La Gestion des Risques Financiers

Exercices de Programmation

Thierry Roncalli

Janvier 2005

Table des matières

1 Principes de simulation de nombres aléatoires

1.1 Simulation d'une loi uniforme

La simulation de nombres aléatoires d'une loi uniforme est fondamentale dans les méthodes de Monte Carlo, car la simulation d'autres distributions est construite généralement à partir de celle-ci. La méthode la plus courante pour simuler une distribution uniforme $[0,1]$ est celle des générateurs congruentiels linéaires de suites de nombres pseudo-aléatoires.

Algorithme 1 *Etant donnés deux paramètres entiers A et M,*

1. *choisir une racine x_0 ;*
2. *à l'étape $n > 0$, poser $x_n = Ax_{n-1}[M]$ et retourner $u_n = \frac{x_n}{M}$.*

Ce type de générateur est périodique de période P strictement inférieure à M. Si l'on souhaite s'approcher le plus possible d'une distribution continue uniforme sur $[0,1]$, il convient donc de choisir A et M de sorte que la période P soit la plus grande possible.

Gauss utilise un générateur légèrement différent. Nous avons :

$$x_n = (Ax_{n-1}[M] + C)\,[M]$$

avec $A = 1664525$, $C = 1013904223$ et $M = 2^{32}$. Nous vérifions que la commande `rndu` et notre procédure `rnd_LCG` donnent les mêmes résultats.

```
new;

seed = 123; ns = 5;

rndseed seed;

rndu(ns,1);

rnd_LCG(seed,1664525,1013904223,2^32,ns);

proc rnd_LCG(x0,A,C,M,ns);
  local u,i;

  u = zeros(ns,1);
  for i(1,ns,1);
    x0 = (((A*x0) % M) + C) % M;
    u[i] = x0 / M;
  endfor;

  retp(u);
endp;
```

1.2 La technique de l'inversion de la fonction de répartition

Soit X une variable aléatoire de distribution F. Pour simuler X, nous exploitons la propriété $F(X) \sim U$ et nous avons :

$$x_i = F^{[-1]}(u_i)$$

1.2.1 Simulation d'une loi de Bernoulli

Pour simuler une loi de Bernoulli $\mathcal{B}(p)$, nous utilisons l'algorithme précédent en remarquant que $x_i = 0$ si $u_i \leq 1 - p$ et $x_i = 1$ si $u_i > 1 - p$.

```
new;

/*
** Loi de Bernoulli de paramètre p
*/

p = 0.3;
u = rndu(10,1);
b1 = u .>= (1-p);
b2 = u .<= p;

print "Simulation de 10 nombres aléatoires de loi B(p)";
print b1~b2;

ns = 100000;
u = rndu(ns,1);
b = u .<= p;
```

```
empirical_mean = meanc(b);
empirical_stddev = stdc(b);

theoretical_mean = p;
theoretical_stddev = sqrt( p .* (1-p) );

print (theoretical_mean ~ empirical_mean)|(theoretical_stddev ~ empirical_stddev);
```

1.2.2 Simulation d'une loi discrète

La simulation d'une loi discrète est légèrement plus compliquée que celle d'une loi de Bernoulli, puisqu'il faut inverser la fonction de répartition qui est en escalier. La méthode la plus simple est alors de chercher de façon séquentielle la marche d'escalier correspondante.

```
new;
library pgraph;

/*
**> simulation d'un LGD aléatoire :
**       0% avec une probabilité 25%
**       50% avec une probabilité 50%
**       70% avec une probabilité 15%
**      100% avec une probabilité 10%
*/

let xLGD = 0 0.5 0.7 1;
let PrLGD = 0.25 0.50 0.15 0.10;

PrCumLGD = cumsumc(PrLGD);

Ns = 100000;
u = rndu(Ns,1);
rndLGD = xLGD[1] * ((0 .< u) .and (u .<= PrCumLGD[1])) +
         xLGD[2] * ((PrCumLGD[1] .< u) .and (u .<= PrCumLGD[2])) +
         xLGD[3] * ((PrCumLGD[2] .< u) .and (u .<= PrCumLGD[3])) +
         xLGD[4] * ((PrCumLGD[3] .< u) .and (u .<= PrCumLGD[4]));

empirical_Pr = counts(rndLGD,xLGD)/rows(rndLGD);
print PrLGD~empirical_Pr;

graphset;
  call hist(rndLGD,4);
```

1.2.3 Simulation d'une loi exponentielle

Considérons un temps de défaut exponentiel de paramètre λ. Nous avons $F(t) = 1 - \exp(-\lambda t)$. Nous en déduisons que $F^{[-1]}(u) = -\ln(1-u)/\lambda$.

```
new;
library pgraph;

lambda = 100/1e4;
```

```
nFirmes = 100;
nSimul = 1000;

u = rndu(nFirmes,nSimul);
t = -ln(1-u)./lambda;
D = t .<= 1;
Freq = sumc(D);

graphset;
  call histp(Freq,20);

/*
** Autre méthode
**
*/

Pr = 1 - exp(-lambda*1);
D = u .<= Pr;
Freq2 = sumc(D);
```

Remarque 1 *Pour de nombreuses distributions continues, il n'existe pas d'expression analytique de l'inverse de la fonction de répartition. Dans ce cas, on peut utilisee une expression approchée ou on peut résoudre numériquement l'équation* $F(x_i) = u_i$.

1.3 La méthode des transformations

Soient X et $Y = g(X)$ deux variables aléatoires. Si X est plus facile à simuler que Y, on exploite la relation $y_i = g(x_i)$.

Remarque 2 *La technique de l'inversion de la fonction de répartition est un cas particulier de cette méthode.*

Dans l'exemple suivant, nous exploitons les relations entre les distributions $\mathcal{N}(0,1)$, χ^2_1, χ^2_ν et t_ν.

```
new;

nu = 5;
ns = 1000;

n = rndn(nu,ns);
rnd_chi_one = n^2;
rnd_chi_nu = sumc(rnd_chi_one);

n = rndn(ns,1);
rnd_t = n ./ sqrt(rnd_chi_nu/nu);
```

1.4 Simulation d'une variable aléatoire lognormale conditionnelle

Pour simuler une variable aléatoire lognormale, nous utilisons la relation suivante :

$$X = e^{\mu+\sigma N} \sim \mathcal{LN}(\mu, \sigma)$$

avec $N \sim \mathcal{N}(0,1)$.

```
/*
** Simulation de variables aléatoires lognormales
**
*/

new;
library pgraph;

mu = 4;
sigma = 1.25;

ns = 1000;
Loss = exp(mu + sigma*rndn(ns,1));

call histp(Loss,30);
```

La première méthode pour simuler la variable aléatoire conditionnelle $X \mid X > H$ consiste à simuler X et à ne garder que les valeurs de X lorsque celles-ci sont supérieures au seuil H.

```
/*
** Simulation de variables aléatoires lognormales
** conditionnelles
**
** Méthode brute
*/

new;
library pgraph;

mu = 4;
sigma = 1.25;

ns = 1000;
Loss = exp(mu + sigma*rndn(ns,1));

H = 500;                          /* Seuil de collecte */
Loss = selif(Loss,Loss .> H);

call histp(Loss,30);
```

