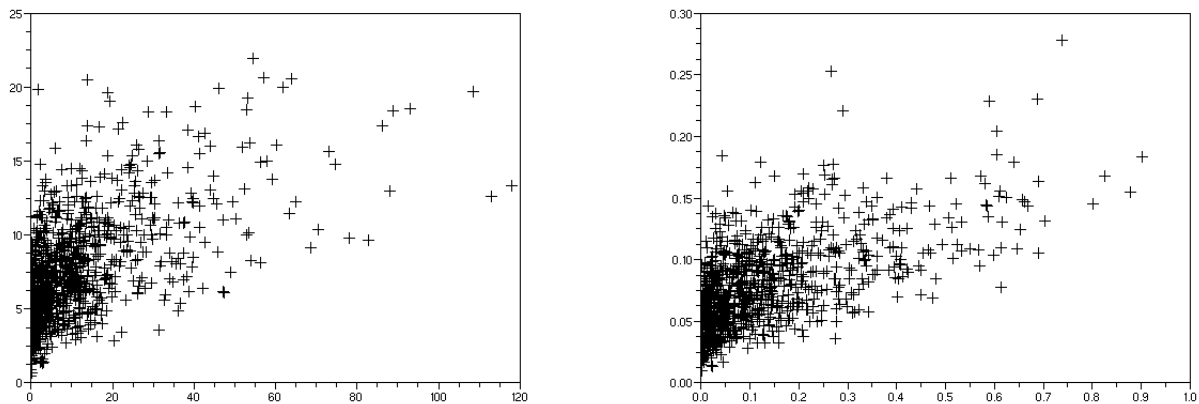


FIG. 17 – marginales exponentielles avec **dépendance forte** et avec $\theta = 0.1$ (à gauche) et $\theta = 10$ (à droite) pour une copule de Student

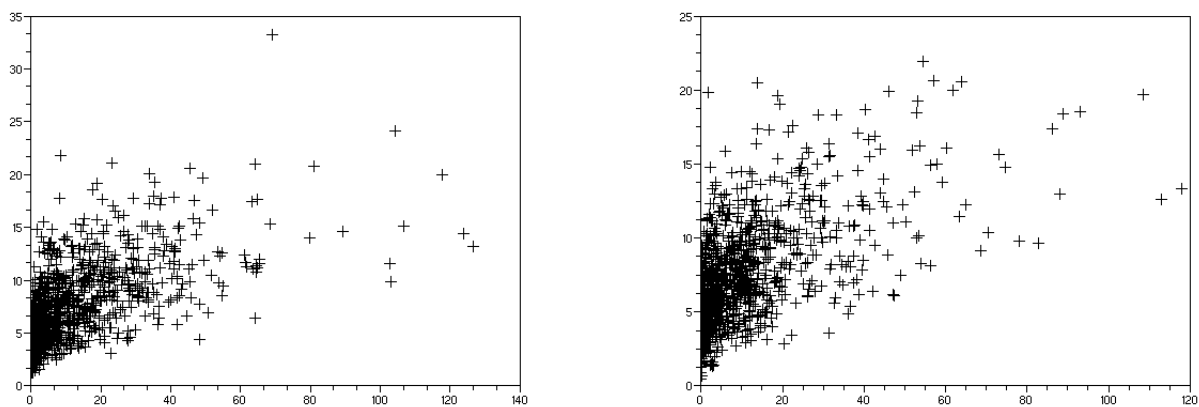


FIG. 18 – marginales exponentielles avec **dépendance forte** et avec $\theta = 0.1$ sur une copule normale (à gauche) et de Student (à droite)

```

// *****
// **   Sujet : Simulations           ***
// **   Auteur : Michael Sibilleau   ***
// **   www.sibilleau.com            ***
// **   Date : 3 fevrier 2007        ***
// *****

// definir chemin comme repertoire de travail
chemin = 'C:\Documents and Settings\Utilisateur\Mes documents\ENSI_ISFA\finances\
          MethodesNum\TD1'

chdir(chemin);
getf(chemin + '\TP.sci'); // pour l'exercice 6

//-----
//                Exercice 1
//-----

n = 50; // taille de l'echantillon

// 1er generateur
grand('setgen','mt');
u=grand(1,n,'def');
// 2eme generateur
grand('setgen','kiss');
v=grand(1,n,'def');

// Question B (trace des courbes)
function QuesB (u,v,bol)
    if (bol==%t) then
        Success_u = u(2:n);
        Success_v = v(2:n);
        u = u(1:(n-1));
        v = v(1:(n-1));
        xbascc();
        subplot(1,2,1); // partage de l'ecran graphique
        plot2d(u,Success_u,style=[-1]);
        xtitle("generateur mt");
        subplot(1,2,2);
        plot2d(v,Success_v,style=[-1]);
        xtitle("generateur kiss");
    end
endfunction

// Question C (trace des fonctions de repartition)
function QuesC (u,v,bol)
    if (bol==%t) then
        u=-sort(-u); // on peut aussi utiliser gsort(u,'c','i')
        v=-sort(-v);
        ordonnee = (1:n)/n;
        xbascc();
        plot2d2(u,ordonnee,style=[2]);
        plot2d2(v,ordonnee,style=[3]);
        plot2d(u,u,style=[1]);
        xtitle("FdR : mt (en bleu), kiss (en vert), theorique (en noir)");
    end
endfunction

// Question D (exportation des simulations dans un fichier de donnees)
function QuesD (u,v,bol)
    if (bol==%t) then
        fichier_u = mopen("u.dat","w+");
        fichier_v = mopen("v.dat","w+");
        fprintfMat("u.dat",u);
        fprintfMat("v.dat",v);
        mclose("all");
    end
endfunction

```

```

// une fonction pour effectuer le test de Kolmogorov Smirnov en Scilab
function ks (x,alpha,bol)
    if (bol==%t) then
        x = -sort(-x);
        D_plus = max((1:n)/n - x);
        D_moins = max(x - (0:n-1)/n);
        K = sqrt(n)*max([D_plus;D_moins]);
        K_seuil = sqrt(log(1/alpha)/2)-1/(6*sqrt(n)) ;
        if (K > K_seuil) then
            write(%io(2),"Resultat du test de ks : Hypothese nulle rejetee");
        else
            write(%io(2),"Resultat du test de ks : Hypothese nulle non rejetee");
        end
    end
endfunction

// routines d'execution
QuesB(u,v,%f);
QuesC(u,v,%f);
QuesD(u,v,%f);
ks(u,0.05,%f);
ks(v,0.05,%f);

//-----
//                Exercice 2
//-----

