GAUSS et la Finance

Thierry Roncalli

L.A.R.E., Université Montesquieu-Bordeaux IV, Avenue Léon Duguit, 33608 Pessac Cedex, France
e-mail : roncalli@montesquieu.u-bordeaux.fr
tél : 05.56.84.85.86 (poste 85.77)

Table des matières

Partie I
Quelques applications de GAUSS en finance

Les méthodes numériques sont devenues essentielles en finance. Elles peuvent paraître difficiles à mettre en œuvre. Parfois, l'efficacité des algorithmes employés est primordiale. Le financier est alors confronté au choix d'un langage de programmation. Celui de GAUSS, qui est un langage interprété, n'est pas forcément optimal. Certains problèmes sont par exemple plus efficacement résolus en Fortran ou en C. GAUSS est néanmoins suffisant pour de nombreux problèmes[1]. Dans le cas d'élaboration de programmes complexes, il se révèle être un outil complémentaire. La phase d'élaboration est souvent une phase d'expérimentation. Les points de départ et d'arrivée du problème sont parfaitement connus, mais le chemin que nous devons employer l'est beaucoup moins. GAUSS peut alors être très utile pour nous aider à trouver plus facilement (et surtout plus **rapidement**) ce chemin, car il permet de simplifier les opérations de programmation.

Cette facilité de programmation et la simplicité du langage expliquent le succès de GAUSS auprès des universitaires. Son emploi dans les cours qui font appel à des techniques numériques améliore la qualité pédagogique des enseignements. A l'origine, nous avons élaboré TSM dans cet esprit. De même, la plupart des procédures de la partie "Méthodes numériques en finance" ont été développées pour illustrer le cours *Finance Stochastique : Application à la valorisation des actifs contingents* en maîtrise d'Économétrie et en 2ème année de Magistère d'Économie et Finance Internationales à l'Université Montesquieu-Bordeaux IV.

1 Méthodes numériques en finance

1.1 Les processus stochastiques

1.1.1 Les processus de Poisson

Les processus de Poisson permettent d'étudier des phénomènes discrets. Ils sont actuellement utilisés par les financiers pour la modélisation des actifs financiers (et la prise en compte de sauts) et l'extraction d'information sur les marchés d'option. Avant d'aborder les processus de diffusion avec saut, il est intéressant de mettre en évidence certaines propriétés d'un processus de Poisson. Il existe plusieurs définitions[2] pour le caractériser. Nous dirons que N_t est un processus de Poisson d'intensité λ si la variable aléatoire $N_{t+1} - N_t$ suit une loi de Poisson de paramètre λ. Si nous ne disposons pas d'un générateur de nombres aléatoires pour la loi de Poisson, nous pouvons considérer une approximation par la loi uniforme pour simuler un processus de Poisson. La procédure `Poisson` implémente cette méthode.

Poisson

■ **Objet**
Simulation de K processus de Poisson $\{N_t, t_0 \leq t \leq T\}$ d'intensité λ.

■ **Syntaxe**
{ti,N} = Poisson(lambda,t0,T,L,K);

■ **Entrées**

lambda	scalaire, paramètre λ.
t0	scalaire, t_0.
T	scalaire, T.
L	scalaire, nombre de points de discrétisation de l'intervalle de temps $[t_0, T]$.
K	scalaire, nombre de simulations.

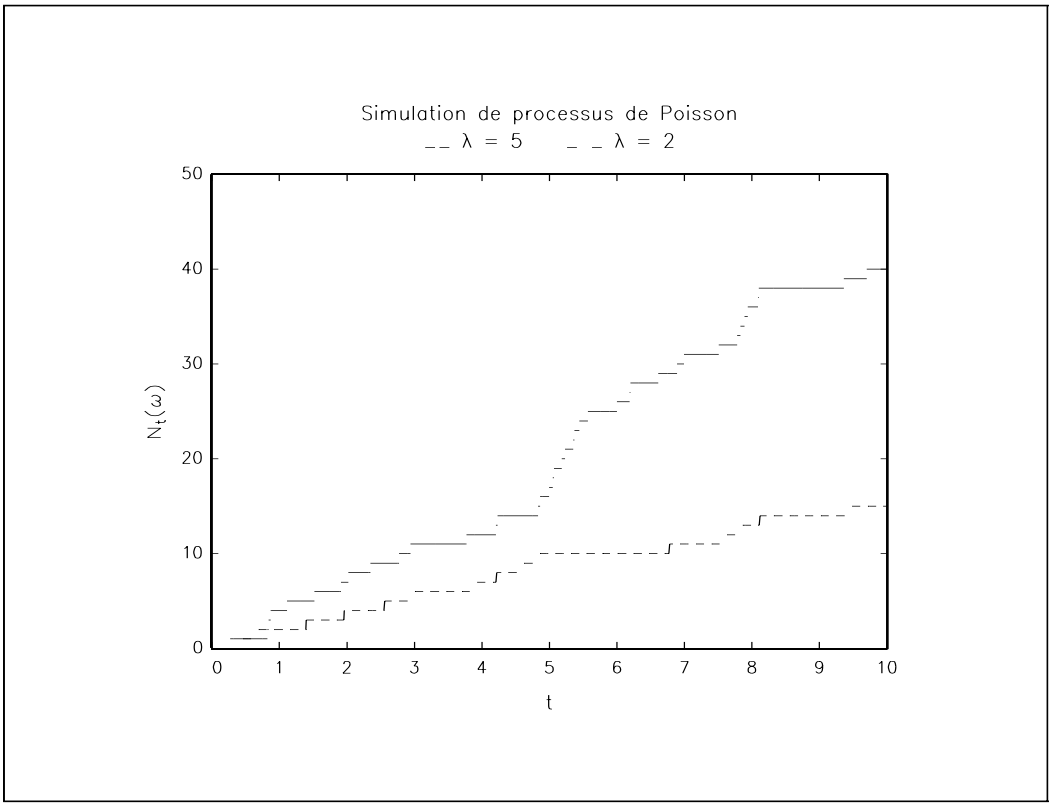
■ **Sorties**

ti	vecteur $(L+1) \times 1$, temps t_i.
N	matrice $(L+1) \times K$, processus de poisson N_{t_i} simulés.

■ **Remarque**
Le i-ième processus simulé correspond au vecteur `N[.,i]`.

Voici le code de la procédure `Poisson` :

[1] Pour information, nous avons effectué l'ensemble des calculs avec un Pentium 100.
[2] voir par exemple, Bouleau, N. [1988], Processus Stochastiques et Applications, Hermann, Paris.



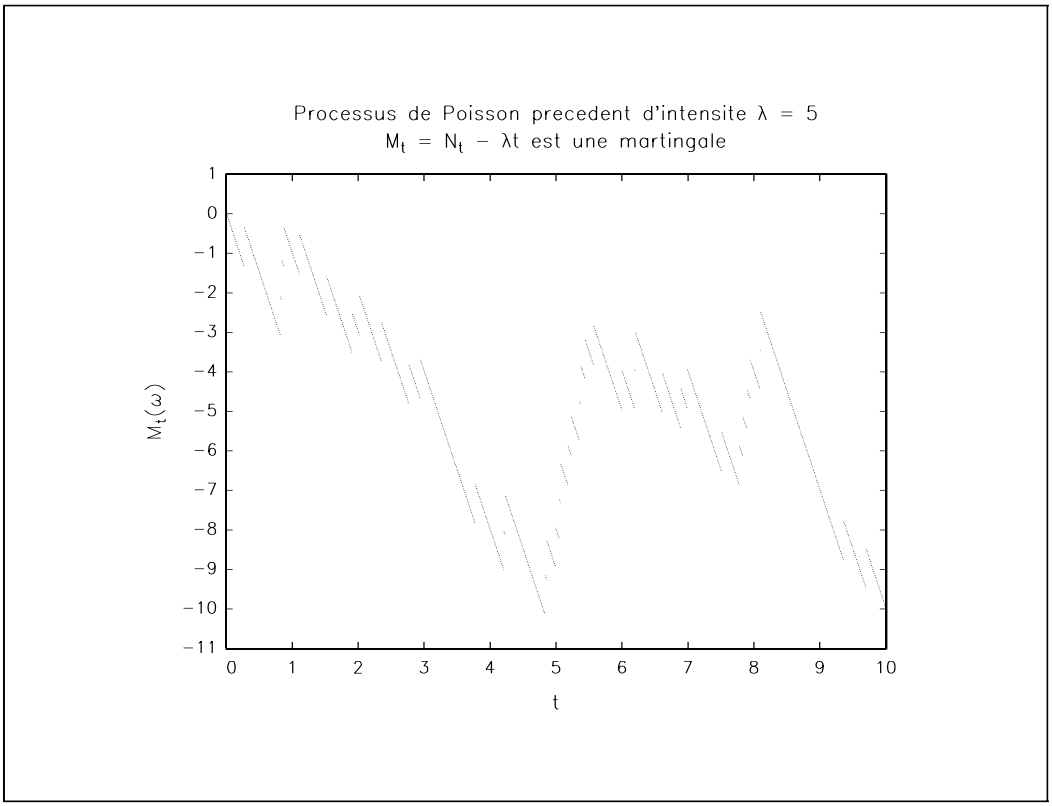
Graphique 1

```
proc (2) = Poisson(lambda,t0,T,L,K);
  local h,ti,u,N;

  h = (T-t0)/L;
  ti = seqa(t0,h,L+1);

  u = rndu(L,K);
  u = u .< (lambda*h);
  u = zeros(1,K)|u;

  N = cumsumc(u);
  retp(ti,N);
endp;
```

Dans l'exemple suivant, nous simulons deux processus de Poisson $N_t^{(1)}$ et $N_t^{(2)}$ d'intensité respective $\lambda_1 = 5$ et $\lambda_2 = 2$. Le graphique 1 représente les deux trajectoires. Nous illustrons ensuite les deux propriétés suivantes : $M_t = N_t - \lambda t$ et $V_t = M_t^2 - \lambda t$ sont des martingales (graphiques 2 et 3).

```
new;
library pgraph;
#include poisson.src;

rndseed 123;

L = 1000;
t0 = 0;
TT = 10;
lambda = 5;
{t,N} = Poisson(lambda,t0,TT,L,1);
{t,N2} = Poisson(2,t0,TT,L,1);
```

```
graphset;
  fonts("simplex simgrma");
  _pdate = "";
  _plwidth = 5; _pnum = 2;
  title("Simulation de processus de Poisson"\
        "\L__ \202l\201 = 5    _ _ \202l\201 = 2");
  xtics(t0,TT,1,0);
  ytics(0,50,10,0);
  xs = t[1:L];
  xe = t[2:L+1];
  ys = N[1:L];
  ye = N[1:L];
  e = ones(L,1);

  p1 = e~6*e~xs~ys~xe~ye~0*e~7*e~0*e;

  ys = N2[1:L];
  ye = N2[1:L];

  p2 = e~3*e~xs~ys~xe~ye~0*e~7*e~0*e;

  _pline = p1|p2;

  xlabel("t");
  ylabel("N]t[(\202w\201)");
  graphprt("-c=1 -cf=poiss1a.eps -w=5");
  draw;


M = N - lambda*t;

graphset;
  fonts("simplex simgrma");
  _pdate = "";
  _plwidth = 5; _pnum = 2;
  title("Processus de Poisson precedent d'intensite \202l \201= 5"\
        "\LM]t[ = N]t[ - \202l\201t est une martingale");
  xtics(t0,TT,1,0);
  ytics(-11,1,1,0);
  xs = t[1:L];
  xe = t[2:L+1];
  ys = M[1:L];
  ye = M[1:L];
  e = ones(L,1);

  _pline = e~6*e~xs~ys~xe~ye~0*e~7*e~0*e;

  xlabel("t");
  ylabel("M]t[(\202w\201)");
  graphprt("-c=1 -cf=poiss1b.eps -w=5");
  draw;


V = M^2 - lambda*t;

graphset;
  fonts("simplex simgrma");
  _pdate = "";
  _plwidth = 5; _pnum = 2;
  title("Processus de Poisson precedent d'intensite \202l \201= 5"\
```



Graphique 2

```
      "\LV]t[ = M]t[ - \2021\201t est une martingale");
xtics(t0,TT,1,0);
ytics(-40,80,10,0);
xs = t[1:L];
xe = t[2:L+1];
ys = V[1:L];
ye = V[1:L];
e = ones(L,1);

_pline = e~6*e~xs~ys~xe~ye~0*e~7*e~0*e;

xlabel("t");
ylabel("V]t[(\202w\201)");
graphprt("-c=1 -cf=poiss1c.eps -w=5");
draw;
```

1.1.2 Les processus de diffusion avec saut

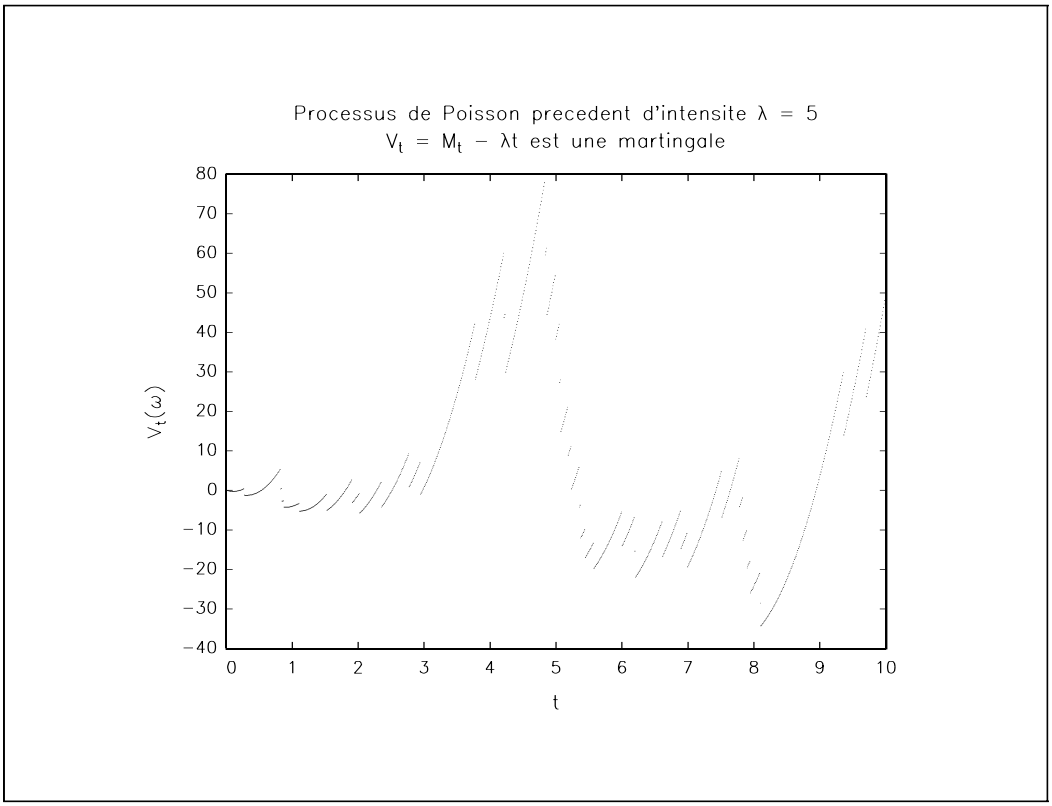
Considérons le processus $X(t)$ défini par

$$\begin{cases} dX_t &= \mu(t, X_t)\, dt + \sigma(t, X_t)\, dW_t + \kappa(t, X_t)\, dN_t \\ X(t_0) &= x_0 \end{cases} \tag{1}$$

avec W_t un processus de Wiener standard et N_t un processus de Poisson d'intensité $\lambda(t, X_t)$. Le processus (1) est appelé processus de diffusion avec saut. Il est actuellement de plus en plus utilisé en finance[3], car il permet de lever les hypothèses gaussienne et de continuité. Outre la valorisation des actifs contingents[4], la représentation (1)

[3] Voir les recherches menés par David S. Bates (*Journal of Finance* [1991], NBER n° 4596 et *Journal of International Money and Finance* [1996]).