La deuxième méthode est basée sur celle de l'inverse de la fonction de répartition. Soit $X \sim F$. Alors

$$G(x) = \Pr\{X = x \mid X > H\} = \frac{F(x) - F(H)}{1 - F(H)}$$

Soit $Z = X \mid X > H$. Nous avons :

$$G(Z) \sim U_{[0,1]}$$

Nous en déduisons que :

$$\begin{aligned} Z &= G^{-1}(U) \\ &= F^{-1}(F(H) + (1 - F(H))U) \end{aligned}$$

```
/*
** Simulation de variables aléatoires lognormales
** conditionnelles
**
** Méthode des transformations
**
*/

new;
library pgraph;

fn cdfLN(x,mu,sigma) = cdfn( (ln(x) - mu)./sigma);
/* CDF de la distribution lognormale */
fn cdfLNi(alpha,mu,sigma) = exp(mu + sigma .* cdfni(alpha));
/* Inverse de la CDF lognormale      */

mu = 4;
sigma = 1.25;
H = 500;                          /* Seuil de collecte */

ns = 1000;
Loss = cdfLNi( cdfLN(H,mu,sigma) + rndu(ns,1) .* (1-cdfLN(H,mu,sigma)), mu, sigma);

call histp(Loss,30);
```

2 Simulation de vecteurs gaussiens

Soient $\mathbf{X}$ un vecteur aléatoire gaussien de distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, B une matrice et a un vecteur. Nous avons :

$$a + B\mathbf{X} \sim \mathcal{N}\left(a, BB^{\top}\right)$$

La simulation de vecteurs gaussiens exploite cette relation. Notons que la simulation de $\mathcal{N}(\mathbf{0}, \mathbf{I})$ est immédiate si nous disposons d'un générateur de nombres aléatoires gaussiens.

2.1 Simulation de variables aléatoires gaussiennes

Nous rappelons que :

$$\mathcal{N}(\mu, \sigma) = \mu + \sigma\mathcal{N}(0, 1)$$

```
new;

cls;

for i(1,10,1);
  x = 2 + 3*rndn(1000,1);
```

```
  m = meanc(x);
  sigma = stdc(x);
  print m~sigma;
endfor;
```

2.2 La méthode de Cholesky

La méthode de Cholesky est une décomposition particulière de la matrice de covariance Σ telle que $\Sigma = PP^\top$ et P est une matrice triangulaire inférieure.

2.2.1 Le cas bivarié

Si nous avons :

$$\Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

alors nous vérifions que :

$$P = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix}$$

```
/*
** Simulation de nombres aléatoires gaussiens corrélés
**
*/

new;
library pgraph;

rho = 0.5;
ns = 1000;

X1 = rndn(ns,1);
X2 = rho * X1 + sqrt(1-rho^2) * rndn(ns,1);

X = X1~X2;   /* Création d'une matrice ns*2 par concaténation horizontale */

c = corrx(x);

print c;

graphset;                    /* Remise à zéro de la bibliothèque pgraph */
  _plctrl = -1;              /* Ne pas relier les points                */
  _pcross = 1;               /* Axes (0,0)                              */
  _pframe = 0;               /* Elimination du cadre                    */
  xy(x1,x2);
```

Nous reprenons le programme précédent en utilisant la procédure `chol` de Gauss.

```
/*
** Même programme que rndn1.prg
**
```

```
** Utilisation de la procédure de décomposition de Cholesky
**
*/

new;
library pgraph;

let rho[2,2] = 1.0 0.5
               0.5 1.0;

ns = 1000;

P = chol(rho);
X = rndn(ns,2)*P;

c = corrx(x);

print c;

graphset;                    /* Remise à zéro de la bibliothèque pgraph */
  _plctrl = -1;              /* Ne pas relier les points                */
  _pcross = 1;               /* Axes (0,0)                              */
  _pframe = 0;               /* Elimination du cadre                    */
  xy(x[.,1],x[.,2]);
```

2.2.2 Le cas multivarié

```
/*
** Simulation d'un vecteur gaussien corrélé
**
*/

new;
library pgraph;

let rho = 1.0
          0.5  1.0
          0.25 0.5 1.0;

rho = xpnd(rho);             /* Construction de la matrice carrée à partir
                             :: de la partie triangulaire inférieure
                             */

ns = 1000;

P = chol(rho);
X = rndn(ns,rows(rho))*P;

c = corrx(x);

print c;

graphset;                    /* Remise à zéro de la bibliothèque pgraph */
  _plctrl = -1;              /* Ne pas relier les points                */
```

```
  _pcolor = 12;              /* Couleur rouge                          */
  _pstype =  2;              /* Symbole carré                          */
  xyz(x[.,1],x[.,2],x[.,3]); /* Graphique 3D                           */
```

2.3 La méthode factorielle

Soit une matrice de corrélation $\Sigma = C(\rho)$. Nous considérons :

$$Z_i = \sqrt{\rho}X + \sqrt{1-\rho}\varepsilon_i$$

avec $X \sim \mathcal{N}(0,1)$, $\varepsilon_i \sim \mathcal{N}(0,1)$, $X \perp \varepsilon_i$ et $\varepsilon_i \perp \varepsilon_j$ $(i \neq j)$. Nous avons alors $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, C(\rho))$.

```
/*
** Simulation d'un vecteur gaussien corrélé
**
** Cas d'une corrélation constante
*/

new;
cls;

rho = 0.3;        /* Valeur du coefficient de corrélation */
N = 10;           /* Dimension du vecteur aléatoire       */

ns = 1000;

/*
:: méthode de décomposition de Chosleky
*/

rhoMatrix = diagrv(rho*ones(N,N),1);
P = chol(rhoMatrix);
U = rndn(ns,N);
Z1 = U*P;

/*
:: méthode factorielle
*/

X = rndn(ns,1);
Z2 = sqrt(rho) * X + sqrt(1-rho) * U;

print corrx(Z1);
print corrx(Z2);

/*
:: Comparaison des temps de calcul
*/

for i(10,1000,10);
  N = i;

  rhoMatrix = diagrv(rho*ones(N,N),1);
```

```
  time_begin = hsec;
  Z1 = rndn(ns,N)*chol(rhoMatrix);
  time_end = hsec;
  ct1 = (time_end - time_begin)/100;      /* Temps de calcul en secondes */

  time_begin = hsec;
  Z2 = sqrt(rho) * rndn(ns,1) + sqrt(1-rho) * rndn(ns,N);
  time_end = hsec;
  ct2 = (time_end - time_begin)/100;

  print /flush N~ct1~ct2~(ct1/miss(ct2,0));

endfor;
```