// Question 2A
function resultat = Ques2A (n,theta)
    grand('setgen','kiss');
    u=grand(1,n,'def'); // le nombre 1 est exclus
    resultat = (-log(1-u))/theta;
endfunction

// Question 2B

function resultat=FdRExp(x,theta) // fonction de repartition de la loi exp de param theta
    resultat = 1-exp(-theta*x);
endfunction

function Ques2B (n,theta,bol)
    if (bol==%t) then
        x = Ques2A(n,theta);
        x = -sort(-x);
        ordonnee = (1:n)/n;
        xbasec();
        plot2d2(x,ordonnee,style=[2]);
        plot2d(x,FdRExp(x,theta),style=[1]);
        xtitle("FdR : empirique (en bleu) et theorique (en noir)");
    end
endfunction

// Question 2C
// m : on effectue m simulations d'un echantillon de taille n
// on veut verifier le nombre de fois ou |P-P1*P2|<alpha
function Ques2C (n,m,alpha,theta,bol)
    if(bol==%t) then
        echantillon = zeros(1,m); // initialisation d'un vecteur de taille m
        for i = 1:m
            x = Ques2A(n,theta);
            P = sum(x > 0.75)/n;
            P1 = sum(x > 0.5)/n;
            P2 = sum(x > 0.25)/n;
            echantillon(1,i) = abs(P-P1*P2);
        end
        resultat = 100*sum(echantillon<=alpha)/m;
        disp(resultat);
    end
endfunction

```

```
// Question 2D
function Ques2D (n,theta,bol)
    if (bol==%t) then
        xbaso();
        for i = 1:4
            subplot(2,2,i); // partage de l'ecran graphique
            x = cumsum(Ques2A(n,theta));
            y = cumsum(grand(1,n,'def'));
            plot2d2(x,y,style=[2]);
        end
    end
endfunction

//routines d'execution
Ques2B(100,2,%f);
Ques2C (100,100,0.1,2,%f);
Ques2D (100,2,%f);
```

```
//-----
//                Exercice 3
//-----
```

```
function resultat = loi(n,x,p)
    if (length(x)<>length(p))
        disp("les vecteurs doivent etre de meme taille");
        resultat = 0;
        break;
    else
        FdR = cumsum(p/sum(p));
        grand('setgen','kiss');
        u = grand(1,n,'def');
        for i = 1:n
            resultat(i) = x(sum(u(i)>FdR)+1);
        end
    end
endfunction
```

```
function Ques3 (bol)
    if (bol==%t) then
        // choix des parametres
        n = 10;
        x = (1:10);
        p = [2,4,8,0,4,7,2,1,7,8];
        xprime = loi(n,x,p);
        disp("-----");
        disp("resultat de la simulation :");
        disp("-----");
        disp(xprime);
    end
endfunction
```

```
// execution
Ques3(%f);
```

```
//-----
//                Exercice 4
//-----
```

```
// algorithme de BOX MULLER
function resultat = BoxMuller (n,moyenne,variance,bol)
    if (bol==%t) then
        grand('setgen','mt');
        u = grand(1,n,'def');
        v = grand(1,n,'def');
        x = sqrt(-2*log(u)).*cos(2*pi*v);
        y = sqrt(-2*log(u)).*sin(2*pi*v);
        // on prend en compte les parametres de la loi normale
```

```

    x = x*sqrt(variance) + moyenne;
    y = y*sqrt(variance) + moyenne;