[4] Lamberton, D. et B. Lapeyre [1991], Introduction au calcul stochastique appliqué à la finance, Collection Mathématiques et Applications, Ellipses



Graphique 3

est employée pour extraire l'information sur les marchés dérivés à partir des volatilités implicites[5]. Pour simuler le processus (1), nous devons combiner un algorithme de simulation d'une EDS[6] et un algorithme de simulation d'un processus de Poisson[7]. La procédure `Saut` implémente l'algorithme d'Euler-Maruyama et utilise le résultat (3.80) de Merton [1990] (Continuous-Time Finance, Basil Blackwell, Cambridge).

Saut

■ **Objet**

Simulation de processus de diffusion avec saut $\{X_t, t_0 \leq t \leq T\}$.

■ **Syntaxe**

{t,xt,dN} = Saut(x0,&mu,&sigma,&kappa,&lambda,t0,TT,N,Ns);

■ **Entrées**

x0	scalaire ou vecteur $Ns \times 1$, valeur initiale x_0 prise par le processus $X(t)$.
&mu	pointeur d'une procédure qui calcule la fonction $\mu(t, X_t)$.
&sigma	pointeur d'une procédure qui calcule la fonction $\sigma(t, X_t)$.
&kappa	pointeur d'une procédure qui calcule la fonction $\kappa(t, X_t)$.
&lambda	pointeur d'une procédure qui calcule la fonction $\lambda(t, X_t)$.
t0	scalaire, t_0.
TT	scalaire, T.
N	scalaire, nombre de points de discrétisation de l'intervalle $[t_0, T]$.
Ns	Nombre de simulations.

■ **Sorties**

t	vecteur $(N+1) \times 1$, temps t_i.
xt	matrice $(N+1) \times Ns$, processus de saut simulés.
dN	matrice $(N+1) \times Ns$, indique l'occurence d'un saut (valeur 1) ou son absence (0).

[5] Borensztein, E.R. et M.P. Dooley [1987], Options on Foreign Exchange and Exchange Rate Expectations, IMF Staff Papers
[6] Kloeden, P.E. et E. Platen [1992], Numerical Solution of Stochastic Differential Equations, Springer-Verlag, New York
[7] Devroye, L. [1986], Non-Uniform Random Variate Generation, Springer-Verlag, New York

■ **Remarque**

Le i-ième processus simulé correspond au vecteur `xt[.,i]`. La procédure `Saut` exploite la vectorisation de GAUSS. Les procédures qui définissent les fonctions doivent donc utiliser les opérateurs élément par élément.

Voici le code de la procédure `Saut` :

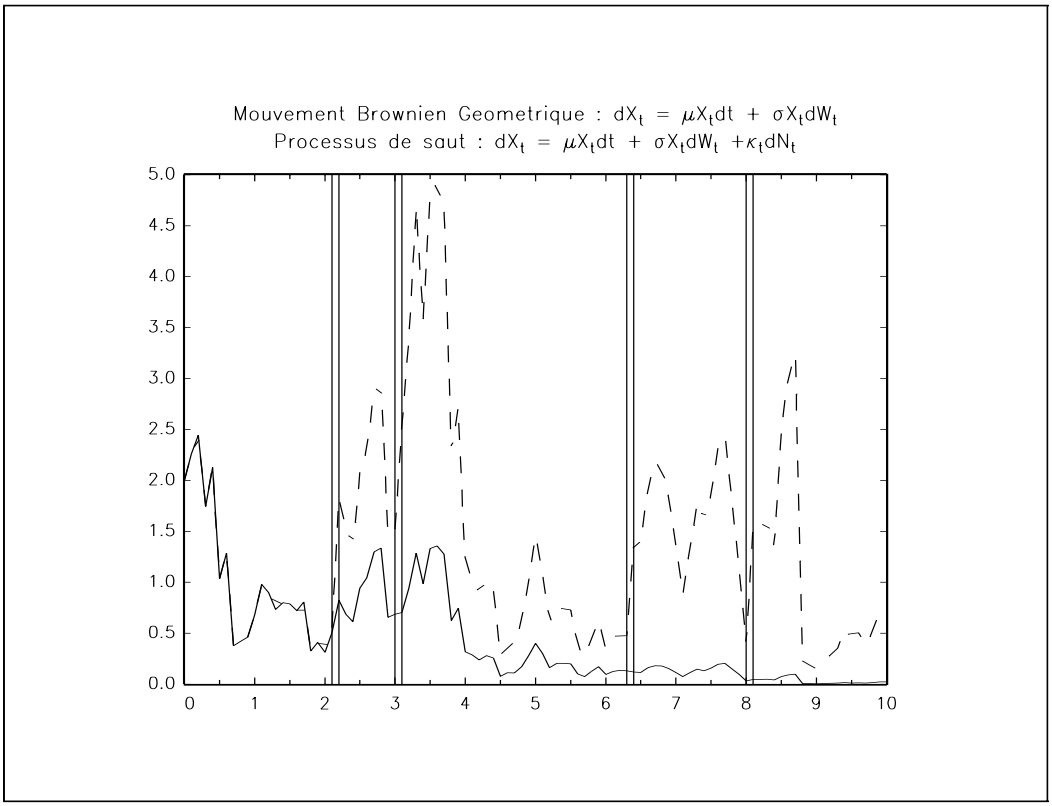
```
proc (3) = saut(x0,mu,sigma,kappa,lambda,t0,TT,N,Ns);
  local h,t,xt,u,dN,i,xi,ti,dN_,lambda_;
  local mu:proc,sigma:proc,kappa:proc,lambda:proc;

  h = (TT-t0)/N;
  t = seqa(t0,h,N+1);

  xt = zeros(N+1,Ns);
  xt[1,.] = x0.*ones(1,Ns);
  u = rndn(N,Ns)*sqrt(h);
  dN = zeros(N+1,Ns);

  i = 1;
  do until i > N;
    xi = xt[i,.]; ti = t[i];
    lambda_ = lambda(ti,xi);
    dN_ = rndu(1,Ns) .< (lambda_*h);
    xt[1+i,.] = xi + mu(ti,xi)*h
                   + sigma(ti,xi).*u[i,.] + kappa(ti,xi).*dN_;
    dN[1+i,.] = dN_;
    i = i + 1;
  endo;

  retp(t,xt,dN);
endp;
```

Considérons le processus suivant

$$\left\{ \begin{array}{rcl} dX_t & = & \mu X_t\, dt + \sigma X_t\, dW_t + \kappa_t\, dN_t \\ X(0) & = & 2 \end{array} \right.$$

avec κ_t égal à 1. $\lambda(t, X_t)$ prend les valeurs 1 si $t \leq 5$ et 0.5 si $t > 5$. Dans le programme suivant, nous simulons ce processus et nous le comparons à celui obtenu sans saut (dans ce cas, il suffit de poser $\lambda(t, X_t) = 0$ ou $\kappa(t, X_t) = 0$). Nous indiquons aussi les occurences de saut sur le graphique 4.

```
new;
library pgraph;
#include saut.src;

x0 = 2;

proc mu(t,x);
  retp(0.0002*x);
endp;

proc sigma(t,x);
  retp(1.09*x);
endp;

proc kappa(t,x);
  retp(1);
endp;

proc lambda1(t,x);
  retp(0);
```



Graphique 4

```
endp;

rndseed 123;
{t,x1,dN} = saut(x0,&mu,&sigma,&kappa,&lambda1,0,10,100,1);

proc lambda2(t,x);
  if t <= 5;
    retp(1);
  else;
    retp(0.5);
  endif;
endp;

rndseed 123;
{t,x2,dN} = saut(x0,&mu,&sigma,&kappa,&lambda2,0,10,100,1);
xs2 = indexcat(dN[.,1],1);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2;
  title("Mouvement Brownien Geometrique : "\
        "dX]t[ = \202m\201X]t[dt + \202s\201X]t[dW]t["\
        "\LProcessus de saut : "\
        "dX]t[ = \202m\201X]t[dt + \202s\201X]t[dW]t[ +\202k\201]t[dN]t[");
  xs = t[xs2];
  e = ones(rows(xs),1);
  xe = xs;
  ys = e*0;
  ye = e*100;
```

```
_pline = e~6*e~xs~ys~xe~ye~0*e~7*e~0*e;

xs = t[xs2-1];
e = ones(rows(xs),1);
xe = xs;
ys = e*0;
ye = e*100;

_pline = _pline|(e~6*e~xs~ys~xe~ye~0*e~7*e~0*e);
 graphprt("-c=1 -cf=saut.eps -w=5");
xy(t,x1~x2);
```

1.2 Valorisation des actifs contingents

Dans le cas d'un modèle d'arbitrage avec l'hypothèse d'un marché complet et viable, la valeur d'un actif contingent est solution d'une équation différentielle partielle parabolique d'ordre 2. La solution de cette EDP n'est pas toujours évidente. Le recours à des méthodes numériques est souvent nécessaire. Trois méthodes sont principalement utilisées : l'approximation par les différences finies et les θ-schémas, la simulation de la représentation de Feynman-Kac et l'approximation par une chaîne de Markov. La méthode des différences finies est présentée dans la section suivante.

1.2.1 Méthode de Monte Carlo pour la valorisation d'une option Look-Back

Supposons que les hypothèses du modèle de Black et Scholes soient vérifiées. En particulier, le prix du sous-jacent S_t suit un processus de diffusion donné par l'EDS suivante :

$$\left\{ \begin{array}{rcl} dS_t & = & \mu S_t\, dt + \sigma S_t\, dW_t \\ S_{t_0} & = & S_0 \end{array} \right.$$

Cet actif distribue un revenu continu égal à $\mathbf{d}S_t$. Soit une option sur cet actif. Nous notons $G(S_T)$ le Pay Off de l'option d'échéance T. Alors la valeur V de l'option est donnée par le théorème de représentation de Feynam-Kac

$$V = E'\left[\exp(-r\tau)\, G(S_T) \middle| \mathcal{F}_{t_0}\right] \tag{2}$$

avec r le taux d'intérêt et τ la maturité de l'option, c'est-à-dire $T - t_0$. E' représente l'espérance mathématique sous la probabilité neutre au risque. Le processus corrigé du risque est donné par le théorème de Girsanov. Nous obtenons

$$\left\{ \begin{array}{rcl} dS_t & = & (r - \mathbf{d})\, S_t\, dt + \sigma S_t\, dW'_t \\ S_{t_0} & = & S_0 \end{array} \right.$$

Nous pouvons utiliser la méthode de Monte Carlo pour obtenir la valeur de V. Soit g_i une valeur simulée de $G(S_T)$, alors nous avons d'après la loi des grands nombres

$$\lim_{n\to\infty} \exp(-r\tau) \left(\frac{1}{n} \sum_{i=1}^{n} g_i \right) = V$$

Pour une option look-back, la valeur du Pay-Off est

$$G(S_T) = S(T) - \min_{t_0 \le t \le T} S(t)$$

dans le cas d'une option d'achat, et

$$G(S_T) = \max_{t_0 \le t \le T} S(t) - S(T)$$

dans le cas d'une option de vente. Nous pouvons aussi utiliser la procédure `mc_lbo` pour les options sur Futures[8] et les options de changes[9].

[8] Soit F_t la valeur d'un future. La dynamique corrigée du risque est alors donnée par

$$\left\{ \begin{array}{rcl} dF_t & = & \sigma F_t\, dW'_t \\ F_{t_0} & = & F_0 \end{array} \right.$$

Il suffit donc de poser $S_0 = F_0$ et $\mathbf{d} = r$.

[9] La valorisation d'une option de change est basée qur dynamique corrigée du risque suivante

$$\left\{ \begin{array}{rcl} dS_t & = & (r - r^\star)\, S_t\, dt + \sigma S_t\, dW'_t \\ S_{t_0} & = & S_0 \end{array} \right.$$

avec $r^\star$ le taux d'intérêt sans risque étranger. Il suffit donc de poser $\mathbf{d} = r^\star$.

mc_lbo

■ **Objet**

Valorisation d'une option look-back par la méthode de Monte Carlo.

■ **Syntaxe**

prime = mc_lbo(S,r,d,sigma,tau);

■ **Entrées**

S	scalaire, prix du sous-jacent S_0.
r	scalaire, taux d'intérêt continu r.
d	scalaire, taux de rendement continu **d** du sous-jacent.
sigma	scalaire, volatilité historique σ.
tau	scalaire, maturité τ de l'option.

■ **Sorties**

prime	scalaire, prime de l'option.

■ **Variables globales**

_print	scalaire, 1 pour l'affichage écran.
_mc_simulations	vecteur 2×1. _mc_simulations[1] indique le nombre de simulations pour chaque réplication (1000 par défaut). _mc_simulations[2] indique le nombre de réplications (1 par défaut).
_mc_nbpoints	scalaire, nombre de points de discrétisation (par défaut, 5 par jour).
_mc_sauvegarde	chaîne de caractères, fichier de sauvergarde des primes simulées.
_type_option	chaîne de caractères, "call" pour une option d'achat et "put" pour une option de vente (défaut = "call").