L'algorithme précédent se généralise dans le cas de plusieurs facteurs communs (§3.1.2 de [GDR]). Dans le cas d'une corrélation inter-sectorielle unique, nous rappelons que (§3.2 de [GDR]) :

$$Z_i = \sqrt{\rho}X + \sqrt{\rho_s - \rho}X_s + \sqrt{1 - \rho_s}\varepsilon_i \qquad \text{si } i \in s$$

```
new;
cls;

let rho_Intra = 0.30 0.40 0.50 0.60;
let rho_Inter = 0.20;

let index_Sector = 1 1 2 2 4 1 3 3 4 1 1 4 1 3 2 2 2;

I = rows(index_Sector);
S = rows(rho_Intra);
rhoMatrix = zeros(I,I);

for i1(1,I,1);
  for i2(1,I,1);

     if i1 == i2;
       rhoMatrix[i1,i2] = 1.0;
       continue;
     endif;

     if index_Sector[i1] == index_Sector[i2];
       rhoMatrix[i1,i2] = rho_Intra[index_Sector[i1]];
     else;
       rhoMatrix[i1,i2] = rho_Inter;
     endif;

  endfor;
endfor;

format 5,2;
print rhoMatrix;

ns = 1000;
Z1 = rndn(ns,I)*chol(rhoMatrix);
```

```
X = rndn(ns,1);
Xs = rndn(ns,S);
Z2 = sqrt(rho_Inter) * X +                                          /* Facteur commun     */
     sqrt(rho_Intra[index_Sector]'-rho_Inter) .* Xs[.,index_Sector] +   /* Facteur sectoriel  */
     sqrt(1-rho_Intra[index_Sector]') .* rndn(ns,I);                 /* Facteur spécifique */
```

3 Le risque opérationnel

3.1 Estimation de la distribution de sévérité

3.1.1 Principe du maximum de vraisemblance

```
/*
**> Estimation de pertes selon une distribution log-normale
**
*/

new;
library pgraph;

rndseed 123;          /* Simulation des memes pertes */

mu = 7;               /* Paramètres de la sévérité */
sigma = 1.75;

Ns = 1000;

u = mu + sigma*rndn(Ns,1);     /* Simulations de va N(mu,sigma) */
Losses = exp(u);               /* Simulations de va LN(mu,sigma) */

/*
**> Estimation classique
**
*/

data = Losses;

m1 = meanc(ln(data));
s1 = stdc(ln(data));

/*
*> Estimation par maximum de vraisemblance
**
*/

proc mlProc(theta);
  local mu,sigma,LogL;

  theta = sqrt(theta^2);     /* Pour être sûr que les paramètres sont positifs */
  mu = theta[1];
  sigma = theta[2];

  /* Vecteur des fonctions de vraisemblance */
```

```
  LogL = -0.5*ln(sigma^2) - 0.5*ln(2*pi) - ln(data) - 0.5*(ln(data)-mu)^2 ./ sigma^2;

  retp(Logl);
endp;

fn negLogL(theta) = -sumc(mlProc(theta));

_qn_PrintIters = 1;
_qn_RelGradTol = 1e-5;

sv = mu | sigma;
{theta_opt,fmin,grd,retcode} = Qnewton(&negLogL,sv);

m2 = theta_opt[1];
s2 = theta_opt[2];

print mu~m1~m2;
print sigma~s1~s2;
```

3.1.2 Estimation ML sans correction

```
/*
**> Estimation de pertes selon une distribution log-normale
**  lorsqu'il y a un threshold
**
*/

new;
library pgraph;

rndseed 123;          /* Simulation des memes pertes */

mu = 7;               /* Paramètres de la sévérité */
sigma = 1.75;

Ns = 10000;

u = mu + sigma*rndn(Ns,1);     /* Simulations de va N(mu,sigma)  */
Losses = exp(u);               /* Simulations de va LN(mu,sigma) */

Threshold = 1000;              /* Seuil de collecte              */
Losses_obs = selif(Losses, Losses .>= Threshold);   /* Pertes collectees */

/*
**> Estimation classique
**
*/

data = Losses_obs;

m1 = meanc(ln(data));
s1 = stdc(ln(data));

/*
```

```
*> Estimation par maximum de vraisemblance sans correction
**
*/

proc mlProc(theta);
  local mu,sigma,LogL;

  theta = sqrt(theta^2);     /* Pour être sûr que les paramètres sont positifs */
  mu = theta[1];
  sigma = theta[2];

  /* Vecteur des fonctions de vraisemblance */
  LogL = -0.5*ln(sigma^2) - 0.5*ln(2*pi) - ln(data) - 0.5*(ln(data)-mu)^2 ./ sigma^2;

  retp(Logl);
endp;

fn negLogL(theta) = -sumc(mlProc(theta));

_qn_PrintIters = 1;
_qn_RelGradTol = 1e-5;

sv = mu | sigma;
{theta_opt,fmin,grd,retcode} = Qnewton(&negLogL,sv);
theta_opt = abs(theta_opt);

m2 = theta_opt[1];
s2 = theta_opt[2];

print mu~m1~m2;
print sigma~s1~s2;
```

3.1.3 Estimation ML avec correction

```
/*
**> Estimation de pertes selon une distribution log-normale
**  lorsqu'il y a un threshold
**
*/

new;
library pgraph;

rndseed 123;          /* Simulation des memes pertes */

mu = 7;               /* Paramètres de la sévérité */
sigma = 1.75;

Ns = 10000;

u = mu + sigma*rndn(Ns,1);     /* Simulations de va N(mu,sigma)  */
Losses = exp(u);               /* Simulations de va LN(mu,sigma) */

Threshold = 1000;              /* Seuil de collecte              */
```

```
Losses_obs = selif(Losses, Losses .>= Threshold);   /* Pertes collectees */

/*
**> Estimation classique
**
*/

data = Losses_obs;

m1 = meanc(ln(data));
s1 = stdc(ln(data));

/*
*> Estimation par maximum de vraisemblance avec correction
**
*/

proc mlProc(theta);
  local mu,sigma,LogL;

  theta = sqrt(theta^2);     /* Pour être sûr que les paramètres sont positifs */
  mu = theta[1];
  sigma = theta[2];

  /* Vecteur des fonctions de vraisemblance */
  LogL = -0.5*ln(sigma^2) - 0.5*ln(2*pi) - ln(data) - 0.5*(ln(data)-mu)^2 / sigma^2 -
        ln(1 - cdfn( (ln(Threshold) - mu)/sigma) );

  retp(Logl);
endp;

fn negLogL(theta) = -sumc(mlProc(theta));

_qn_PrintIters = 1;
_qn_RelGradTol = 1e-5;

sv = mu | sigma;
{theta_opt,fmin,grd,retcode} = Qnewton(&negLogL,sv);
theta_opt = abs(theta_opt);

m2 = theta_opt[1];
s2 = theta_opt[2];

print mu~m1~m2;
print sigma~s1~s2;

/*
**> Estimation de la fréquence
**
*/

lambda_above_threshold = 100;      /* Dire d'expert */
lambda = lambda_above_threshold/(1 - cdfn( (ln(Threshold) - m2)/s2));
print lambda;
```