    resultat = [x;y]; // valeur retournee par la fonction
else
    resultat = 0; // eviter les erreurs d'execution si bol==%f
end
endfunction

// Questions B/a
function resultat = SimuDisque(n,graphe)
    grand('setgen','mt');
    for i = 1:n
        bol = %t; // on veut une iteration du type do ... until
        while (bol==%t)
            unif = grand(1,2,'def');
            unif = 2*unif-1;
            if (unif(1)^2+unif(2)^2 <= 1) then
                resultat(:,i) = [unif(1);unif(2)]; // on forme une matrice de 2 lignes
                bol = %f;
            end
        end
    end
    // verif graphique (option)
    if (graphe == %t) then
        xbas();
        plot2d(resultat(1,:),resultat(2,:),style=[0]);
    end
endfunction

function resultat = SimuCercle(n,graphe)
    disque = SimuDisque(n,%f);
    x = disque(1,:);
    y = disque(2,:);
    R = sqrt(x.^2+y.^2);
    resultat = [x./R;y./R];
    // verif graphique (option)
    if (graphe == %t) then
        xbas();
        plot2d(resultat(1,:),resultat(2,:),style=[0]);
    end
endfunction

// algorithme de MARSAGLIA BRAY
function resultat = MarBray(n,moyenne,variance,bol)
    if (bol==%t) then
        disque = SimuDisque(n,%f);
        S = (disque(1,:).^2+(disque(2,:)).^2);
        Z = sqrt(-2*log(S)./S);
        x = Z.*disque(1,:);
        y = Z.*disque(2,:);
        // on prend en compte les parametres de la loi normale
        x = x*sqrt(variance) + moyenne;
        y = y*sqrt(variance) + moyenne;
        resultat = [x;y]; // valeur retournee par la fonction
    else
        resultat = 0; // eviter les erreurs d'execution avec bol==%f
    end
endfunction

// Questions B/ d et e
function ExporterVersR (n,moyenne,variance,bol)
    if (bol==%t) then
        BM = BoxMuller(n,moyenne,variance,%t);
        MB = MarBray(n,moyenne,variance,%t);
        fichier_BM = mopen("BM.dat","w+");
        fichier_MB = mopen("MB.dat","w+");
        fprintfMat("BM.dat",BM);
        fprintfMat("MB.dat",MB);
        mclose("all");
    end
endfunction

```

```

endfunction

// routines d'execution
// -----
BoxMuller(1000,5,10,%f);
MarBray(1000,5,10,%f)

ExporterVersR (1000,0,1,%f);

// SimuDisque(10000,%t);
// SimuCercle(10000,%t);

//-----
//                Exercice 5
//-----

// Question 5A
// -----

// Fonction Beta(3,2)
function y = Beta(x)
    if (x<0 | x>1) then
        y = 0;
    else
        y = 12*(x^2)*(1-x);
    end
endfunction

// simulation de Beta(3,2)
function Ques5A (n, bol)
    grand('setgen', 'mt');
    c = 16/9;
    compteur = 1;
    if (bol==%t) then
        ReSimu = zeros(1,n); // allocation d'un vecteur de taille n
        while (compteur <= n)
            y = grand(1,1,'def'); // simu de g (loi unif)
            u = grand(1,1,'def'); // simu d'une uniforme independante de g
            if (c*u < Beta(y)) then
                ReSimu(compteur) = y;
                compteur = compteur+1;