Voici le code de la procédure `mc_lbo` :

```
declare matrix _print                       = 1;
declare matrix _mc_simulations              = {1000,1};
declare matrix _mc_nbpoints                 = 0;
declare matrix _mc_sauvegarde               = 0;
declare string _type_option                 ?= "call";

proc mc_lbo(S,r,d,sigma,tau);
  local Ns,Nr,n,h,hsqrt,mu,x,i,j,u;
  local somme,g,k,nb_simulations,prime;
  local fh,varnames,l,old_cursor;

  old_cursor = csrtype(0);
  Ns = _mc_simulations[1];  /* Nombre de simulations pour chaque replication */
  Nr = _mc_simulations[2];  /* Nombre de replications */
  somme = 0;
  nb_simulations = 0;       /* Nombre de simulations totales */

  if _mc_nbpoints == 0;
    N = trunc(tau*365*5);
  else;
    N = _mc_nbpoints;       /* Nombre de points de discretisation */
  endif;

  h = tau/n;  hsqrt = sqrt(h);  /* h : pas de discretisation */
  mu = r-d;

  if _print == 1;
    cls;
    locate 2,6;  print chrs(218~196*ones(1,72)~191);
    l = 1; do until l>3;
      locate 2+l,6; print chrs(179~zeros(1,72)~179);
```

```
    l = l+1;
  endo;
  locate 6,6;  print chrs(192~196*ones(1,72)~217);
  locate 4,8;
  if lower(_type_option) $== "put";
    print "Valorisation d'une option de vente LOOK-BACK -- PUT -- par"\
          " Monte Carlo";
  else;
    print "Valorisation d'une option d'achat LOOK-BACK -- CALL -- par"\
          " Monte Carlo";
  endif;
  locate 8,20;  print chrs(218~196*ones(1,40)~191);
  l = 1; do until l>6;
    locate 8+l,20; print chrs(179~zeros(1,40)~179);
    l = l+1;
  endo;
  locate 14,20;  print chrs(192~196*ones(1,40)~217);
  locate 9,22; print "Prix du sous-jacent :";
  locate 9,50; print ftos(s,"%lf",10,5);
  locate 10,22; print "Taux d'interet :     ";
  locate 10,50; print ftos(r,"%lf",10,5);
  locate 11,22; print "Taux de rendement :  ";
  locate 11,50; print ftos(d,"%lf",10,5);
  locate 12,22; print "Volatilite :         ";
  locate 12,50; print ftos(sigma,"%lf",10,5);
  locate 13,22; print "Maturite :           ";
  locate 13,50; print ftos(tau,"%lf",10,5);
endif;

if type(_mc_sauvegarde) == 13;
  varnames = { g } ;
  create fh = ^_mc_sauvegarde with varnames,0,8;
endif;

j = 1;
do until j > nr;
  x = s.*ones(1,ns);
  k = x;
  i = 1;
  do until i > N;
    u = rndn(1,Ns);
    u = hsqrt*u;
    x = x+ (mu*x)*h + (sigma*x).*u;
    if lower(_type_option) $== "put";
      K = maxc(K|x); K = K';
    else;
      K = minc(K|x); K = K';
    endif;
    i=i+1;
  endo;
  nb_simulations = nb_simulations + Ns;
  if lower(_type_option) $== "put";
    g = K-x;                              /* Pay Off d'un put */
  else;
    g = x-K;                              /* Pay Off d'un call */
  endif;

  g = g|zeros(1,ns);
  g = maxc(g);
  if type(_mc_sauvegarde) == 13;
```

```
      gosub sauvegarde;
    endif;
    somme = somme + sumc(g);
    prime = exp(-r*tau)*(somme/nb_simulations);  /* Formule de Feynman-Kac */
    if _print == 1;
      gosub affichage;
    endif;
    if key == 1079;
      retp(prime);
    endif;
    j = j+1;
  endo;
  locate 23,1;
  call csrtype(old_cursor);
  retp(prime);

  affichage:
    locate 16,20;  print chrs(218~196*ones(1,40)~191);
    l = 1; do until l>5;
      locate 16+l,20; print chrs(179~zeros(1,40)~179);
      l=l+1;
    endo;
    locate 21,20; print chrs(192~196*ones(1,40)~217);
    locate 17,22; print "Nombre de points :   ";
    locate 17,50; print ftos(n,"%lf",10,0);
    locate 18,22; print "Pas :                ";
    locate 18,50; print ftos(h,"%lf",10,8);
    locate 19,22; print "Nombre de simulations :";
    locate 19,50; print ftos(nb_simulations,"%lf",10,0);
    locate 20,22; print "Valeur de la prime : ";
    locate 20,50; print ftos(prime,"%lf",10,6);
  return;

  sauvegarde:
    call writer(fh,g);
  return;

endp;
```

Cherchons la valeur d'une option d'achat look-back de maturité 1 an sur un actif ne distribuant pas de dividendes. La valeur actuelle du sous-jacent est 100 Francs. La volatilité historique est estimée à 20%/an et le taux d'intérêt discret est 10%.

```
new;
#include lbo.src;

tt = hsec;

rndseed 123;
_mc_simulations = {100,10};  /* 1000 simulations */
_type_option = "call";
_print = 1;
prime1 = mc_lbo(100,ln(1+0.10),0,0.2,1);

rndseed 123;
_mc_nbpoints = 100;
prime2 = mc_lbo(100,ln(1+0.10),0,0.2,1);

output file = lbo.out reset;

print ftos(prime1,"prime1 : %lf",10,5);
```

```
print ftos(prime2,"prime2 : %lf",10,5);

print;
print ftos((hsec-tt)/100,"Temps de calcul : %lf secondes",5,3);

output off;


prime1 :   18.92782
prime2 :   18.58551

Temps de calcul : 11.860 secondes
```

1.2.2 Un exemple d'accélérateur de Monte Carlo

Nous pouvons améliorer la méthode de Monte Carlo en utilisant des accélérateurs[10]. Considérons par exemple le modèle binomial de Cox, Ross et Rubinstein. Nous notons u le coefficient de hausse, d le coefficient de baisse et π la probabilité neutre au risque. Soit N le nombre d'arbitrages. Une trajectoire simulée $\mathcal{S}$ du processus corrigé du risque est obtenue en considérant la formule récursive suivante

$$S_{i+1} = [ub_i + d(1 - b_i)] S_i$$

avec b_i un nombre aléatoire de loi de Bernoulli de paramètre π. Une valeur simulée g_i de $G(S_T)$ peut alors être obtenue à partir de la trajectoire $\mathcal{S}$. Parmi les méthodes d'accélération, la plus célèbre est la méthode de réduction de variance. Considérons par exemple l'utilisation des variables antithétiques[11]. L'idée est d'obtenir à partir des mêmes nombres aléatoires une autre trajectoire $\mathcal{S}^*$. Nous pouvons alors associer à g_i une valeur simulée *antithétique* $g_i^\star$. Nous définissons alors l'accélérateur de Monte Carlo par l'opération suivante

$$g_i \longleftarrow \frac{g_i + g_i^\star}{2}$$

(cela revient à considérer la moyenne de la valeur simulée et de celle *antithétique*). La génération des variables b_i peut être obtenue à partir d'un nombre aléatoire uniforme w_i. Comme $v_i = 1 - w_i$ est aussi un nombre aléatoire uniforme, nous pouvons considérer pour chaque variable b_i une variable antithétique $b_i^\star$ basée sur v_i. Nous avons implémenté cette méthode pour la procédure `CRR_asian`. Nous avons considéré des actifs ne distribuant pas de dividendes, mais nous pouvons facilement adapter cette procédure à d'autres types d'options (il suffit de modifier la procédure `parametre_option` qui définit u, d et π).

CRR_asian

■ **Objet**
Valorisation des options asiatiques et européennes par le modèle binomial de Cox, Ross et Rubinstein.

■ **Syntaxe**
prime = CRR_asian(S,K,tau,sigma,r,N,Ns);

■ **Entrées**

S	scalaire, prix du sous-jacent S_0.
K	scalaire, prix d'exercice K de l'option européenne.
tau	scalaire, maturité τ de l'option.
sigma	scalaire, volatilité historique σ.
r	scalaire, taux d'intérêt continu r.
N	scalaire, nombre d'arbitrage du modèle binomial.
Ns	scalaire, nombre de simulations.

■ **Sorties**

prime	vecteur 6×1, valeur des primes.
	prime[1] : option asiatique d'achat (Floating Strike).
	prime[2] : option asiatique de vente (Floating Strike).
	prime[3] : option asiatique d'achat (Fixed Strike).
	prime[4] : option asiatique de vente (Fixed Strike).
	prime[5] : option européenne d'achat.
	prime[6] : option européenne de vente.

[10] Voir la section 4.3 de RUBINSTEIN, R.Y. [1981], Simulation and the Monte Carlo Method, John Wiley & Sons, New York
[11] Voir la section 5.3 de RIPLEY, B.D. [1987], Stochastic Simulation, John Wiley & Sons, New York

■ **Variables globales**

_print	scalaire, 1 pour l'affichage des résultats.
_accelerateur	scalaire, 1 (par défaut) pour la méthode de Monte Carlo avec l'accélérateur et 0 pour la méthode MC sans accélérateur.

Voici le code de la procédure `CRR_asian` :

```
declare matrix _print = 1;
declare matrix _accelerateur = 1;

proc (2) = simulation(_pi,u,d,n,m);
  local w,b1,b2,p1,p2;
  w = rndu(n,m);     /* variable aleatoire uniforme      */
  b1 = w.<=_pi;      /* variable aleatoire de Bernouilli */
  w = 1 - w;
  b2 = w.<=_pi;      /* variable aleatoire antithetique  */

  p1 = b1*u + (1-b1)*d;
  p1 = cumprodc(p1);

  p2 = b2*u + (1-b2)*d;
  p2 = cumprodc(p2);

  retp(p1,p2);
endp;


proc (3) = parametre_option(S,tau,sigma,r,n);
  local u,d,_pi;
  u = exp(sigma*sqrt(tau/n));
  d = 1/u;
  _pi = (exp(r*tau/n)-d)/(u-d);
  retp(_pi,u,d);
endp;


proc PayOFF(Stilde,K,r,tau);
  local N,ST,Sm,G,Prime;

  Prime = zeros(6,1);
  N = rows(Stilde);

  ST = Stilde[N,.]';          /* Valeurs finales du sous-jacent */
  Sm = meanc(Stilde);         /* valeurs moyennes prises par le sous-jacent */

  /* Floating Strike Options --- CALL */
  G = (ST-Sm);
  G = G.*(G.>0);
  Prime[1] = meanc(G);

  /* Floating Strike Options --- PUT  */
  G = (Sm-ST);
  G = G.*(G.>0);
  Prime[2] = meanc(G);

  /* Fixed Strike Options --- CALL    */
  G = (Sm-K);
  G = G.*(G.>0);
  Prime[3] = meanc(G);
```

```
  /* Fixed Strike Options --- PUT     */
  G = (K-Sm);
  G = G.*(G.>0);
  Prime[4] = meanc(G);

  /* European CALL                    */
  G = (ST-K);
  G = G.*(G.>0);
  Prime[5] = meanc(G);

  /* European PUT                     */
  G = (K-ST);
  G = G.*(G.>0);
  Prime[6] = meanc(G);

  Prime = exp(-r*tau)*Prime;

  retp(prime);
endp;


proc CRR_asian(S,K,Tau,Sigma,r,N,M);
  local _pi,u,d,p1,p2,Stilde,prime1,prime2,prime,str;

  {_pi,u,d} = parametre_option(S,Tau,Sigma,r,N);

  {p1,p2} = simulation(_pi,u,d,n,m);

  Stilde = S*p1;            /* Premiere trajectoire simulee du sous-jacent */
  Prime1 = PayOff(Stilde,K,r,tau);

  if _accelerateur /= 1;
    retp(Prime1);
  endif;

  Stilde = S*p2;            /* Deuxieme trajectoire simulee du sous-jacent */
  Prime2 = PayOff(Stilde,K,r,tau);

  Prime = 0.5*(Prime1+Prime2);  /* Accelerateur de Monte carlo */

  if _print == 1;

    print "Floating Strike Option:";
    str = ftos(prime[1],"call = %lf",10,5);
    str = str$+ftos(prime[2],"     put = %lf",10,5);
    print str; print;

    print "Fixed Strike Option:";
    str = ftos(prime[3],"call = %lf",10,5);
    str = str$+ftos(prime[4],"     put = %lf",10,5);
    print str; print;

    print "European Option:";
    str = ftos(prime[5],"call = %lf",10,5);
    str = str$+ftos(prime[6],"     put = %lf",10,5);
    print str; print;

  endif;

  retp(Prime);
```

```
endp;
```

Considérons l'option suivante :

Prix du sous-jacent	100 Francs
Maturité de l'option	90 jours
volatilité σ	20%
taux d'intérêt r	8%
Nombre d'arbitrages	3
Prix d'exercice de l'option européenne	98 Francs