3.2 Estimation de la charge en capital

3.2.1 Exemple d'estimation d'une charge en capital

```
/*
**> Estimation de la charge en capital
**
*/

new;
library pgraph;

rndseed 123;

mu = 7;               /* Paramètres de la sévérité */
sigma = 1.95;

lambda = 100;         /* Paramètre de la fréquence */

Ns = 10000;           /* Nombre de simulations du Monte Carlo */

AgLosses = zeros(Ns,1);

i = 1;
do until i > Ns;
  Number_Losses = rndp(1,1,lambda);    /* Simulation du nombre de pertes annuelle */

  if Number_Losses == 0;
    i = i + 1;
    continue;
  endif;

  individual_Losses = exp( mu + sigma*rndn(Number_Losses,1) );
  AgLosses[i] = sumc(individual_Losses);

  i = i + 1;
endo;

moy_AgLosses = meanc(AgLosses);
moy_theorique = lambda * exp(mu + 0.5*sigma^2);
print moy_AgLosses|moy_theorique;

VaR = quantile(AgLosses,0.999);
EL = moy_theorique;
UL = VaR - EL;

CaR = VaR;
print "CaR (en millions d'euros) = " CaR/1e6;
```

3.2.2 Agrégation des charges en capital

```
/*
**> Estimation de la charge en capital aggrégée
**
*/
```

```
new;
library pgraph;

rndseed 123;

mu = 7 | 6;               /* Paramètres de la sévérité */
sigma = 1.75 | 1.85;

lambda = 100 | 120;         /* Paramètre de la fréquence */

Ns = 10000;           /* Nombre de simulations du Monte Carlo  */

AgLosses = zeros(Ns,2);

i = 1;
do until i > Ns;
  Number_Losses = rndp(2,1,lambda);    /* Simulation du nombre de pertes annuelle */

  if Number_Losses[1] /= 0;
    individual_Losses = exp( mu[1] + sigma[1]*rndn(Number_Losses[1],1) );
    AgLosses[i,1] = sumc(individual_Losses);
  endif;

  if Number_Losses[2] /= 0;
    individual_Losses = exp( mu[2] + sigma[2]*rndn(Number_Losses[2],1) );
    AgLosses[i,2] = sumc(individual_Losses);
  endif;

  i = i + 1;
endo;

CaR1 = quantile(AgLosses[.,1],0.999);
EL1 = lambda[1]*exp(mu[1]+0.5*sigma[1]^2);
UL1 = CaR1 - EL1;

CaR2 = quantile(AgLosses[.,2],0.999);
EL2 = lambda[2]*exp(mu[2]+0.5*sigma[2]^2);
UL2 = CaR2 - EL2;

CaR12 = CaR1 + CaR2;

/*
**> Si les types de risque sont inédpendants
**
*/

TotLosses = AgLosses[.,1] + AgLosses[.,2];
CaR = quantile(TotLosses,0.999);

print "CaR1+2 = " CaR12/1e6;
print "CaR = " CaR/1e6;
```

```
/*
**> Approximation Gaussienne
**
*/

rho = 0.0;

CaRgauss = EL1 + EL2 + sqrt(UL1^2 + 2*rho*UL1*UL2 + UL2^2);
print "CaR Gaussienne = " CaRgauss/1e6;
```

3.3 Analyse de scénarios

3.3.1 Calcul d'un temps de retour

```
new;

fn ReturnTime(Loss,lambda,mu,sigma) = 1 ./ (lambda .* (1 - cdfn((ln(Loss)-mu)./sigma)));

Loss = 1e6;  /* 1 million d'euros */

print ReturnTime(Loss,1000,6,1.5);
print ReturnTime(Loss,1000,8,1.9);
print ReturnTime(Loss,1000,8,2.0);
print ReturnTime(Loss,1000,8,2.5);
print;

Loss = 10e6;  /* 10 millions d'euros */

print ReturnTime(Loss,1000,6,1.5);
print ReturnTime(Loss,1000,8,1.9);
print ReturnTime(Loss,1000,8,2.0);
print ReturnTime(Loss,1000,8,2.5);
```

3.3.2 Procédure de calibration des temps de retour

```
/*
**> ScenarioAnalysis
**
*/

proc (4) = ScenarioAnalysis(x,d,w,lambda,sv);
  local theta,fmin,grd,retcode;

  if w == 0 or w == 1;
    w = ones(rows(x),1);
  endif;

  _ScenarioAnalysis_data = x~d~w;
  _ScenarioAnalysis_lambda = lambda;

  {theta,fmin,grd,retcode} = optmum(&_ScenarioAnalysis,sv);
  theta = sqrt(theta^2);
  if _ScenarioAnalysis_lambda == 0;
    lambda = theta[3];
```

```
  else;
    lambda = _ScenarioAnalysis_lambda;
  endif;

  d = 1./(lambda.*(1-cdfLN(x,theta[1],theta[2])));

  retp(theta[1],theta[2],lambda,d);
endp;

proc (1) = _ScenarioAnalysis(theta);
  local mu,sigma,lambda,u;

  theta = sqrt(theta^2);
  mu = theta[1];
  sigma = theta[2];
  if _ScenarioAnalysis_lambda == 0;
    lambda = theta[3];
  else;
    lambda = _ScenarioAnalysis_lambda;
  endif;

  u = _ScenarioAnalysis_data[.,2] -
      1./(lambda.*(1-cdfLN(_ScenarioAnalysis_data[.,1],mu,sigma)));

  retp( sumc(_ScenarioAnalysis_data[.,3] .* u^2) );
endp;
```

3.3.3 Exemple de calibration des temps de retour

```
new;
library gdr,pgraph,optmum;
gdrSet;

cls;
__output = 0;

mu = 9;
sigma = 2.0;
lambda = 500;

let x = 1e6 2.5e6 5e6 7.5e6 1e7 2e7;
d0 = 1./(lambda.*(1-cdfLN(x,mu,sigma)));
let d = 0.25 1 3 6 10 40;

sv = mu|sigma;
{mu,sigma,lambda,dt} = ScenarioAnalysis(x,d,1,lambda,sv);
print mu|sigma|lambda;

sv = sv|500;
{mu,sigma,lambda,dt} = ScenarioAnalysis(x,d,1./d^2,0,sv);
print mu|sigma|lambda;

iter = 1;
do until iter > 6;
```

```
  sv = mu|sigma|lambda;
  w = 1./dt^2;
  {mu,sigma,lambda,dt} = ScenarioAnalysis(x,d,w,0,sv);
  iter = iter + 1;
endo;

print mu|sigma|lambda;
```

4 Le risque de marché

4.1 Les portefeuilles linéaires

4.1.1 La VaR analytique

```
new;

let mu = 0.005 0.003 0.002;
let sigma = 0.020 0.030 0.010;

let cor = 1.0
          0.5 1.0
          0.25 0.60 1;
cor = xpnd(cor);
sigma = cor .* sigma' .* sigma;              // Matrice de covariance

let P0 = 244 135 315;                        // Valeur actuelle des actifs
let theta = 2 -1 1;                          // Composition du portefeuille

/*
:: Formule de la VaR analytique gaussienne
:: les facteurs de risque sont les rendements
*/

thetaStar = theta .* P0;
VaR = -thetaStar'*mu + cdfni(0.99)*sqrt(thetaStar'*sigma*thetaStar);
print VAR;
```