            end
        end
        // exportation des donnees en vue d'un test sous R
        fichier_beta = mopen("Beta.dat", "w+");
        fprintfMat("Beta.dat", ReSimu);
        mclose("Beta.dat");
    end
endfunction

// Question 5B
// -----

function y = g(x)
    y = 0.5*exp(-abs(x));
endfunction

// simulation de g
function resultat = SimuG(n)
    grand('setgen', 'mt');
    u = grand(1,n,'def');
    epsilon = (u < 0.5)*2-1;
    y = Ques2A (n,0.5);
    resultat = 0.5*epsilon.*y;
endfunction

// densite de la loi normale centree reduite
function y = N(x)

```

```

    y = (exp(-x^2/2))/(sqrt(2* %pi));
endfunction

// trace optionnel des courbes
function courbes (n, bol)
    c = (2*exp(0.5))/(sqrt(2* %pi));
    x = [-n:0.1:n];
    if (bol==%t) then
        xbas();
        subplot(1,2,1);
        plot2d(x,g(x),style=[2]);
        plot2d(x,N(x),style=[3]);
        xtitle("fonctions f et g");
        subplot(1,2,2);
        plot2d(x,c*g(x),style=[2]);
        plot2d(x,N(x),style=[3]);
        xtitle("fonctions f et c*g");
    end
endfunction

// Question 5B : simulation d'une loi normale centree reduite
function Ques5B (n, bol)
    grand('setgen', 'mt');
    c = (2*exp(0.5))/(sqrt(2* %pi));
    compteur = 1;
    if (bol==%t) then
        ReSimu = zeros(1,n); // allocation d'un vecteur de taille n
        while (compteur <= n)
            y = SimuG(1); // simu de g
            u = grand(1,1,'def'); // simu d'une uniforme independante de g
            if (c*u*g(y) < N(y)) then
                ReSimu(compteur) = y;
                compteur = compteur+1;
            end
        end
    end
    // exportation des donnees en vue d'un test sous R
    fichier_Normale = mopen("Normale.dat", "w+");
    fprintfMat("Normale.dat", ReSimu);
    mclose("Normale.dat");
end
endfunction

// routines d'execution
courbes(10, %f); // traces de f et g et de f et c*g
Ques5A (1000,%f);
Ques5B (1000,%f);

//-----
//                               Exercice 6
//-----

// Question 6A
// -----
function resultat = LogNormale (n,mu,sigma2)
    Normale = MarBray(n,mu,sigma2,%t);
    resultat = exp(Normale(1,:));
endfunction

// Question 6C
// -----
function resultat = SimuS (n,S0,mu,sigma2,T)
    mu = (mu-sigma2/2)*T;
    sigma2 = sigma2*T;
    resultat = S0*LogNormale(n,mu,sigma2);
endfunction

// Question 6D
// -----

```

```

function Call (S0,r,sigma2,T,K,bol)
    if (bol==%t) then
        m = 5000; // valeur max du nombre de simulations (multiple de 100)
        h = 100; // pas
        abscisse = [10:h:m];
        S = SimuS(abscisse(length(abscisse)),S0,r,sigma2,T);
        for i = 1:length(abscisse)
            S_i = S(1:abscisse(i));
            moyenne = mean(exp(-r*T)*max(S_i-K,0));
            ecart_type = st_deviation(exp(-r*T)*max(S_i-K,0));
            down = moyenne - (1.96*ecart_type)/(sqrt(abscisse(i)));
            up = moyenne + (1.96*ecart_type)/(sqrt(abscisse(i)));