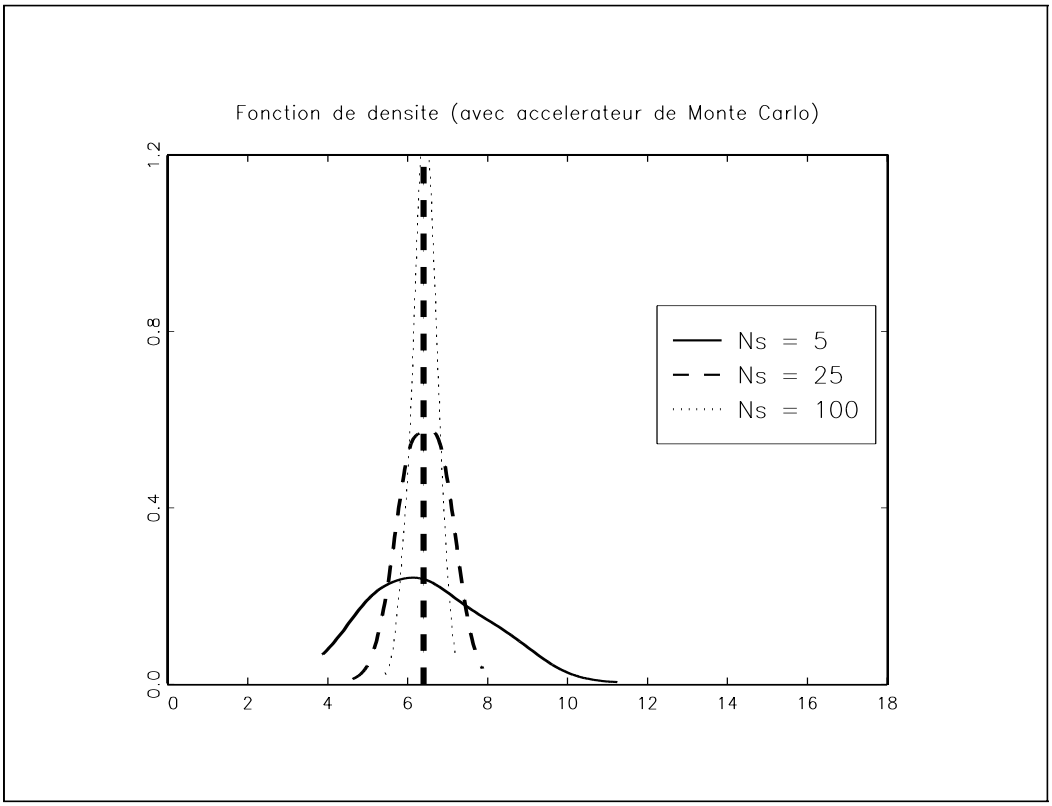
Nous utilisons 10 000 simulations pour calculer les différents prix d'option.

```
new;
#include asian.src;

output file = asian.out reset;

tt = hsec;

Prime = CRR_asian(100,98,90/365,0.2,0.08,3,10000);

print;
print ftos((hsec-tt)/100,"Temps de calcul : %lf secondes",5,3);

output off;


Floating Strike Option:
call =    2.24892     put =    1.57524

Fixed Strike Option:
call =    4.84020     put =    1.59075

European Option:
call =    6.38268     put =    2.45955


Temps de calcul : 0.940 secondes
```

Dans l'exemple suivant, nous montrons l'influence de l'accélérateur sur la convergence. Pour cela, nous considérons la fonction de densité empirique de l'estimateur de Monte carlo avec et sans accélérateur. Nous utilisons 250 réplications. Les graphiques (5) et (6) montrent clairement l'**accélération** de Monte Carlo. Ainsi, l'estimateur accéléré basé sur 25 simulations a la même puissance que celui non accéléré basé sur 100 simulations !

```
new;
library tsm,optmum,pgraph;
TSMset;

output file = mcarlo.out reset;
tt = hsec;

#include asian.src;

MC1 = zeros(250,3);    /* avec accelerateur de Monte Carlo */
MC2 = zeros(250,3);    /* sans accelerateur de Monte Carlo */
_print = 0;

rndseed 123; _accelerateur = 1;

i = 1;
do until i > 250;
```

```
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,5);
  MC1[i,1] = Prime[5];
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,25);
  MC1[i,2] = Prime[5];
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,100);
  MC1[i,3] = Prime[5];
  i = i+1;
endo;

rndseed 123; _accelerateur = 0;

i = 1;
do until i > 250;
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,5);
  MC2[i,1] = Prime[5];
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,25);
  MC2[i,2] = Prime[5];
  Prime = CRR_asian(100,98,90/365,0.2,0.08,3,100);
  MC2[i,3] = Prime[5];
  i = i+1;
endo;


x1 = zeros(128,3); x2 = x1;
d1 = zeros(128,3); d2 = d1;

i = 1;
do until i > 3;
  {x1[.,i],d1[.,i],F,retcode} = Kernel(MC1[.,i]);
  {x2[.,i],d2[.,i],F,retcode} = Kernel(MC2[.,i]);
  i = i + 1;
endo;

print ftos((hsec-tt)/100,"Temps de calcul : %lf secondes",10,5);

output off;

graphset;
  TITLE("Fonction de densite (avec accelerateur de Monte Carlo)");
  _plwidth = 6;
  _pdate = "";
  _pline = 1~1~6.3957115~0~6.3957115~10~1~7~10;
  _plegstr = "Ns = 5\000Ns = 25\000Ns = 100";
  _plegctl = {2 7 6 3};
  xtics(0,18,2,0); ytics(0,1.2,0.4,0);
  graphprt("-c=1 -cf=mcarlo1.eps -w=10");
  xy(x1,d1);

graphset;
  TITLE("Fonction de densite (sans accelerateur de Monte Carlo)");
  _plwidth = 6;
  _pdate = "";
  _pline = 1~1~6.3957115~0~6.3957115~10~1~7~10;
  _plegstr = "Ns = 5\000Ns = 25\000Ns = 100";
  _plegctl = {2 7 6 3};
  xtics(0,18,2,0); ytics(0,1.2,0.4,0);
  graphprt("-c=1 -cf=mcarlo2.eps -w=10");
  xy(x2,d2);
```



Graphique 5

```
Temps de calcul :    5.16000 secondes
```

1.2.3 Approximation par une chaîne de Markov discrète

Considérons l'EDS suivante

$$dX_t = \mu\left(t, X_t\right)\, dt + \sigma\left(t, X_t\right)\, dW_t \tag{3}$$

Nous cherchons à caractériser la fonction de distribution des probabilités de transition. La définition de celles-ci est

$$P\left(x,t;y,k\right) = \Pr\left(\, X\left(t+k\right) \leq y \middle|\, X\left(t\right) = x\right)$$

Nous cherchons à approximer le processus en temps continu (3) par un processus en temps discret et plus particulièrement par une chaîne de Markov. Soit $P^{\star}\left(x,t;y,k\right)$ une approximation de $P\left(x,t;y,k\right)$ Il est facile de montrer que nous avons

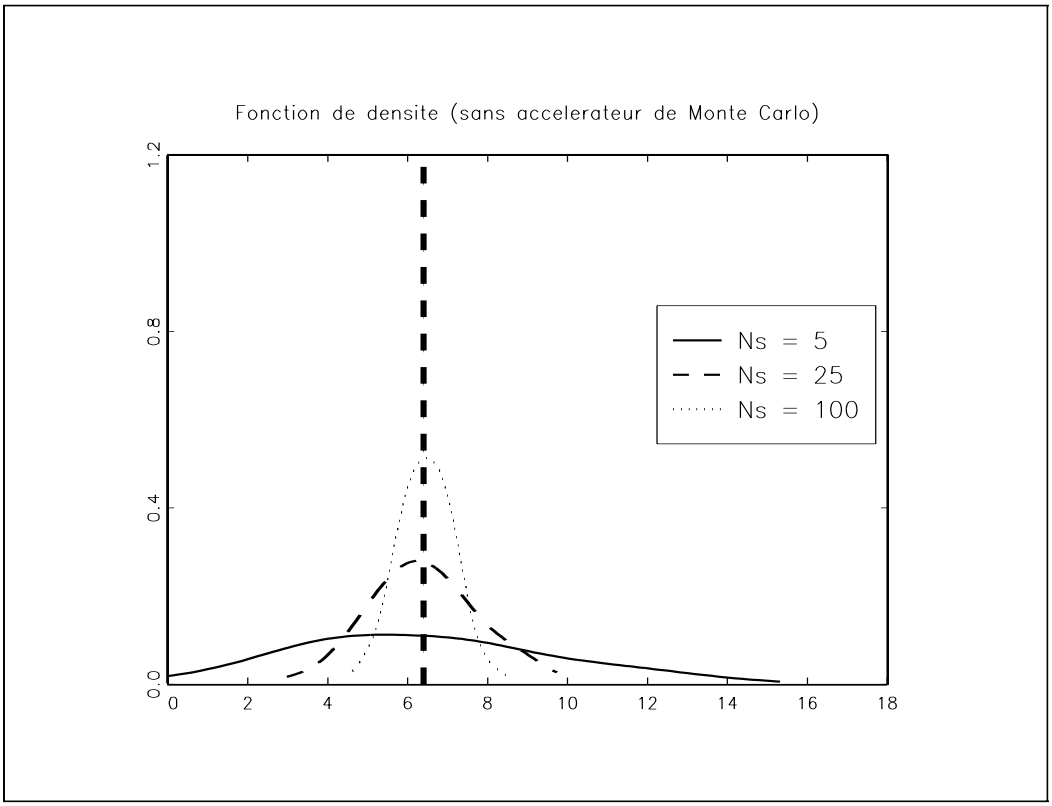
$$P^{\star}\left(x,t;y,k\right) = \Phi\left[\frac{y - x - \mu\left(t,x\right)k}{\sigma\left(t,x\right)\sqrt{k}}\right]$$

Considérons un nombre fini d'états de la nature ($X\left(s\right) \in E$ avec $\operatorname{card} E < \infty$). Par souci de simplification, nous posons $E = \left\{x_s = x_0 + sh, s = 0, \ldots, S\right\}$. Nous avons donc $x \in \left[x_0, x_S\right]$. Soit $p_{i,j}\left(t,t+k\right)$ la probabilité de transition de $x\left(t\right) = x_i$ vers $x\left(t+k\right) = x_j$. Plusieurs approximation sont possibles

$$\left\{\begin{array}{rcl} p_{i,j}\left(t,t+k\right) & = & P^{\star}\left(x_i,t;x_j,k\right) - P^{\star}\left(x_i,t;x_{j-1},k\right) \\ & = & P^{\star}\left(x_i,t;x_{j+1},k\right) - P^{\star}\left(x_i,t;x_j,k\right) \\ & = & P^{\star}\left(x_i,t;\frac{x_{j+1}+x_j}{2},k\right) - P^{\star}\left(x_i,t;\frac{x_j+x_{j-1}}{2},k\right) \end{array}\right.$$

Si nous prenons la dernière forme d'approximation (qui est une forme centrée), nous obtenons

$$p_{i,j}\left(t,t+k\right) = \Phi\left[\frac{\left(j+\frac{1}{2}-i\right)h - \mu\left(t,x_i\right)k}{\sigma\left(t,x_i\right)\sqrt{k}}\right] - \Phi\left[\frac{\left(j-\frac{1}{2}-i\right)h - \mu\left(t,x_i\right)k}{\sigma\left(t,x_i\right)\sqrt{k}}\right]$$



Graphique 6

Pour les bornes de l'intervalle, nous avons

$$p_{i,0}\left(t,t+k\right)=\Phi\left[\frac{\left(\frac{1}{2}-i\right)h-\mu\left(t,x_i\right)k}{\sigma\left(t,x_i\right)\sqrt{k}}\right]$$

et

$$p_{i,S}\left(t,t+k\right)=1-\Phi\left[\frac{\left(S-\frac{1}{2}-i\right)h-\mu\left(t,x_i\right)k}{\sigma\left(t,x_i\right)\sqrt{k}}\right]$$

Nous vérifions bien que $\sum_{j=0}^{S} p_{i,j}\left(t,t+k\right)=1$. Nous notons $\mathbf{P}\left(t,t+k\right)$ la matrice de Markov des probabilités de transition de l'instant t à l'instant $t+k$. La procédure `EDS_to_MarkovChain` permet d'obtenir $\mathbf{P}\left(t,t+k\right)$ dans le cas où le processus est de la forme

$$dX_t=\mu\left(X_t\right)\,dt+\sigma\left(X_t\right)\,dW_t$$

c'est-à-dire que les fonctions μ et σ ne dépendent pas de la variable t.

EDS_to_MarkovChain

■ **Objet**
Approximation d'une EDS par une chaîne de Markov discrète.

■ **Syntaxe**
{X,Pij} = EDS_toMarkovChain(&mu,&sigma,cn);

■ **Entrées**

&mu	pointeur d'une procédure qui calcule la fonction $\mu\left(X_t\right)$.
&sigma	pointeur d'une procédure qui calcule la fonction $\sigma\left(X_t\right)$.
cn	vecteur 4×1, paramètres de contrôle de l'approximation.
	cn[1] : valeur inférieure x_0 de l'intervalle d'approximation de X.
	cn[2] : valeur supérieure x_S de l'intervalle d'approximation de X.
	cn[3] : Nombre de points S de discrétisation de l'intervalle d'approximation de X.
	cn[4] : valeur de k.

■ **Sorties**

x	vecteur $(S+1) \times 1$, valeurs prises par $X(t)$.
Pij	matrice $(S+1) \times (S+1)$, matrice de Markov $\mathbf{P}(t, t+k)$.

■ **Remarque**

La procédure `EDS_to_MarkovChain` exploite la vectorisation de GAUSS. Les procédures qui définissent les fonctions doivent donc utiliser les opérateurs élément par élément.