4.1.2 La VaR historique

```
new;

load Prices[251,3] = prix.dat;

let P0 = 244 135 315;
let theta = 2 -1 1;

VaR = gaussianVaR(Prices,P0,theta,0.99);
print VaR;

VaR = historicalVaR(Prices,P0,theta,0.99);
print VaR;

/*
**> gaussianVaR
```

```
**
*/

proc (1) = gaussianVaR(data,P0,theta,alpha);
  local r,mu,sigma,thetaStar,VaR;

  data = ln(data);
  r = packr(data - lag1(data));
  mu = meanc(r);
  Sigma = vcx(r);

  thetaStar = theta .* P0;
  VaR = - thetaStar'mu + cdfni(alpha) * sqrt(thetaStar'Sigma*thetaStar);

  retp(VaR);
endp;

/*
**> historicalVaR
**
*/

proc (1) = historicalVaR(data,P0,theta,alpha);
  local r,thetaStar,PnL,VaR;

  data = ln(data);
  r = packr(data - lag1(data));
  thetaStar = theta .* P0;
  PnL = r*thetaStar;                    /* PnL = sumc(r' .* theta .* P0) */
  VaR = -quantile(PnL,1-alpha);

  retp(VaR);
endp;
```

4.1.3 La VaR monte carlo

```
new;
library pgraph;

load Prices[251,3] = prix.dat;

let P0 = 244 135 315;
let theta = 2 -1 1;

data = ln(Prices);
r = packr(data - lag1(data));

mu = meanc(r);                          // vecteur des moyennes
Sigma = vcx(r);                         // matrice de covariance

ns = 100000;                            // Nombre de simulations
rs = mu' + rndn(ns,3)*chol(Sigma);      // Rendements simulés

Ps = P0' .* (1+rs);
```

```
PnL = Ps*theta - P0'theta;            // Simulation du PnL

VaR = -quantile(PnL,0.01);
print ftos(VaR,"Monte-Carlo VaR(0.99) = %lf",5,2);
thetaStar = theta .* P0;
VaR = -thetaStar'mu + cdfni(0.99)*sqrt(thetaStar'*Sigma*thetaStar);
print ftos(VaR,"Theoretical VaR(0.99) = %lf",5,2);

graphset;
  _pdate = ""; _pframe = 0; _pnum = 2;
  _pbartyp = 6~13;
  title("\214PnL");
  xlabel("\214variation (in euros)");
  ylabel("\214frequency (in %)");
  call histp(PnL,250);
```

4.2 Les portefeuilles exotiques

4.2.1 Le cas d'un seul facteur de risque

```
new;

proc EuropeanBS(S0,K,sigma,T,r);
  local w,d1,d2,P;

  w = sigma.*sqrt(T);
  d1 = ( ln(S0./K) + r.*T )./w + 0.5 * w;
  d2 = d1 - w;

  P = S0.*cdfn(d1) - K.*exp(-r.*T).*cdfn(d2);
  retp(P);
endp;

load Prices[251,3] = prix.dat;
S = Prices[.,1];
r = packr( ln(S) - lag1(ln(S)) );

S_t = 234.31;

K = 220;
tau = 90/365;
ir = 0.05;
sigma_t = 0.237;
C_t = EuropeanBS(S_t,K,sigma_t,tau,ir);
shock = 1e-4;
Delta_t = (EuropeanBS(S_t*(1+shock),K,sigma_t,tau,ir) - EuropeanBS(S_t,K,sigma_t,tau,ir))/(shock*S_t

theta1 = -1; theta2 = 10;                    // Composition du portefeuille
P_t = theta1*S_t + theta2*C_t;               // valeur actuelle du portefeuille

/*
**> VaR historique (le facteur de risque est la rendement de l'actif)
**
*/
```

```
S_tp1 = S_t .* (1+r);
C_tp1 = EuropeanBS(S_tp1,K,sigma_t,tau-1/365,ir);
P_tp1 = theta1*S_tp1 + theta2*C_tp1;
PnL = P_tp1 - P_t;
VaR_histo = -quantile(PnL,0.01);
print VaR_histo;

/*
**> VaR monte carlo
**
*/

n = 100000;
mu_r = meanc(r);
sigma_r = stdc(r);
rs = mu_r + rndn(n,1) * sigma_r;
S_tp1 = S_t .* (1+rs);
C_tp1 = EuropeanBS(S_tp1,K,sigma_t,tau-1/365,ir);
P_tp1 = theta1*S_tp1 + theta2*C_tp1;
PnL_MC = P_tp1 - P_t;
VaR_MC = -quantile(PnL_MC,0.01);
print VaR_MC;

/*
**> VaR delta
**
*/

PnL_Delta = (theta1+theta2*Delta_t)*S_t.*r;
VaR_Delta = -quantile(PnL_Delta,0.01);
print VaR_Delta;
```

4.2.2 Le cas de plusieurs facteurs de risque

```
new;

proc EuropeanBS(S0,K,sigma,T,r);
  local w,d1,d2,P;

  w = sigma.*sqrt(T);
  d1 = ( ln(S0./K) + r.*T )./w + 0.5 * w;
  d2 = d1 - w;

  P = S0.*cdfn(d1) - K.*exp(-r.*T).*cdfn(d2);
  retp(P);
endp;

load Prices[251,3] = prix.dat;
S = Prices[.,1];
rS = packr( ln(S) - lag1(ln(S)) );

load V[251,1] = vols.dat;
rV = packr( ln(V) - lag1(ln(V)) );
```

```
S_t = 244;

K = 250; tau = 1; ir = 0.05; sigma_t = 0.23;
C_t = EuropeanBS(S_t,K,sigma_t,tau,ir);

theta1 = -1; theta2 = 10;
P_t = theta1*S_t + theta2*C_t;

/*
**> un seul facteur de risque : S(t)
*/

S_tp1 = S_t .* (1+rS);
C_tp1 = EuropeanBS(S_tp1,K,sigma_t,tau-1/365,ir);
P_tp1 = theta1*S_tp1 + theta2*C_tp1;
PnL = P_tp1 - P_t;

VaR = -quantile(PnL,0.01);
print VaR;

/*
**> deux facteurs de risque : S(t) et sigma(t)
*/

S_tp1 = S_t .* (1+rS);
sigma_tp1 = sigma_t .* (1+rV);
C_tp1 = EuropeanBS(S_tp1,K,sigma_tp1,tau-1/365,ir);
P_tp1 = theta1*S_tp1 + theta2*C_tp1;
PnL2 = P_tp1 - P_t;

VaR = -quantile(PnL2,0.01);
print VaR;
```

4.3 La valorisation

4.3.1 Les options européennes

```
new;
library pgraph;

proc BlackScholes(S0,K,sigma,tau,r,call_or_put);
  local w,d1,d2,Price;

  w = sigma.*sqrt(tau);
  d1 = ( ln(S0./K) + r.*tau )./w + 0.5 * w;
  d2 = d1 - w;

  if lower(call_or_put) $== "put";
    Price = -S0.*cdfn(-d1) + K.*exp(-r.*tau).*cdfn(-d2);
  else;
    Price = S0.*cdfn(d1) - K.*exp(-r.*tau).*cdfn(d2);
  endif;
```

```
  retp(Price);
endp;


vol = 25/100;          /* Volatilité = 25%                */
T = 3/12;              /* Maturité = 3 mois               */
S0 = 100;              /* Prix actuel du sous-jacent = 100 */
r = 0.05;

K = S0;                /* Option ATM (at-the-money)       */

BS = BlackScholes(S0,K,vol,T,r,"call");

rndseed 123456;

ns = 1000;

WT = sqrt(T) * rndn(ns,1);
ST = S0 * exp( (r-0.5*vol^2)*T + vol*WT);

PayOff = (ST - K) .* (ST .> K);              /* Payoff = max( S(T) - K, 0) */
Cs = exp(-r*T) * PayOff;

simul = seqa(1,1,ns);
stderr = stdc(Cs);
mcBS = cumsumc(Cs) ./ simul;
mcSTD = stderr ./ sqrt(simul);

graphset;
  _pdate = 0;                    /* Suppression de la date                    */
  _pframe = 0;                   /* Suppression du cadre                      */
  ytics(0,10,2,1);               /* Echelle de l'axes des Y                   */
  _pline = 1~3~0~BS~ns~BS~1~15~5; /* Trait horizontal indiquant le vrai prix BS */

  xy(simul,mcBS + (-1.96~0~1.96) .* mcSTD);
```