            IC(:,i) = [up ; moyenne ; down];
        end
        plot2d(abscisse,IC(1,:),style=[1]);
        plot2d(abscisse,IC(2,:),style=[2]);
        plot2d(abscisse,IC(3,:),style=[1]);
        disp("Pricing du Call");
        disp("-----");
        disp(moyenne);
    end
endfunction

// Question 6E (quasi Monte Carlo)
// -----

// fonction de Box Muller avec une suite a discrepance faible (Faure)
function resultat = BoxMuller_QMC (n,moyenne,variance,bol)
    if (bol==%t) then
        suite = Faure(5,n,2,2);
        u = suite(:,1)';
        v = suite(:,2)';
        x = sqrt(-2*log(u)).*cos(2*pi*v);
        y = sqrt(-2*log(u)).*sin(2*pi*v);
        // on prend en compte les parametres de la loi normale
        x = x*sqrt(variance) + moyenne;
        y = y*sqrt(variance) + moyenne;
        resultat = [x;y]; // valeur retournee par la fonction
    else
        resultat = 0; // eviter les erreurs d'execution si bol==%f
    end
endfunction

// on retrouve les memes fonctions qu'aux questions precedentes
// mais adaptees a BoxMuller_QMC

function resultat = LogNormale_QMC (n,mu,sigma2)
    Normale = BoxMuller_QMC(n,mu,sigma2,%t);
    resultat = exp(Normale(1,:));
endfunction

function resultat = SimuS_QMC (n,S0,mu,sigma2,T)
    mu = (mu-sigma2/2)*T;
    sigma2 = sigma2*T;
    resultat = S0*LogNormale_QMC(n,mu,sigma2);
endfunction

function Call_QMC (S0,r,sigma2,T,K,bol)
    if (bol==%t) then
        m = 5000; // valeur max du nombre de simulations (multiple de 100)
        h = 100; // pas
        abscisse = [10:h:m];
        S = SimuS_QMC(abscisse(length(abscisse)),S0,r,sigma2,T);
        for i = 1:length(abscisse)
            S_i = S(1:abscisse(i));
            moyenne = mean(exp(-r*T)*max(S_i-K,0));
            ecart_type = st_deviation(exp(-r*T)*max(S_i-K,0));
            down = moyenne - (1.96*ecart_type)/(sqrt(abscisse(i)));
            up = moyenne + (1.96*ecart_type)/(sqrt(abscisse(i)));
            IC(:,i) = [up ; moyenne ; down];
        end
    end
endfunction

```

```

        end
        plot2d(abscisse,IC(1,:),style=[1]);
        plot2d(abscisse,IC(2,:),style=[2]);
        plot2d(abscisse,IC(3,:),style=[1]);
        disp("Pricing du Call");
        disp("-----");
        disp(moyenne);
    end
endfunction

// routines d'execution
Call(1,0.5,0.45^2,1,1.2,%f);
Call_QMC(1,0.5,0.45^2,1,1.2,%f);

//-----
//                Exercice 7
//-----

// Question 7A
// -----

// attention aux formats : MU : m*1 et SIGMA : m*m
function resultat = Ques6A (n,MU,SIGMA,bol)
    if (bol==%t) then
        m = length(MU);
        if (size(MU) == [m,1] & size(SIGMA) == [m m]) then
            for i=1:n
                B = chol(SIGMA); // donne la matrice de Cholesky de SIGMA
                Y = MarBray(m,0,1,%t);
                Y = Y(1,:);
                resultat(:,i) = MU+B*Y';
            end
            disp(resultat);
        else