Voici le code de la procédure `EDS_to_MarkovChain` :

```
proc (2) = EDS_to_MarkovChain(mu,sigma,cn);
  local mu:proc,sigma:proc;
  local X_min,X_max,Nx,k,h,X,i,j,Pij,mu_,sigma_;
  local div,num,num1,num2;
  X_min = cn[1];
  X_max = cn[2];
  Nx = cn[3];
  k = cn[4];
  h = (X_max-X_min)/Nx;
  X = seqa(X_min,h,Nx+1);
  i = seqa(0,1,Nx+1); j = i';
  Pij = zeros(Nx+1,Nx+1);
  mu_ = mu(X); sigma_ = sigma(X);
  div = sigma_*sqrt(k);
  num = (j[1,2:Nx]-i)*h-mu_*k;
  num1 = (num+0.5*h)./div;
  num2 = (num-0.5*h)./div;
  Pij[.,2:Nx] = cdfn(num1)-cdfn(num2);
  num = ((-i+0.5)*h-mu_*k)./div;
  Pij[.,1] = cdfn(num);
  num = ((Nx-i-0.5)*h-mu_*k)./div;
  Pij[.,Nx+1] = cdfnc(num);
  retp(X,Pij);
endp;
```

La valeur d'une prime d'option d'achat européenne de Black et Scholes est donnée par

$$C = \exp(-r\tau) E'\left[(S(T) - K)_+ \middle| \mathcal{F}_{t_0} \right]$$

avec

$$\left\{ \begin{array}{rcl} dS_t & = & rS_t\, dt + \sigma S_t\, dW_t' \\ S_{t_0} & = & S_0 \end{array} \right.$$

Pour calculer l'expression précédente, nous avons besoin de la fonction de densité de la variable $(S(T) - K)_+$. Nous pouvons utiliser la procédure `EDS_to_MarkovChain` pour obtenir $\mathbf{P}(t_0, t_0 + \tau)$. Nous avons

$$\mathbf{P}(t, t+nk) = \mathbf{P}(t, t+nk-k)\, \mathbf{P}(t+nk-k, t+nk)$$

En posant $\mathbf{P}(t,t) = \mathbf{I}$ et $k = \frac{\tau}{N}$, nous avons

$$\mathbf{P}(t, t+\tau) = \prod_{n=0}^{N} \mathbf{P}(t+nk, t+nk+k) \tag{4}$$

Pour calculer la fonction de densité conditionnelle de $(S(T) - K)_+$, nous utilisons la relation (4).

```
new;
#include markov.src;

hsec_ = hsec;

K = 98;
S0 = 100;
```

```
sig = 0.15;
t0 = 0;
TT = 95/365;
tau = TT - t0;
r = ln(1+0.08);

Prime = 5.3677;   /* Valeur de la prime de l'option d'achat dans
                     le modele de Black et Scholes */

proc mu(x);
  retp(r*x);
endp;

proc sigma(x);
  retp(sig*x);
endp;

S_min = 80;
S_max = 120;
Ns = 100;     /* Nombre de points pour la partition de S */
Nt = 50;      /* Nombre de points pour la partition du temps */

cn = S_min|S_max|Ns|(tau/Nt);

{S,P} = EDS_to_MarkovChain(&mu,&sigma,cn);

Prob = P;

i = 1;
do until i > Nt;
  Prob = Prob*P;
  i = i + 1;
endo;

indx = indexcat(S,S0);
Prob = Prob[indx,.]';

PayOff = S - K;
PayOff = PayOff.*(PayOff.>0);
C = exp(-r*tau)*(Prob'PayOff);

output file = markov.out reset;

print ftos(Prime,"Valeur de la prime de l'option d'achat : %lf",5,4);
print ftos(C,"Valeur approchee par la chaine de Markov : %lf",5,4);

print;
print ftos((hsec-hsec_)/100,"Temps de calcul : %lf secondes",5,3);

output off;


Valeur de la prime de l'option d'achat : 5.3677
Valeur approchee par la chaine de Markov : 5.3532

Temps de calcul : 5.660 secondes
```

1.3 Modèle de structure par terme à un seul facteur

Pour obtenir la structure par terme dans les modèles financiers d'arbitrage à une variable d'état, nous devons résoudre :

$$\begin{cases} \frac{1}{2}P_{xx}\sigma^2(t,x) + [\mu(t,x) - \lambda(t,x)\,\sigma(t,x)]\,P_x + P_t - rP = 0 \\ P(T,r) = 1 \\ R(t,T) = -\frac{1}{T-t}\ln P(T,r) \end{cases} \tag{5}$$

En général, le taux d'intérêt instantané r dépend de la variable d'état x

$$r = \alpha(t,x)$$

Dans les modèles financiers, les fonctions $\mu(x,t)$ et $\sigma(x,t)$ de l'équation différentielle stochastique qui modèlise la dynamique de la variable d'état ne dépendent pas en général du temps. Nous posons alors

$$\mu(t,x) = \mu(x)$$

et

$$\sigma(t,x) = \sigma(x)$$

Posons $\tau = T - t$. Le système (5) devient

$$\begin{cases} \frac{1}{2}P^\star_{xx}\sigma^2(x) + [\mu(x) - \lambda(x)\,\sigma(x)]\,P^\star_x - P^\star_\tau - rP^\star = 0 \\ P^\star(0,x) = 1 \\ R(\tau) = -\frac{1}{\tau}\ln P^\star(\tau,x) \end{cases} \tag{6}$$

Nous avons alors

$$P^\star(\tau,x) = P(T,x)$$

Avec ce changement de variable, nous transformons un problème de conditions aux bornes finales en un problème de conditions aux bornes intiales. Nous avons alors un problème de Cauchy, qui peut être facilement résolu par une méthode numérique. Pour cela, nous utilisons la méthode des différences finies particulièrement adaptée aux équations paraboliques d'ordre 2 en x.

1.3.1 La méthode des différences finies

Pour résoudre le système (6), nous discrétisons le processus $P^\star(\tau,x)$ dans l'espace et dans le temps. Soit $u(\tau,x)$ la solution de (6). Nous avons $\tau \in [0,\infty)$ et $x \in (-\infty,\infty)$. Considérons $\tau_{\max}$, x^- et x^+ trois scalaires. Nous nous proposons de résoudre le système pour $\tau \in [0,\tau_{\max}]$ et $x \in [x^-,x^+]$. Soient k et h les pas de discrétisations de τ et x, alors nous avons

$$\begin{aligned} x_j &= x^- + jh \\ \tau_n &= nk \end{aligned}$$

Soient $J = \left\lceil 1 + \frac{x^+ - x^-}{h} \right\rceil$ et $N = \left\lceil 1 + \frac{\tau_{\max}}{k} \right\rceil$ le nombre de points de discrétisation dans l'espace et dans le temps. Nous notons u_j^n la solution numérique de $P(\tau_n, x_j)$.

Discrétisation dans l'espace. Si nous employons la méthode des différences centrales d'approximation d'une dérivée, nous avons

$$\begin{aligned} \frac{\partial P^\star}{\partial x} &\simeq \left(\frac{u^n_{j+1} - u^n_{j-1}}{2h}\right) \\ \frac{\partial^2 P^\star}{\partial x^2} &\simeq \left(\frac{u^n_{j-1} - 2u^n_j + u^n_{j+1}}{h^2}\right) \end{aligned}$$

Nous remarquons que l'équation parabolique d'ordre 2 s'écrit aussi

$$\frac{\partial u}{\partial \tau} - \mathcal{A}_\tau u + ru = 0 \tag{7}$$

avec $\mathcal{A}$ l'opérateur différentiel du second ordre. En posant $\mathcal{A}^\bullet_\tau u = \mathcal{A}_\tau u - ru$, l'équation (7) devient

$$\frac{\partial u}{\partial \tau} = \mathcal{A}^\bullet_\tau u \tag{8}$$

Soit $\mathcal{A}^n_j$ la discrétisation de $\mathcal{A}^\bullet_\tau u$, nous avons

$$\mathcal{A}^n_j = \frac{1}{2}\sigma_j^2\left(\frac{u^n_{j-1} - 2u^n_j + u^n_{j+1}}{h^2}\right) + [\mu_j - \lambda_j\sigma_j]\left(\frac{u^n_{j+1} - u^n_{j-1}}{2h}\right) - r_j u^n_j$$

avec

$$\begin{aligned}\sigma_j &= \sigma(x_j)\\ \mu_j &= \mu(x_j)\\ \lambda_j &= \lambda(x_j)\\ r_j &= \alpha(x_j)\end{aligned}$$

Discrétisation dans le temps. Pour résoudre l'équation (8), nous utilisons le schéma d'Euler. Nous avons alors

$$\frac{u_j^{n+1}-u_j^n}{k} = \mathcal{A}_\tau^\bullet u$$

Comme la fonction $\mathcal{A}_\tau^\bullet u$ dépend du temps et de l'espace, nous ne pouvons pas utiliser l'algorithme classique d'Euler

$$u_j^{n+1} = u_j^n + k\mathcal{A}_\tau^\bullet u$$

Nous remplaçons alors la fonction $\mathcal{A}_\tau^\bullet u$ par la solution numérique $\mathcal{A}_j^n$. Nous devons alors résoudre

$$\frac{u_j^{n+1}-u_j^n}{k} = \mathcal{A}_j^n$$

La méthode des θ-schémas. Dans la sous-section précédente, nous utilisons la méthode des différences à droite d'approximation d'une dérivée. D'autres méthodes existent (différence à gauche, différence centrale, extrapolation de Richardson, etc). La méthode des θ-schémas s'intéresse à l'approximation de la fonction $\mathcal{A}_\tau^\bullet u$. Soit $\theta \in [0,1]$. Nous avons

$$\frac{u_j^{n+1}-u_j^n}{k} = \theta\mathcal{A}_j^{n+1} + (1-\theta)\mathcal{A}_j^n \tag{9}$$

Cette méthode revient à utiliser une différence à droite pour la discrétisation dans le temps pour $\theta = 1$ et une différence à gauche pour $\theta = 0$.

En remplaçant $\mathcal{A}_j^n$ par son expression, nous obtenons

$$\begin{aligned}&u_{j+1}^{n+1}\left[\tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k\theta + \tfrac{1}{2h}(\mu_j-\lambda_j\sigma_j)k\theta\right] + u_j^{n+1}\left[-1-\tfrac{\sigma_j^2}{h^2}k\theta - r_jk\theta\right]\\ &\qquad + u_{j-1}^{n+1}\left[\tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k\theta - \tfrac{1}{2h}(\mu_j-\lambda_j\sigma j)k\theta\right]\\ &+u_{j+1}^{n}\left[\tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k(1-\theta) + \tfrac{1}{2h}(\mu_j-\lambda_j\sigma_j)k(1-\theta)\right] + u_j^n\left[1-\tfrac{\sigma_j^2}{h^2}k(1-\theta) - r_jk(1-\theta)\right]\\ &\qquad + u_{j-1}^{n}\left[\tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k(1-\theta) - \tfrac{1}{2h}(\mu_j-\lambda_j\sigma j)k(1-\theta)\right] \quad = \quad 0\end{aligned} \tag{10}$$

1.3.2 Les différents algorithmes de résolution

Le schéma explicite. Ce schéma[12] correspond à $\theta = 0$. L'équation (10) devient

$$-u_j^{n+1} + a_j u_{j+1}^n + b_j u_j^n + c_j u_{j-1}^n = 0 \tag{11}$$

avec

$$\begin{aligned}a_j &= \tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k + \tfrac{1}{2h}(\mu_j-\lambda_j\sigma_j)k\\ b_j &= 1-\tfrac{\sigma_j^2}{h^2}k - r_jk\\ c_j &= \tfrac{1}{2}\tfrac{\sigma_j^2}{h^2}k - \tfrac{1}{2h}(\mu_j-\lambda_j\sigma j)k\end{aligned}$$

La solution numérique u_j^n s'obtient de manière itérative en imposant les conditions de Dirichlet aux bornes et en prenant en compte la condition initiale. Nous avons

$$u_j^{n+1} = a_j u_{j+1}^n + b_j u_j^n + c_j u_{j-1}^n$$

avec

$$\left\{\begin{array}{ll} u_j^0 = 1 & \forall j = 0,\ldots,J\\ u_{-1}^n = 0 & \forall n = 0,\ldots,N\\ u_{J+1}^n = 0 & \forall n = 0,\ldots,N\end{array}\right.$$

[12] On parle aussi de méthode d'Euler progressive (voir RAVIART et THOMAS [1983] page 173).

Le schéma totalement implicite. Ce schéma[13] correspond à $\theta = 1$. L'équation (10) devient

$$d_j u_{j+1}^{n+1} + e_j u_j^{n+1} + f_j u_{j-1}^{n+1} + u_j^n = 0 \tag{12}$$

avec

$$\begin{array}{rcl} d_j & = & a_j \\ e_j & = & b_j - 2 \\ f_j & = & c_j \end{array}$$

En imposant les conditions de Neumann aux bornes, c'est-à-dire $u_0^n = u_{-1}^n$ et $u_J^n = u_{J+1}^n$, la solution numérique u_j^n s'obtient en résolvant le système tridiagonal linéaire suivant :

$$\left[\begin{array}{cccccccc} f_0+e_0 & d_0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ f_1 & e_1 & d_1 & 0 & & & & \vdots \\ 0 & f_2 & e_2 & d_2 & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & f_j & e_j & d_j & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & 0 & f_{J-1} & e_{J-1} & d_{J-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & f_J & e_J+d_J \end{array}\right] \left[\begin{array}{c} u_0^{n+1} \\ u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_j^{n+1} \\ \vdots \\ u_{J-1}^{n+1} \\ u_J^{n+1} \end{array}\right] = \left[\begin{array}{c} -u_0^n \\ -u_1^n \\ -u_2^n \\ \vdots \\ -u_j^n \\ \vdots \\ -u_{J-1}^n \\ -u_J^n \end{array}\right] \tag{13}$$

Les schémas mixtes. Nous avons alors $\theta \in]0,1[$. Si $\theta = \frac{1}{2}$, nous obtenons le schéma de Crank-Nicholson. L'équation (10) devient

$$A_j u_{j+1}^n + B_j u_j^n + C_j u_{j-1}^n + D_j u_{j+1}^{n+1} + E_j u_j^{n+1} + F_j u_{j-1}^{n+1} = 0 \tag{14}$$

avec

$$\begin{array}{rcl} A_j & = & (1-\theta)\, a_j \\ B_j & = & (1-\theta)\,(b_j - 1) + 1 \\ C_j & = & (1-\theta)\, c_j \\ D_j & = & \theta d_j \\ E_j & = & \theta\,(e_j + 1) - 1 \\ F_j & = & \theta f_j \end{array}$$

En imposant les conditions de Dirichlet et de Neumann, la solution numérique u_j^n s'obtient en résolvant le système tridiagonal linéaire suivant :

$$\Upsilon \mathbf{u}_{n+1} = -\Lambda \mathbf{u}_n$$

avec

$$\Upsilon = \left[\begin{array}{cccccccc} F_0+E_0 & D_0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ F_1 & E_1 & D_1 & 0 & & & & \vdots \\ 0 & F_2 & E_2 & D_2 & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & F_j & E_j & D_j & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & 0 & F_{J-1} & E_{J-1} & D_{J-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & F_J & E_J+D_J \end{array}\right]$$

[13] On parle aussi de méthode d'Euler régressive (voir RAVIART et THOMAS [1983] page 173).

$$\Lambda = \begin{bmatrix} B_0 & A_0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ C_1 & B_1 & A_1 & 0 & & & & \vdots \\ 0 & C_2 & B_2 & A_2 & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & C_j & B_j & A_j & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & 0 & C_{J-1} & B_{J-1} & A_{J-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & C_J & B_J \end{bmatrix}$$

et

$$\mathbf{u}_n = \begin{bmatrix} u_0^n \\ u_1^n \\ u_2^n \\ \vdots \\ u_j^n \\ \vdots \\ u_{J-1}^n \\ u_J^n \end{bmatrix}$$

1.3.3 Programmation efficace des algorithmes

La solution numérique de (6) donnée par les θ-schémas s'obtient en résolvant le système suivant :

$$\Upsilon \mathbf{u}_{n+1} = -\Lambda \mathbf{u}_n \tag{15}$$

avec

$$\mathbf{u}_0 = \mathbf{1}_{J+1}$$

Nous avons donc

$$\mathbf{u}_{n+1} = -\Upsilon^{-1} \Lambda \mathbf{u}_n$$

Comme les matrices Υ et Λ sont constantes, l'algorithme numérique demande une seule inversion de matrice et $N+1$ multiplication de matrice.