4.3.2 Les options basket

```
/*
** Valorisation d'une option sur basket
**
*/

new;
library pgraph;

let S0 = 100 100 100 100;
let vol = 0.25 0.30 0.25 0.40;
T = 6/12;
r = 0.05;

rndseed 123456;

ns = 50000;
```

```
let rho = 1.0
          0.5  1.0
          0.25 0.5  1.0
          0.10 0.25 0.5 1.0;
rho = xpnd(rho);

epsilon = chol(rho)'*rndn(4,ns);

WT = sqrt(T) * epsilon;
ST = S0 .* exp( (r-0.5*vol^2)*T + vol .* WT);
ST = ST';

PayOff = (ST[.,1] - ST[.,2] + ST[.,3] - ST[.,4]);
PayOff = PayOff .* (PayOff .> 0);

Cs = exp(-r*T) * PayOff;

simul = seqa(1,1,ns);
stderr = stdc(Cs);
mcBS = cumsumc(Cs) ./ simul;
mcSTD = stderr ./ sqrt(simul);

graphset;
  _pdate = 0;                    /* Suppression de la date                    */
  _pframe = 0;                   /* Suppression du cadre                      */
  ytics(8,16,2,1);               /* Echelle de l'axes des Y                   */

  xy(simul,mcBS + (-1.96~0~1.96) .* mcSTD);
```

4.3.3 Les options path-dependent

5 Le risque de crédit

5.1 Simulation de temps de défaut exponentiels

```
/*
** Simulation de temps de défaut exponentiels
**
*/

new;
library pgraph;

let lambda = 100;           /* lambda = 100 bp */
lambda = lambda / 1e4;

ns = 1000;
u = rndu(ns,1);
tau = -ln(1-u)/lambda;
```

```
graphset;
  _pnum = 2; _pdate = 0; _pframe = 0;
  xlabel("Temps de defaut");
  call histp(tau,seqa(0,20,30));
```

5.2 Simulation de temps de défaut corrélés

```
/*
** Simulation de min(tau_1,...,tau_N) avec une dépendance correspondant à une copule gaussienne
**
*/

new;
library pgraph;

rndseed 123;                            // Initialisation du générateur pseudo-aléatoire

N = 5;
lambda = 200/1e4;
rho = 0.25;
rhoMatrix = diagrv(rho*ones(N,N),1);   // Matrice des corrélations canoniques

Ns = 1000;
u = cdfn(rndn(ns,n)*chol(rhoMatrix));         // Copule gaussienne

t = -ln(1-u)./lambda;                         /*
                                              :: Simulation de la matrice Ns*N des
                                              :: temps de défaut corrélés
                                              */

tmin = minc(t');                              // Simulation de min(tau_1,...,tau_N)

graphset;
  _pnum = 2; _pdate = 0; _pframe = 0;
  xlabel("min(tau_1,...,tau_N)");
  call histp(tmin,25);
```

5.3 La formule Bâle II pour mesurer le capital

```
/*
** Mise en évidence de la formule de Bâle II
**
*/

new;
library pgraph;

N = 100;                        // Nombre de firmes
EaD = 100;                       // EaD
muLGD = 0.5;                     // E[LGD]
sigmaLGD = 0.20;                 // sigma[LGD]
PD = 0.2;                       // PD

EaD = EaD .* ones(N,1); muLGD = muLGD .* ones(N,1);
```

```
sigmaLGD = sigmaLGD .* ones(N,1); PD = PD .* ones(N,1);

rho = 0.25;
alpha = seqa(0.01,0.01,99);

K = sumc(EAD .* muLGD .* cdfn( (cdfni(PD)-sqrt(rho).*cdfni(1-alpha'))./sqrt(1-rho) ));

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6; _pcolor = 9; _plwidth = 12;
  ylabel("Capital");
  xlabel("alpha");
  xy(alpha,K);

Ns = 10000;

/*
** Simulation des défauts corrélés
*/

Z = sqrt(rho)*rndn(1,Ns) + sqrt(1-rho)*rndn(N,Ns);
D = (Z .<= cdfni(PD));

/*
** Simulation des LGD aléatoires
*/

a = -muLGD + (muLGD^2) .* (1-muLGD) ./ (sigmaLGD^2);
b = a .* (1-muLGD)./muLGD;
rndLGD = rndBeta(N,Ns,a,b);

Loss = sumc(EaD .* rndLGD .* D);
Q = quantile(Loss,alpha);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|3; _pcolor = 9|12; _plwidth = 10;
  xlabel("Perte");
  ylabel("CDF");
  xy(K~Q,alpha);
```

5.4 La notion de granularité

```
/*
** Mise en évidence de la granularité
**
*/

new;
library pgraph;

N = 100;                        // Nombre de firmes
EaD = 1;                        // EaD
muLGD = 0.5;                     // E[LGD]
```

```
sigmaLGD = 0.20;                // sigma[LGD]
PD = 0.2;                      // PD

EaD = EaD .* ones(N,1); muLGD = muLGD .* ones(N,1);
sigmaLGD = sigmaLGD .* ones(N,1); PD = PD .* ones(N,1);

rho = 0.25;
alpha = seqa(0.01,0.01,99);

K = sumc(EAD .* muLGD .* cdfn( (cdfni(PD)-sqrt(rho).*cdfni(1-alpha'))./sqrt(1-rho) ));

Ns = 10000;

/*
** Non-granularité du portefeuille
**
*/

EaD[1] = N;

/*
** Simulation des défauts corrélés
*/

Z = sqrt(rho)*rndn(1,Ns) + sqrt(1-rho)*rndn(N,Ns);
D = (Z .<= cdfni(PD));

/*
** Simulation des LGD aléatoires
*/

a = -muLGD + (muLGD^2) .* (1-muLGD) ./ (sigmaLGD^2);
b = a .* (1-muLGD)./muLGD;
rndLGD = rndBeta(N,Ns,a,b);

Loss = sumc(EaD .* rndLGD .* D);
Q = quantile(Loss,alpha);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|3; _pcolor = 9|12; _plwidth = 10;
  xlabel("Perte");
  ylabel("CDF");
  xy(K~Q,alpha);
```