            disp("-----");
            disp("dimensions incompatibles, saisir : ");
            disp("mu : n*1");
            disp("sigma : n*n");
            disp("-----");
            resultat = 0;
        end
    else
        resultat = 0;
    end
endfunction

// Question 7B
// -----

// theta : parametre de la loi exponentielle
function resultat = CopuleNormale (theta,SIGMA,bol)
    if (bol==%t) then
        dim = size(SIGMA);
        k = dim(1);
        B = chol(SIGMA);
        Y = MarBray(k,0,1,%t);
        Y = Y(1,:);
        Z = B*Y';
        UNIF = cdfnorf("PQ",Z,zeros(k,1),ones(k,1));
        EXP = (-log(UNIF))/theta;
        resultat=[UNIF EXP];
    //    disp(resultat);
    else
        resultat = 0;
    end
endfunction

// Question 7C
// -----

```

```

// theta : parametre de la loi exponentielle
// ddl : degres de liberte de la loi de Student
function resultat = CopuleStudent (theta,ddl,SIGMA,bol)
    if (bol==%t) then
        // 1er etape : simulation d'un vecteur de loi normal standard multivarie
        dim = size(SIGMA);
        k = dim(1);
        B = chol(SIGMA);
        Y = MarBray(k,0,1,%t);
        Y = Y(1,:);
        Z = B*Y';
        // 2eme etape : simulation d'une var du khi deux
        W = grand(1,1,'chi',ddl);
        // 3eme etape : simulation d'un vecteur de loi de Student multivarie
        Z = Z/(sqrt(W/ddl));
        // 4eme etape : vecteur de loi exponentielle
        UNIF = cdf("PQ",Z,ddl*ones(k,1));
        EXP = (-log(UNIF))/theta;
        resultat=[UNIF EXP];
    // disp(resultat);
    else
        resultat = 0;
    end
endfunction

// Addenta (graphiques)
// -----

// rho : argument d'une matrice SIGMA 2*2 hors diagonale
// theta : parametre de la loi exponentielle
// ddl : degres de liberte de la loi de Student
// SIGMA : matrice de correlation
// n : nombre de simulations

function Graphes (rho,theta,ddl,n,option)
    SIGMA = [1 rho ; rho 1];
    select option
        case 1 then // Copule normale, marginales uniformes
            for i = 1:n
                CopNorm = CopuleNormale(theta,SIGMA,%t);
                unif = CopNorm(:,1);
                point(:,i) = [unif(1);unif(2)];
            end
            plot2d(point(1,:),point(2,:),style=[-1]);
        case 2 then // Copule normale, marginales exponentielles
            for i = 1:n
                CopNorm = CopuleNormale(theta,SIGMA,%t);
                expo = CopNorm(:,2);
                point(:,i) = [expo(1);expo(2)];
            end
            plot2d(point(1,:),point(2,:),style=[-1]);
        case 3 then // Copule de Student, marginales uniformes
            for i = 1:n
                tCop = CopuleStudent(theta,ddl,SIGMA,%t);
                unif = tCop(:,1);
                point(:,i) = [unif(1);unif(2)];
            end
            plot2d(point(1,:),point(2,:),style=[-1]);
        case 4 then // Copule de Student, marginales exponentielles