Nous pouvons étendre l'analyse précédente au cas où les fonctions μ, λ, σ et α dépendent du temps. Posons

$$\begin{array}{rcl} \sigma_j^n & = & \sigma\left(t_n, x_j\right) \\ \mu_j^n & = & \mu\left(t_n, x_j\right) \\ \lambda_j^n & = & \lambda\left(t_n, x_j\right) \\ r_j^n & = & \alpha\left(t_n, x_j\right) \end{array}$$

Soient

$$\begin{array}{rcl} a_j^n & = & \frac{1}{2}\left(\frac{\sigma_j^n}{h}\right)^2 k + \frac{1}{2h}\left(\mu_j^n - \lambda_j^n \sigma_j^n\right) k \\ b_j^n & = & 1 - \left(\frac{\sigma_j^n}{h}\right)^2 k - r_j^n k \\ c_j^n & = & \frac{1}{2}\left(\frac{\sigma_j^n}{h}\right)^2 k - \frac{1}{2h}\left(\mu_j^n - \lambda_j^n \sigma_j^n\right) k \end{array}$$

et

$$\begin{array}{rcl} A_j^n & = & (1-\theta)\, a_j^n \\ B_j^n & = & (1-\theta)\left(b_j^n - 1\right) + 1 \\ C_j^n & = & (1-\theta)\, c_j^n \\ D_j^n & = & \theta a_j^n \\ E_j^n & = & \theta\left(b_j^n - 1\right) - 1 \\ F_j^n & = & \theta c_j^n \end{array}$$

La solution numérique de (5)[14] donnée par les θ-schémas s'obtient en résolvant le système suivant :

$$\Upsilon_n \mathbf{u}_n = -\Lambda_{n+1} \mathbf{u}_{n+1} \tag{16}$$

[14] Le θ-schéma correspond à

$$-\frac{u_j^{n+1} - u_j^n}{k} = \theta \mathcal{A}_j^n + (1-\theta)\, \mathcal{A}_j^{n+1}$$

avec

$$\mathbf{u}_N = \mathbf{1}_{J+1}$$

Les matrices Υ_n et Λ_n sont définies de la façon suivante :

$$\Upsilon_n = \begin{bmatrix} F_0^n + E_0^n & D_0^n & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ F_1^n & E_1^n & D_1^n & 0 & & & & \vdots \\ 0 & F_2^n & E_2^n & D_2^n & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & F_j^n & E_j^n & D_j^n & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & 0 & F_{J-1}^n & E_{J-1}^n & D_{J-1}^n \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & F_J^n & E_J^n + D_J^n \end{bmatrix}$$

$$\Lambda_n = \begin{bmatrix} B_0^n & A_0^n & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ C_1^n & B_1^n & A_1^n & 0 & & & & \vdots \\ 0 & C_2^n & B_2^n & A_2^n & 0 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & 0 & C_j^n & B_j^n & A_j^n & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & 0 & C_{J-1}^n & B_{J-1}^n & A_{J-1}^n \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & C_J^n & B_J^n \end{bmatrix}$$

Comme la matrice Υ_n est tridiagonale, nous pouvons résoudre (16) avec l'algorithme tridiagonal. Soit le système suivant

$$Rx = r \tag{17}$$

c'est-à-dire

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \\ & & \ddots & & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix}$$

Le système (17) est alors équivalent à

$$\begin{pmatrix} b'_1 & 0 & & & 0 \\ a_2 & b'_2 & 0 & & \\ & & \ddots & & \\ & & a_{n-1} & b'_{n-1} & 0 \\ 0 & & & a_n & b'_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r'_1 \\ r'_2 \\ \vdots \\ r'_{n-1} \\ r'_n \end{pmatrix}$$

avec

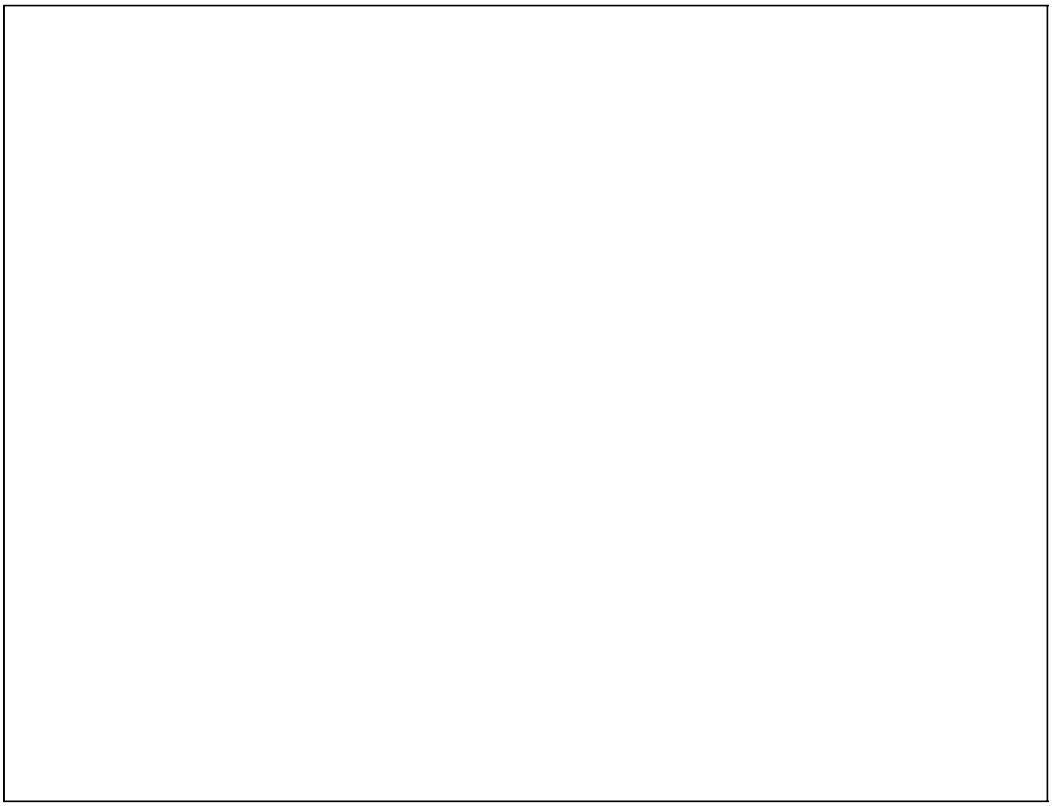
$$\begin{cases} b'_i &= b_i - \frac{c_i a_{i+1}}{b'_{i+1}} \quad \forall i = 1, \ldots, n-1 \\ b'_n &= b_n \end{cases}$$

et

$$\begin{cases} r'_i &= r_i - \frac{c_i r'_{i+1}}{b'_{i+1}} \quad \forall i = 1, \ldots, n-1 \\ r'_n &= r_n \end{cases}$$

Les solutions du système (17) sont alors

$$\begin{cases} x_1 &= \frac{r'_1}{b'_1} \quad \forall i = 1, \ldots, n-1 \\ x_i &= \frac{r'_i - a_i x_{i-1}}{b'_i} \quad \forall i = 1, \ldots, n-1 \end{cases}$$



Graphique 7

Nous pouvons tester l'efficacité relative des deux algorithmes LU et TRIDIAG. Considérons t_{LU} et t_{TRIDIAG} les temps de calcul pour le même système linéaire tridiagonal. Nous mesurons l'efficacité relative de l'algorithme tridiagonal par rapport à la factorisation LU par le rapport $\frac{t_{\text{LU}}}{t_{\text{TRIDIAG}}}$. Le graphique (7) montre l'évolution de ce rapport en fonction du rang du système. Ainsi, pour un système linéaire tridiagonal de dimension 700, l'algorithme tridiagonal est 360 fois plus rapide que l'algorithme LU. Il est donc préférable d'utiliser l'algorithme tridiagonal pour la résolution de (16).

Compte tenu des résultats précédents, une méthode efficace d'inversion de la matrice Υ est d'utiliser l'algorithme tridiagonal. Soit $\Upsilon^\star$ l'inverse de la matrice Υ. Nous avons

$$\Upsilon\Upsilon^\star = \mathbf{I}_{J+1}$$

Considérons la partition vectorielle $\Upsilon^\star = \left[\begin{array}{ccccc} \Upsilon_1^\star & \cdots & \Upsilon_j^\star & \cdots & \Upsilon_{J+1}^\star \end{array} \right]$. Calculer $\Upsilon_j^\star$ revient alors à résoudre le système linéaire tridiagonal suivant :

$$\Upsilon\Upsilon_j^\star = \mathbf{e}_j$$

Tridiagonal

■ **Objet**
Résolution du système linéaire tridiagonal $Rx = r$.

■ **Syntaxe**
x = Tridiagonal(RR,r);

■ **Entrées**

RR	matrice $n \times n$, matrice R.
r	vecteur $n \times 1$, vecteur r.

■ **Sorties**

x	vecteur $n \times 1$, solution x.

Tridiag_inv

■ **Objet**
Calcule l'inverse d'une matrice tridigonale A.

■ **Syntaxe**
A_inv = Tridiag_inv(A);

■ **Entrées**

A	matrice $n \times n$, matrice A.

■ **Sorties**

A_inv	matrice $n \times n$, matrice A^{-1}.

Voici le code des procédures `Tridiagonal` et `Tridiagonal` :

```
proc Tridiagonal(RR,r);
  local N,x,b_prime,r_prime,i,_aux;
  N = rows(RR);
  x = zeros(n,1); b_prime = zeros(N,1); r_prime = zeros(N,1);
  b_prime[n] = RR[n,n]; r_prime[n] = r[n];
  i = N-1;
  do while i>0;
    _aux = RR[i,i+1]/b_prime[i+1];
    b_prime[i]=RR[i,i]-_aux*RR[i+1,i];
    r_prime[i]=r[i]-_aux*r_prime[i+1];
    i=i-1;
  endo;
  x[1]=r_prime[1]/b_prime[1];
  i=2;
  do until i>n;
    x[i]=(r_prime[i]-RR[i,i-1]*x[i-1])/b_prime[i];
    i=i+1;
  endo;
  retp(x);
endp;

proc Tridiag_inv(A);
  local N,Id,A_inv,j;
  N = rows(A); Id = eye(N); A_inv = zeros(N,N);
  j=1;
  do until j>N;
    A_inv[.,j] = Tridiagonal(A,Id[.,j]);
    j=j+1;
  endo;
  retp(A_inv);
endp;
```

EDP2

■ **Objet**
Résolution de l'équation différentielle partielle parabolique du modèle de structure par terme à un seul facteur.

■ **Syntaxe**
{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,theta,cn);

■ **Entrées**

&mu	pointeur d'une procédure qui calcule la fonction μ.
&sigma	pointeur d'une procédure qui calcule la fonction σ.
&lambda	pointeur d'une procédure qui calcule la fonction λ.
&alpha	pointeur d'une procédure qui calcule la fonction α.
theta	scalaire, valeur de θ.
cn	vecteur 6×1, paramètres de contrôle.
	cn[1] : valeur inférieure x^- de l'intervalle d'approximation de X.
	cn[2] : valeur supérieure x^+ de l'intervalle d'approximation de X.
	cn[3] : nombre de points J de discrétisation dans l'espace.
	cn[4] : valeur de $\tau_{\max}$.
	cn[5] : Nombre de points N de discrétisation dans le temps.
	cn[6] : 0 si le processus ne dépend pas de la variable t et 1 s'il dépend de t.

■ **Sorties**

t	vecteur $(N+1) \times 1$, valeurs de τ_n.
x	vecteur $(J+1) \times 1$, valeurs de x_j.
u	matrice $(J+1) \times (N+1)$, valeurs de u_j^n.

■ **Variables globales**

_print	1 pour l'affichage écran.

■ **Remarque**

La procédure `EDP2` exploite la vectorisation de GAUSS. Les procédures qui définissent les fonctions doivent donc utiliser les opérateurs élément par élément.