5.5 Simulation du nombre de défauts annuel

```
/*
** Simulation du nombre de défauts annuel
**
*/

new;
```

```
library pgraph;

rho = 0.2;

let lambda = 20 100 200;          /* lambda = 20 bp, 100 bp & 200 bp */
lambda = lambda / 1e4;
let nFirms = 200 400 150;         /* Composition du portefeuille     */

lambda = lambda[1]*ones(nFirms[1],1) |
         lambda[2]*ones(nFirms[2],1) |
         lambda[3]*ones(nFirms[3],1) ;

T = 1;                            /* Horizon = 1 an                  */
PD = 1 - exp(-lambda .* T);       /* Probabilités de défaut          */

ns = 10000;
X = rndn(1,ns);
epsilon = rndn(sumc(nFirms),ns);
Z = sqrt(rho) * X + sqrt(1-rho) * epsilon;    /* Modèle de Merton à un facteur */

D = cdfn(Z) .<= PD;                                /* Indicatrices de défaut       */
ND = sumc(D);                                      /* Nombre de défauts en [0,T]   */

print "Nombre de défauts moyens :     " meanc(ND);
print "Nombre de défauts théoriques : " sumc(PD);   /* Ne dépend pas de la correl.  */

graphset;
  _pnum = 2;
  xlabel("Nombre de defauts annuel");
  call histp(ND,seqa(0,1,100));
```

6 Les dérivés de crédit

6.1 Pricing d'un CDS

```
new;

Maturity = 5;
Nominal = 1e6;

Lambda = 100/1e4;   // 100 bp
Margin_CDS = 80/1e4;    // 80 bp
RR = 0.40;

r = 0.03;           // Taux d'intérêt
Ns = 1e5;           // Nombre de simulations

Tm = seqa(0.25,0.25,Maturity/0.25);       // Echeancier trimestriel
dTm = 0.25;

B0Tm = exp(-r*Tm);                           // B(t0,tm)

tau = -ln(rndu(1,Ns))/lambda;
```

```
JF = sumc(dTm .* B0Tm .* (Tm .<= tau));                          // Premium leg
tau = tau';
JV = (1-RR) .* exp(-r*tau) .* (tau .<= Maturity);                // Default leg

CDS = meanc(Nominal * Margin_CDS * JF) - meanc(Nominal .* JV);

print CDS;
print "Prix du CDS = " CDS "euros";

ATM = meanc(JV)/meanc(JF);
print "Spread du CDS = " 1e4*ATM "bp";

s = lambda*(1-RR);
print "Spread du CDS (formule approchee) = " 1e4*s "bp";
```

6.2 Pricing d'un First-to-Default

```
new;
library pgraph;

Maturity = 5;

Lambda = 100|150|80|200;
Lambda = lambda/1e4;
RR = 0.40;

r = 0.03;           // Taux d'intérêt
Ns = 3e4;           // Nombre de simulations

Tm = seqa(0.25,0.25,Maturity/0.25);       // Echeancier trimestriel
dTm = 0.25;

B0Tm = exp(-r*Tm);                           // B(t0,tm)

rho = seqa(0,1/20,21);
F2D = zeros(rows(rho),1);
ATM = zeros(rows(rho),1);

N = rows(Lambda);

i = 1;
do until i > rows(rho);
  rndseed 123;
  if i == rows(rho);
    u = rndu(1,Ns) .* ones(N,1);
  else;
    u = cdfn(sqrt(rho[i])*rndn(1,Ns) + sqrt(1-rho[i])*rndn(N,Ns));
  endif;
  tau = -ln(1-u)./lambda;
  tmin = minc(tau);

  JF = sumc(dTm .* B0Tm .* (Tm .<= tmin'));                      // Premium leg
  JV = (1-RR) .* exp(-r*tmin) .* (tmin .<= Maturity);            // Default leg
```

```
  ATM[i] = meanc(JV)/meanc(JF);

  i = i + 1;
endo;

s = 1e4*lambda .* (1-RR);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  xlabel("Correlation de defaut");
  ylabel("Spread du F2D (en bp)");
  xtics(0,1,0.2,2);
  _pline = 1~6~0~maxc(s)~1~maxc(s)~1~12~10 |         // max(spread) <---> rho = 0%
           1~6~0~sumc(s)~1~sumc(s)~1~12~10 ;         // sum(spread) <---> rho = 100%
  xy(rho,1e4*ATM);
```

6.3 Pricing d'un CDO

```
new;
library pgraph;

Maturity = 5;

N = 100;

Nominal = 100;
Lambda = 50/1e4;
RR = 0.40;

r = 0.03;           // Taux d'intérêt
Ns = 1e3;           // Nombre de simulations

Tm = seqa(0.25,0.25,Maturity/0.25);       // Echeancier trimestriel
dTm = 0.25;

B0Tm = exp(-r*Tm);                           // B(t0,tm)

rho = 0.5;
u = cdfn(sqrt(rho)*rndn(1,Ns) + sqrt(1-rho)*rndn(N,Ns));
tau = -ln(1-u)./lambda;

Lower_Tranche = 0.05;
Upper_Tranche = 0.10;

MaxLoss = N * Nominal * (1-RR);
Lower_Tranche = Lower_Tranche * MaxLoss;
Upper_Tranche = Upper_Tranche * MaxLoss;

Loss = zeros(rows(Tm),Ns);
for i(1,Ns,1);
  Loss[.,i] = sumc( Nominal * (1-RR) * (tau[.,i] .<= Tm') );
endfor;

fn _max_(x,y) = x .* (x .> y) + y .* (x .<= y);
```

```
fn _min_(x,y) = x .* (x .< y) + y .* (x .>= y);

/*
**> Tranche Equity
*/

N1 = Lower_Tranche - _min_(Loss,Lower_Tranche);
Loss1 =  missrv(lagn(N1,1),Lower_Tranche) - N1;
JF = sumc(dTm .* B0Tm .* N1);
JV = sumc(B0Tm .* Loss1);
Spread = meanc(JV)/meanc(JF);
print "Spread de la tranche equity : " 1e4*Spread " bp";

/*
**> Tranche Mezzanine
*/

N2 = _max_((Upper_Tranche-Lower_Tranche) - _max_(Loss-Lower_Tranche,0),0);
Loss2 =  missrv(lagn(N2,1),Upper_Tranche-Lower_Tranche) - N2;
JF = sumc(dTm .* B0Tm .* N2);
JV = sumc(B0Tm .* Loss2);
Spread = meanc(JV)/meanc(JF);
print "Spread de la tranche mezzanine : " 1e4*Spread " bp";

/*
**> Tranche Senior
*/

N3 = _max_((maxLoss-Upper_Tranche) - _max_(Loss-Upper_Tranche,0),0);
Loss3 =  missrv(lagn(N3,1),maxLoss-Upper_Tranche) - N3;
JF = sumc(dTm .* B0Tm .* N3);
JV = sumc(B0Tm .* Loss3);
Spread = meanc(JV)/meanc(JF);
print "Spread de la tranche senior : " 1e4*Spread " bp";
```