            for i = 1:n
                tCop = CopuleStudent(theta,ddl,SIGMA,%t);
                expo = tCop(:,2);
                point(:,i) = [expo(1);expo(2)];
            end
            plot2d(point(1,:),point(2,:),style=[-1]);
        else
            break;
    end
endfunction

```

```
// tests :  
// MU = [0;0;0];  
// SIGMA = [1 0.3 0.5;0.3 1 0.4;0.7 0.5 1];  
// Ques6A (1,MU,SIGMA,%t);  
// CopuleNormale (0.1,SIGMA,%f);  
// CopuleStudent (0.1,10,SIGMA,%f);  
Graphes (0,1,100,1000,5);
```

```
#####
#          Sujet : simulations          #
#          Auteur : Michael Sibilleau  #
#          www.sibilleau.com           #
#          Date : 3 fevrier 2007       #
#####

# Definir chemin comme repertoire de travail
chemin <- "C:/Documents and Settings/Utilisateur/Mes documents/ENSI_ISFA/finances/
          MethodesNum/TD1"
setwd(chemin)

# -----
#      Exercice 1
# -----

Exo1 <- function (bol=FALSE)
{
  u <- read.table("u.dat")
  v <- read.table("v.dat")
  # test de Kolmogorov Smirnov sur u et v
  resultat <- rbind(ks.test(u, "punif"),ks.test(v, "punif"))
  if (bol==T)
  {
    print(resultat)
  }
}

# -----
#      Exercice 4
# -----

Exo4 <- function (bol=FALSE)
{
  BM <- read.table("BM.dat")
  MB <- read.table("MB.dat")
  BM1 <- as.numeric(BM[1,])
  BM2 <- as.numeric(BM[2,])
  MB1 <- as.numeric(MB[1,])
  MB2 <- as.numeric(MB[2,])
  if (bol==T)
  {
    par(mfrow=c(2,3), pty = 's')
    par(mfrow=c(1,2), pty = 's')

    # definitions des parametres graphiques
    initpar <- par ()
    par(bg="lightyellow", col.axis="blue", bty="n")
    layout(matrix(1:4,2,2))

    # qq plot pour Box Muller
    qqnorm(BM1,xlab="1er qqplot pour Box Muller")
    abline(mean(BM1),sd(BM1))
    qqnorm(BM2,xlab="2eme qqplot pour Box Muller")
    abline(mean(BM2),sd(BM2))

    # qq plot pour Marsaglia Bray
    qqnorm(MB1,xlab="1er qqplot pour Marasaglia Bray")
    abline(mean(MB1),sd(MB1))
    qqnorm(MB2,xlab="2eme qqplot pour Marasaglia Bray")
    abline(mean(MB2),sd(MB2))
    initpar # restauration des parametres graphiques par default

    # test de Shapiro-Wilks
    cat("\n Resultats pour Box Muller")
    cat("\n-----\n")
    resultat <- rbind(shapiro.test(BM1),shapiro.test(BM2))
    print(resultat)
    cat("\n \n Resultats pour Marasaglia Bray")
  }
}
```

```
        cat("\n-----\n")
        resultat <- rbind(shapiro.test(MB1),shapiro.test(MB2))
        print(resultat)
    }
}

# -----
# Exercice 5
# -----

Exo5 <- function (bol=FALSE)
{
  LoiBeta <- read.table("Beta.dat")
  LoiNormale <- read.table("Normale.dat")
  LoiBeta <- as.numeric(LoiBeta)
  LoiNormale <- as.numeric(LoiNormale)
  # test de Kolmogorov Smirnov
  resultat <- rbind(ks.test(LoiBeta,"pbeta",3,2),ks.test(LoiNormale,"pnorm",0,1))
  if (bol)
  {
    print(resultat)
    qqnorm(LoiNormale)
    abline(mean(LoiNormale),sd(LoiNormale))
  }
}

# routines d'execution
Exo1(F)
Exo4(F)
Exo5(F)
```