Pour `cn[6] = 0`, les procédures sont de la forme

```
proc mu(x);
  local y;
  y =
  retp(y);
endp;
```

Pour `cn[6] = 1`, les procédures sont de la forme

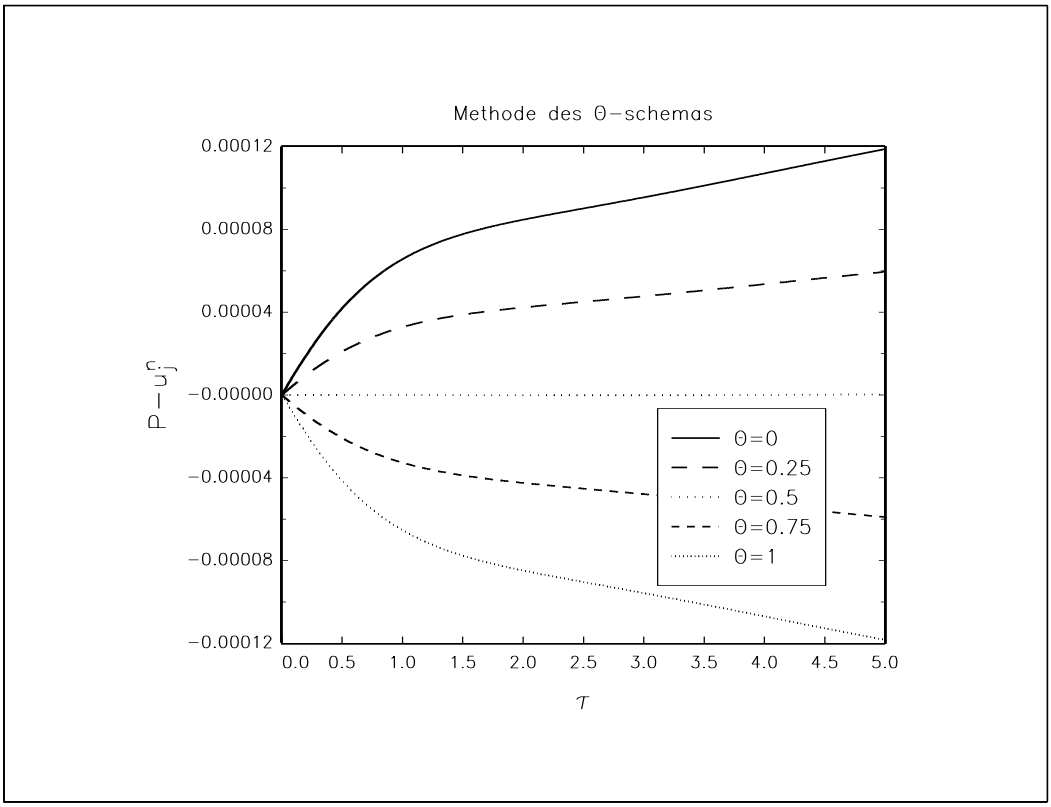
```
proc mu(t,x);
  local y;
  y =
  retp(y);
endp;
```

Voici le code de la procédure `EDP2` :

```
declare external _print;

proc (3) = EDP2(mu,sigma,lambda,alpha,theta,cn);
  local mu:proc,sigma:proc,lambda:proc,alpha:proc;
  local x_min,x_max,J,tau,N,Temps,h,k,old,t,x,r_j;
  local d0,d1,d2,_Upsilon,_Lambda;
  local sigma_j,mu_j,lambda_j,_s1,_s2,_s3,a_j,b_j,c_j,d_j,e_j,f_j;
  local M,u_n,u,nn,t1,t2,t_n;
  x_min = cn[1]; x_max = cn[2]; J = cn[3];
  tau = cn[4]; N = cn[5]; Temps = cn[6];
  k = tau/N; h = (x_max-x_min)/J;
  old = csrtype(0);
  if _print == 1;
    call EDP2_affichage(N|J|k|h|theta);
  endif;
  if Temps /= 1;
    Temps = 0;
  endif;
  t = seqa(0,k,N+1); x = seqa(x_min,h,J+1);
```

```
d0 = seqa(1,1,J+1); d1 = seqa(2,1,J); d2 = seqa(1,1,J);
_Upsilon = zeros(J+1,J+1);
_Lambda = zeros(J+1,J+1);

if temps == 0;
  sigma_j = sigma(x); mu_j = mu(x); lambda_j = lambda(x); r_j = alpha(x);
  _s1 = (sigma_j)^2*k/(h^2); _s2 = (mu_j-lambda_j.*sigma_j)*k/h;_s3 = 1-theta;
  a_j = 0.5*(_s1+_s2);
  b_j = 1-_s1-r_j*k;
  c_j = 0.5*(_s1-_s2);
  d_j = a_j; e_j = b_j-2; f_j = c_j;
  clear sigma_j,mu_j,lambda_j;
  A_j = _s3*a_j; B_j = _s3*(b_j-1)+1; C_j = _s3*c_j;
  D_j = theta*d_j; E_j = theta*(e_j+1)-1; F_j = theta*f_j;
  E_j[1] = F_j[1]+E_j[1]; E_j[J+1] = E_j[J+1]+D_j[J+1];
  _Upsilon = Tridiag_create(F_j,E_j,D_j);
  _Lambda = Tridiag_create(C_j,B_j,A_j);
  clear A_j,B_j,C_j,D_j,E_j,F_j;
  M = -inv(_Upsilon)*_Lambda; /* M = -Tridiag_inv(_Upsilon)*_Lambda; */
  clear _Upsilon,_Lambda;
  u_n = ones(J+1,1);
  u = zeros(J+1,N+1); u[.,1] = u_n;
  nn = 1;
  t1 = hsec;
  do until nn>N;
    u_n = M*u_n;
    u[.,nn+1] = u_n;
    if _print == 1;
      t2 = hsec;
      call EDP2_print(t1,t2,nn,N);
    endif;
    nn = nn+1;
  endo;
else;
  u_n = ones(J+1,1);
  u = zeros(J+1,N+1); u[.,N+1] = u_n;
  t1 = hsec;
  t_n = t[N+1];
  sigma_j = sigma(t_n,x); mu_j = mu(t_n,x); lambda_j = lambda(t_n,x);
  r_j = alpha(t_n,x);
  _s1 = (sigma_j)^2*k/(h^2); _s2 = (mu_j-lambda_j.*sigma_j)*k/h;
  _s3 = 1-theta;
  clear sigma_j,mu_j,lambda_j;
  a_j = 0.5*(_s1+_s2);
  b_j = 1-_s1-r_j*k;
  c_j = 0.5*(_s1-_s2);
  nn = N;
  do while nn>=1;
    A_j = _s3*a_j; B_j = _s3*(b_j-1)+1; C_j = _s3*c_j;
    _Lambda = Tridiag_create(C_j,B_j,A_j);
    t_n = t[nn];
    sigma_j = sigma(t_n,x); mu_j = mu(t_n,x); lambda_j = lambda(t_n,x);
    r_j = alpha(t_n,x);
    _s1 = (sigma_j)^2*k/(h^2); _s2 = (mu_j-lambda_j.*sigma_j)*k/h;
    a_j = 0.5*(_s1+_s2);
    b_j = 1-_s1-r_j*k;
    c_j = 0.5*(_s1-_s2);
    clear sigma_j,mu_j,lambda_j;
    D_j = theta*a_j; E_j = theta*(b_j-1)-1; F_j = theta*c_j;
    E_j[1] = F_j[1]+E_j[1]; E_j[J+1] = E_j[J+1]+D_j[J+1];
```

```
      _Upsilon = Tridiag_create(F_j,E_j,D_j);
      clear D_j,E_j,F_j;
      u_n = Tridiagonal(_Upsilon,-_Lambda*u_n);
      clear _Upsilon,_Lambda;
      u[.,nn] = u_n;
      if _print == 1;
        t2 = hsec;
        call EDP2_print(t1,t2,N-nn+1,N);
      endif;
      nn = nn-1;
    endo;
    u=rev(u')';
  endif;
 retp(t,x,u);
endp;


proc (1) = Tridiag_create(A0,A1,A2);
  local m,A,i;
  m = rows(A1); A = zeros(m,m);
  A[1,1] = A1[1]; A[1,2] = A2[1];
  i=2;
  do until i>m-1;
    A[i,i-1] = A0[i];
    A[i,i] = A1[i];
    A[i,i+1] = A2[i];
    i=i+1;
  endo;
  A[m,m-1] = A0[m]; A[m,m] = A1[m];
  retp(A);
endp;


proc (0) = EDP2_affichage(cn);
  local q0,q1,q2,w1,w2,w3,w4;
  cls;
  q0 = 196*ones(1,70); q1 = zeros(1,70); q2 = zeros(1,5);
  w1 = chrs(q2~218~q0~191); w2 = chrs(q2~179~q1~179); w3 = chrs(q2~192~q0~217);
  print w1; print w2; print w2; print w2; print w2; print w2; print w3;
  locate 3,17; print "Resolution de la valorisation des coupons zeros";
  locate 4,13;
  print "dans le modele financier d'arbitrage a une variable d'etat";
  locate 5,23; print "par la methode des theta-schemas.";
  locate 10,1;
  q0 = 196*ones(1,50); q1 = zeros(1,50); q2 = zeros(1,16);
  w1 = chrs(q2~218~q0~191); w2 = chrs(q2~179~q1~179); w3 = chrs(q2~192~q0~217);
  w4 = chrs(q2~195~q0~180);
  print w1; print w2; print w2; print w4; print w2; print w4;
  print w2; print w2; print w2; print w3;
  locate 11,22; print ftos(cn[1],"N : %lf    ",7,0);
  locate 11,45; print ftos(cn[2],"J : %lf    ",7,0);
  locate 12,22; print ftos(cn[3],"k : %lf    ",7,5);
  locate 12,45; print ftos(cn[4],"h : %lf    ",7,5);
  locate 14,22; print ftos(cn[5],"THETA : %lf    ",7,5);
  retp;
endp;


proc (0) = EDP2_print(t1,t2,nn,N);
  local t3;
```



Graphique 8

```
  t3=(t2-t1)/6000;
  locate 16,22; print ftos(t3,"Temps ecoule : %lf minutes",5,2);
  t3=t3/nn*(N-nn);
  locate 17,22; print ftos(t3,"Temps restant (estimation) : %lf minutes",5,2);
  locate 18,22; print ftos(nn,"Iteration no %lf",5,0);
  retp;
endp;
```

Considérons le modèle de vasicek. La variable d'état est le taux d'intérêt instantané r. La dynamique de celui-ci est donnée par l'équation différentielle stochastique suivante

$$dr = (0.10 - r)\ dt + 0.1\, dW$$

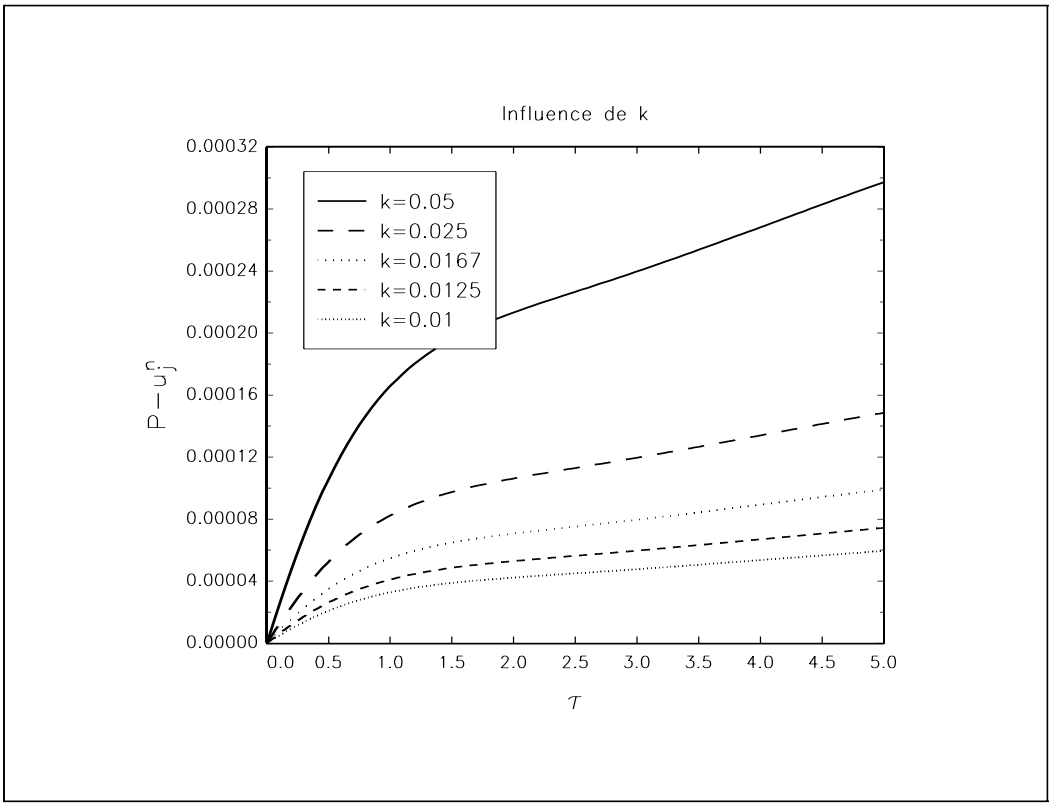
Nous posons $\lambda(r) = 0.1$. Le graphique 8 représente la différence entre la valeur du prix du coupon zéro obtenu par la formule de Vasicek (P) et celle donnée par la méthode numérique des θ-schémas (u_i^n) pour $r = 0.1$. Nous avons pris $r^- = -1$, $r^+ = 1$, $\tau_{\max} = 5$, $h = 0.02$ et $k = 0.01$. Nous remarquons que la différence $P - u_i^n$ dépend de la valeur θ. Le différence est la plus faible pour $\theta = \frac{1}{2}$, ce qui correspond au schéma de Crank-Nicholson.

```
new;
library pgraph;
#include tridiag.src;
#include edp2.src;

proc mu(x);
  retp(0.10-x);
endp;

proc sigma(x);
  retp(0.1);
endp;
```

```
proc lambda(x);
  retp(0.1);
endp;

proc alpha(x);
  retp(x);
endp;

_print =1;
cn = -1|1|100|5|500|0;

{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,0,cn);
r = x[56]; sol=u[56,.]';
{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,0.25,cn); sol=sol~u[56,.]';
{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,0.5,cn); sol=sol~u[56,.]';
{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,0.75,cn); sol=sol~u[56,.]';
{t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,1,cn); sol=sol~u[56,.]';

P = vasicek(t,1,0.10,0.1,r,0.1);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _plwidth = 5; _paxht = 0.2; _pnum = 2;
  title("Methode des \202Q\201-schemas");
  xlabel("\202t\201"); ylabel("P-u[n]]j[");
  _plegstr = "\202Q\201=0\000\202Q\201=0.25\000\202Q\201=0.5\000"\
             "\202Q\201=0.75\000\202Q\201=1";
  _plegctl = { 2 6 6 1.5};
  graphprt("-c=1 -cf=edp1.eps -w=5");
  xy(t,P-sol);


proc vasicek(tau,a,b,sigma,r0,lamda);
  local b_prime,taux_long,x,c;
  b_prime=b-lamda.*sigma./a;
  taux_long=b_prime-(sigma^2)./(2*a^2);
  x=(1-exp(-a.*tau))./a;
  c=exp(-taux_long.*tau+(taux_long-r0).*x-(x^2).*(sigma^2)./(4*a));
  retp(c);
endp;
```

Dans la pratique, nous constatons que les erreurs sont minimales pour les valeurs de x proches du milieu du segment $[x^-, x^+]$. Nous pouvons en tirer l'enseignement suivant : si nous voulons connaître la trajectoire de $P(t,x)$ pour une valeur particulière $x^\star$, il est judicieux de choisir l'intervalle $[x^-, x^+]$ tel que $x^\star$ corresponde au milieu du segment. Nous remarquons aussi que la relation entre la précision des résultats et le pas h est difficile à appréhender. En effet, à partir d'un certain seuil $h^\star$, la différence $P - u_i^n$ augmente lorsque h diminue. L'approximation des dérivées par la méthode des différences n'est plus pertinente pour des valeurs de h très faibles d'un point de vue informatique (problèmes d'arrondi). Nous ne constatons pas ce phénomène pour le pas k. Le graphique 9 représente la différence $P - u_i^n$ en fonction de k pour $\theta = 0.25$ (les autres valeurs sont les mêmes que celles utilisées pour le programme précédent).

```
new;
library pgraph;
#include tridiag.src;
#include edp2.src;

proc mu(x);
  retp(0.10-x);
endp;

proc sigma(x);
```