7 Gestion de portefeuille

7.1 Allocation de portefeuille

7.1.1 Optimisation risque/rendement

```
new;

let sigma  = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;
let mu     = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;

sharpe = mu ./ sigma;

mu = mu / 100; sigma = sigma / 100;

n = rows(mu); rho = eye(n);

cov = sigma .* sigma' .* rho;
```

```
Q = 2*cov; R = mu;

A = ones(1,n); B = 1; C = 0; D = 0; bounds = ones(n,1) .* (0~1);

sv = ones(n,1)/n;

phi = 0.1; {x,u1,u2,u3,u4,retcode} =
QProg(sv,Q,phi*R,A,B,C,D,bounds);

er = x'mu; vol = sqrt(x'cov*x);

print 100*(er~vol);
```

7.1.2 Construction de la frontière efficiente

```
new; library pgraph;

let sigma  = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;
let mu     = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;

sharpe = mu ./ sigma;

mu = mu / 100; sigma = sigma / 100;

n = rows(mu); rho = eye(n);

cov = sigma .* sigma' .* rho;

Q = 2*cov; R = mu;

A = ones(1,n); B = 1; C = 0; D = 0; bounds = ones(n,1) .* (0~1);

sv = ones(n,1)/n;

phi = seqa(-2,0.005,601); nPhi = rows(phi); er = zeros(nPhi,1); vol
= zeros(nPhi,1);

i = 1; do until i > nPhi;
  {x,u1,u2,u3,u4,retcode} = QProg(sv,Q,phi[i]*R,A,B,C,D,bounds);
  er[i] = x'mu;
  vol[i] = sqrt(x'cov*x);
  i = i + 1;
endo;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  xlabel("\216Risk");
  ylabel("\216Expected Return");
  xy(100*vol,100*er);
```

7.1.3 Détermination du portefeuille optimal

```
new; library pgraph;
```

```
let sigma  = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;
let mu     = 5.00   1.00    5.00    2.50    5.00    7.50    10.00;

sharpe = mu ./ sigma;

mu = mu / 100; sigma = sigma / 100;

n = rows(mu); rho = eye(n);

cov = sigma .* sigma' .* rho;

Q = 2*cov; R = mu;

A = 0; B = 0; C = -ones(1,n); D = -1; bounds = ones(n,1) .* (0~1);

sv = ones(n,1)/n;

Target_Vol = 1/100;

phi = Bisection(&diff_vol,0,1,1e-10); {x,u1,u2,u3,u4,retcode} =
QProg(sv,Q,phi*R,A,B,C,D,bounds); er = x'mu; vol = sqrt(x'cov*x);

print "Allocation = " 100*x; print "Allocation non risquée = "
100*(1-sumc(x)); print "Expected Return = " 100*er; print
"Volatility = " 100*vol;

proc diff_vol(phi);
  local x,u1,u2,u3,u4,retcode;
  local vol;

  {x,u1,u2,u3,u4,retcode} = QProg(sv,Q,phi*R,A,B,C,D,bounds);
  vol = sqrt(x'cov*x);

  retp(vol-Target_Vol);
endp;

proc Bisection(f,a,b,tol);
  local f:proc;
  local ya,yb,c,yc,e;

  ya = f(a);
  yb = f(b);

  if ya*yb > 0;
    retp(error(0));
  endif;

  if ya == 0;
    retp(a);
  endif;

  if yb == 0;
    retp(b);
  endif;
```

```
  if ya < 0;
    do while maxc(abs(a-b)) > tol;
      c = (a+b)/2;
      yc = f(c);
      e = yc < 0;
      a = e*c + (1-e)*a;
      b = e*b + (1-e)*c;
    endo;
  else;
    do while maxc(abs(a-b)) > tol;
      c = (a+b)/2;
      yc = f(c);
      e = yc > 0;
      a = e*c + (1-e)*a;
      b = e*b + (1-e)*c;
    endo;
  endif;

  c = (a+b)/2;

  retp(c);
endp;
```

7.2 Calcul du beta

```
new; library pgraph;

cls;

varNames = SpreadSheetReadSA("cac.xls","a5:g5",1); data     =
SpreadSheetReadM("cac.xls","a7:g529",1);

indx_CAC = 2; indx_TOTAL = 3; indx_SANOFI = 4; indx_SG = 5; indx_FT
= 6; indx_DAX = 7;

P = data; R = packr(ln(P) - lag(ln(P)));

R_Total = R[.,indx_Total]; R_SANOFI = R[.,indx_SANOFI]; R_SG =
R[.,indx_SG]; R_CAC = R[.,indx_CAC];

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pcross = 1; _plctrl = -1; _pstype = 11;
  xy(100*R_Total,100*R_CAC);

T = rows(R_CAC); X = ones(T,1)~R_CAC; K = cols(X); Y = R_SG; coeffs
= invpd(X'X)*X'Y;

alpha = coeffs[1]; beta = coeffs[2];

U = Y - X*coeffs; TSS = sumc(Y^2); RSS = sumc(U^2); R2 = 1 -
RSS/TSS;
```

```
print "R2 = " R2;

sigma = stdc(U); covCoeffs = sigma^2 * invpd(X'X); stdCoeffs =
sqrt(diag(covCoeffs)); tstudent = (Coeffs - 0.0) ./ stdCoeffs;
pvalue = 2*cdftc(abs(tstudent),T-K);

print Coeffs~stdCoeffs~tstudent~pvalue;
```

7.3 Détermination de l'alpha et régression de style

```
new;
library pgraph;

cls;

loadm VL_FUND;
loadm VL_BENCHMARK;

loads NAME_FUND;
loads NAME_BENCHMARK;

/*
**> Test des dates
*/

d1 = VL_FUND[.,1];
d2 = VL_BENCHMARK[.,1];

if d1 /= d2;
  errorlog "error: dates do not match.";
endif;

indx = 1;
do until indx > rows(NAME_FUND);

  Name = NAME_FUND[indx];
  print "------------------------------";
  print "Analyse du fonds "$+Name;
  VL = VL_Fund[.,indx+1];
  B = VL_BENCHMARK[.,2:4];

  Y = ln(VL) - lag1(ln(VL));
  X = ln(B) - lag1(ln(B));
  data = packr(Y~X);
  Y = data[.,1]; X = data[.,2:4];
  T = rows(Y);

  K = cols(X);
  A = ones(1,K);
  B = 1;
  C = 0;
  D = 0;
  bnds = zeros(K,1)~ones(K,1);
```

```
  XX = X'X;
  XY = X'Y;
  sv = ones(K,1)/K;
  {beta,u1,u2,u3,u4,retcode} = Qprog(sv,2*XX,2*XY,A,B,C,D,bnds);
  u = y - x*beta;
  rss = sumc(u .* u);
  tss = sumc(y .* y);
  R2 = 1 - rss/tss;

  if R2 < 0.70;
    indx = indx + 1;
    continue;
  endif;

  print Name_BENCHMARK$~(""$+ftocv(beta,1,4));
  print "R2 = " R2;

  X = X[.,selif(1|2|3,beta .> 0.10)];

  X = ones(T,1)~X;
  XX = X'X;
  XY = X'Y;
  coeffs = invpd(XX)*XY;
  U = Y - X*coeffs;
  sigma = stdc(U);
  cov = sigma^2 * invpd(XX);
  stderr = sqrt(diag(cov));

  print "alpha = " coeffs[1];
  print "stderr = " stderr[1];

  indx = indx + 1;
endo;
```