Graphique 9

```
  retp(0.1);
endp;

proc lambda(x);
  retp(0.1);
endp;

proc alpha(x);
  retp(x);
endp;

_print =1;

np = seqa(100,100,5);
axe1 = miss(zeros(501,5),0);
sol1 = miss(zeros(501,5),0);

i=1;
do until i>5;
  cn = -1|1|100|5|np[i]|0;
  {t,x,u} = EDP2(&mu,&sigma,&lambda,&alpha,0.25,cn);
  r = x[56];
  P = vasicek(t,1,0.10,0.1,r,0.1);
  sol1[1:rows(P),i] = P-u[56,.]';
  axe1[1:rows(t),i] = t;
  i = i+1;
endo;

graphset;
  fonts("simplex simgrma");
```

```
  _pdate = ""; _plwidth = 5; _paxht = 0.2; _pnum = 2;
  title("Influence de k");
  xlabel("\202t\201"); ylabel("P-u[n]]j[");
  _plegstr = "k=0.05\000k=0.025\000k=0.0167\000"\
             "k=0.0125\000k=0.01";
  _plegctl = { 2 6 2 4};
  graphprt("-c=1 -cf=edp2.eps -w=5");
  xy(axe1,sol1);


proc vasicek(tau,a,b,sigma,r0,lamda);
  local b_prime,taux_long,x,c;
  b_prime=b-lamda.*sigma./a;
  taux_long=b_prime-(sigma^2)./(2*a^2);
  x=(1-exp(-a.*tau))./a;
  c=exp(-taux_long.*tau+(taux_long-r0).*x-(x^2).*(sigma^2)./(4*a));
  retp(c);
endp;
```

1.4 Résolution d'un problème numérique fréquent en finance

La valorisation des actifs contingents conduit souvent à des solutions complexes. Pour obtenir cette solution, nous devons parfois alors évaluer une intégrale d'une fonction, qui dépend aussi d'une intégrale. C'est notament le cas du théorème de représentation de Feynman-Kac lorsque le taux d'intérêt et le revenu distribué ne sont pas constants. C'est aussi le cas du modèle de Heath, Jarrow et Morton [1992][15]. La solution du modèle nécessite le calcul de la fonction suivante

$$\int_{t_0}^{t} \beta(u,t) \int_{u}^{t} \beta(v,u)\, du\, dv$$

Cette fonction est difficile à évaluer parce que la première intégrale dépend de la solution d'une deuxième intégrale et que la borne inférieure d'intégration de la deuxième intégrale est aussi une variable d'intégration. Mises à part les fonctions simples (par exemple, la fonction exponentielle), il n'existe pas de solution symbolique à ce problème. Nous pourrions penser que la résolution est facile. Cependant, nous sommes confrontés à un problème de programmation récursive. Nous ne pouvons donc pas utiliser les procédures traditionnelles d'intégration que nous trouvons dans IMSL, NAG, etc. De même, les procédures `intquad1` et `intsimp` de GAUSS ne sont pas adaptés à ce genre de problème numérique. Mais les variables externes de GAUSS nous permettent de "simuler" cette récursivité. Le code suivant est basé sur l'algorithme de Simpson.

HJM

■ **Objet**
Approximation numérique de $\int_{t_0}^{t} \beta(u,t) \int_{u}^{t} \beta(v,u)\, du\, dv$.

■ **Syntaxe**
I = HJM(&beta,t0,t);

■ **Entrées**

&beta	pointeur d'une procédure qui calcule la fonction $\beta(t,T)$.
t0	scalaire, valeur de t_0.
t	scalaire, valeur de t.

■ **Sorties**

I	scalaire, valeur approchée de l'intégrale.

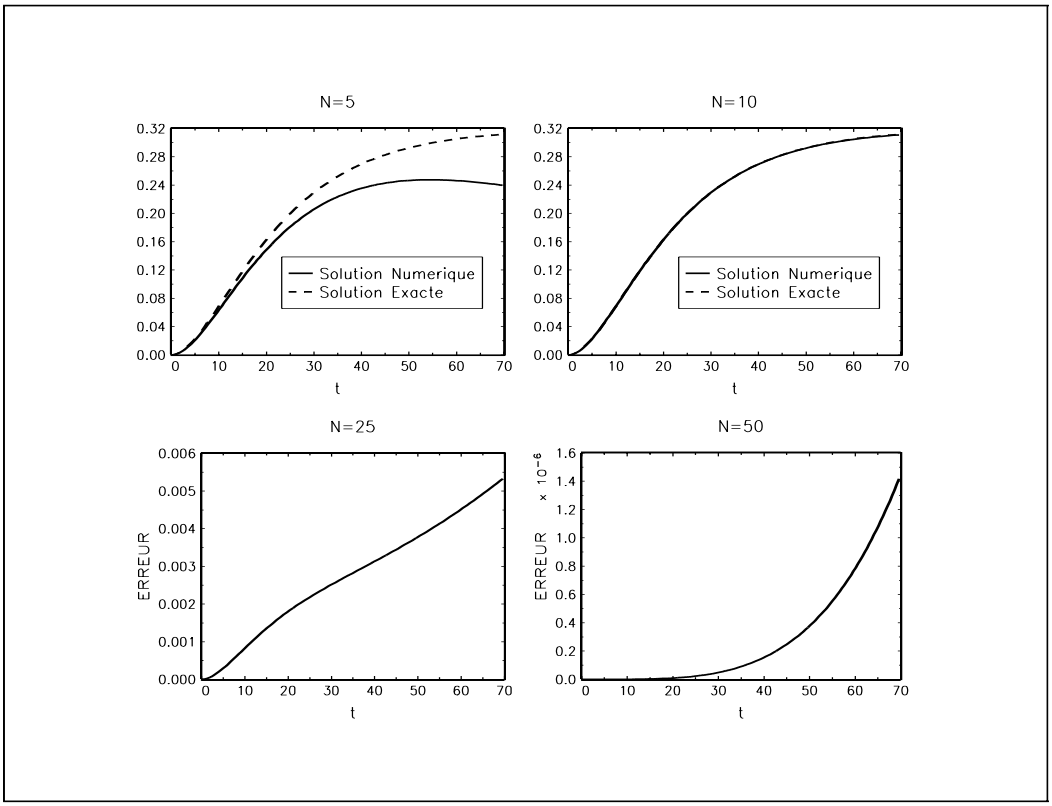
■ **Variables globales**

_n	scalaire, Nombre de points d'approximation pour l'algorithme de Simpson (défaut = 20).

Voici le code de la procédure `HJM` :

[15] Heath, D., R. Jarrow et A. Morton [1992], Bond pricing and the term structure of interest rates: a new methodology for contingent claim valuation, *Econometrica*, **60**, 77-105.

```
declare matrix _n = 20; /* Nombre de points d'approximation
                           pour l'algorithme de SIMPSON */

/* Variables globales */

declare matrix _beta;  /* Fonction beta(t,T) */
declare matrix _A1;
declare matrix _A2;

proc _HJM_1(v);
  local f,res,u;
  u = _A1;
  f = _beta;
  local f:proc;
  res = f(u,v);
  retp(res);
endp;

proc _HJM_2(u,t);
  local h,v,s,I,j;
  _A1 = u;
  h = (t-u)/_n;
  v = seqa(u,h,_n+1);
  s = 1|reshape(4~2,1,_n-1)'|1;
  I = 0;
  j = 1;
  do until j > _n+1;
    I = I+h*(s[j]/3)*_HJM_1(v[j]);
    j = j+1;
  endo;
  retp(I);
endp;

proc _HJM_3(u);
  local res,t;
  t = _A2;
  res = _HJM_2(u,t);
  retp(res);
endp;

proc _HJM_4(u);
  local f,res,t;
  t = _A2;
  f = _beta;
  local f:proc;
  res = f(u,t)*_HJM_3(u);
  retp(res);
endp;

proc HJM(beta,t0,t);
  local beta:proc;
  _beta = &beta;
  local h,u,s,I,j;
  _A2 = t;
  h = (t-t0)/_n;
  u = seqa(t0,h,_n+1);
  s = 1|reshape(4~2,1,_n-1)'|1;
  I = 0;
  j = 1;
  do until j > _n+1;
```



Graphique 10

```
    I = I+h*(s[j]/3)*_HJM_4(u[j]);
    j = j+1;
  endo;
  retp(I);
endp;
```

Reprenons l'exemple donné par Heath, Jarrow et Morton [1992] page 91. Ils utilisent la fonction $\beta(t,T) = \sigma_2 \exp\left[-\frac{\lambda}{2}(T-t)\right]$ et obtiennent

$$\int_0^t \beta(u,t) \int_u^t \beta(v,u)\, du\, dv = -2\frac{\sigma_2^2}{\lambda^2}\left[\left(1-e^{-\lambda t}\right) - 2\left(1-e^{-\frac{\lambda}{2}t}\right)\right]$$

Même si la fonction est en exponentielle, la solution n'est pas évidente à trouver ! L'exemple suivant permet de comparer la solution symbolique des auteurs avec la solution numérique obtenue par la procédure `HJM`. Nous posons $\sigma_2 = 0.05$ et $\lambda = 0.125$ et t varie de 0 à 70. La différence entre la solution analytique et celle numérique dépend bien sûr du nombre de points d'approximation de l'algorithme de Simpson.

```
new;
library pgraph;

#include HJM.src;

sigma2 = 0.05;
lambda = 0.125;

proc beta(t,_T);
  local res;
  res = sigma2.*exp(-lambda/2.*(_T-t));
  retp(res);
endp;
```

```
proc solution(t);
  local y;
  y = -2*(sigma2/lambda)^2*( (1-exp(-lambda*t)) - 2*(1-exp(-lambda/2*t)) );
  retp(y);
endp;

N = 5|10|25|50;
ch = "Sol";

j = 1;
do until j > 4;

_N = N[j];
_T = 140;
t = seqa(0.1,0.5,_T);
r = zeros(_T,2);
i = 1;
do until i > _T;
  r[i,.] = HJM(&beta,0,t[i])~solution(t[i]);
  i = i+1;
endo;
call varput(r,ch$+ftos(j,"%lf",1,0));
j = j+1;
endo;

graphset;
begwind;
window(2,2,0);
setwind(1);
  _ptitlht = 0.25; _paxht = 0.25; _pnum = 2; _pnumht = 0.20; _plwidth = 5;
  title("N=5");
  _plegstr = "Solution Numerique\0Solution Exacte";
  _plegctl = { 2 6 3.5 2};
  xlabel("t");
  r = varget("Sol1");
  xy(t,r);
nextwind;
  title("N=10");
  r = varget("Sol2");
  xy(t,r);
nextwind;
  graphset;
  _ptitlht = 0.25; _paxht = 0.25; _pnum = 2; _pnumht = 0.20; _plwidth = 5;
  title("N=25");
  xlabel("t"); ylabel("ERREUR");
  r = varget("Sol3");
  xy(t,r[.,2]-r[.,1]);
nextwind;
  title("N=50");
  r = varget("Sol4");
  xy(t,r[.,2]-r[.,1]);
  graphprt("-c=1 -cf=hjm.eps -w=5");
endwind;
```

1.5 Calcul de la volatilité implicite d'une option d'achat sur Future

Le but de cette sous-section est de montrer la simplicité et la rapidité du langage GAUSS. Considérons le problème du calcul de la volatilité implicite. Nous notons K le prix d'exercice de l'option, F_0 la valeur du future, σ la volatilité implicite, τ la maturité de l'option et r le taux d'intérêt. Dans le modèle de Black, le prix de l'option

d'achat est donné par la formule suivante

$$C = F_0 e^{-r\tau} \Phi(d_1) - K e^{-r\tau} \Phi(d_2) \tag{18}$$

avec Φ la fonction de répartition de la loi normale centrée et réduite et

$$\begin{aligned} d_1 &= \frac{1}{\sigma\sqrt{\tau}} \ln \frac{F_0}{K} + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= \frac{1}{\sigma\sqrt{\tau}} \ln \frac{F_0}{K} - \frac{1}{2}\sigma\sqrt{\tau} \end{aligned}$$

Le prix théorique de l'option dépend donc de 4 paramètres parfaitement objectifs (K, F_0, τ et r) et d'un paramètre *subjectif* (la mesure de σ dépend de la méthode utilisée, de la période considérée, etc.). Soit $C^\bullet$ la valeur observée de l'option sur le marché. La volatilité implicite correspond alors à la valeur de σ telle que le prix théorique C soit égal au prix observé $C^\bullet$:

$$C(\sigma) = C^\bullet \tag{19}$$

Pour résoudre cette équation, nous pouvons employer différents algorithmes : Newton, Broyden, SQRF, Wang-Tewarson, GC, etc. Mais, cherchons à résoudre ce problème de façon simple. Considérons la fonction f définie par

$$f(\sigma) = C(\sigma) - C^\bullet \tag{20}$$

Pour résoudre l'équation (19), nous pouvons chercher la valeur de σ qui annule la fonction f. Supposons deux scalaires a et b tels que

$$f(a) < 0 < f(b)$$

alors la solution de l'équation $f(\sigma) = 0$ vérifie $a < \sigma < b$. Considérons le point c tel que $c = \frac{a+b}{2}$. Alors si $f(c) \leq 0$, la solution appartient à l'intervalle $[c, b]$. Dans le cas contraire, la solution appartient à l'intervalle $[a, c]$. Nous avons réduit l'intervalle qui contient la solution de moitié. Nous pouvons recommencer l'opération. Dans ce cas, la longueur du nouvel intervalle sera quatre fois plus petite que l'intervalle originel. En réitérant l'opération un certain nombre de fois, nous pouvons trouver un intervalle relativement petit qui encadre la solution. Cet algorithme est appelé méthode de la bisection. La description précédente est valide si la fonction f est strictement croissante sur l'intervalle $[a, b]$. C'est vérifié dans notre cas puisque le Vega de l'option est strictement positif (résultat page 245 de Dalbarade [1990], Mathématiques des marchés financiers, Editions ESKA, Paris). Cela implique que la solution obtenue est unique.

Volimp

■ **Objet**

Calcul de la volatilité implicite d'une option d'achat sur future basé sur le modèle de Black.

■ **Syntaxe**

sigma = Volimp(Prime,K,F0,tau,r,a,b);

■ **Entrées**

Prime	vecteur $N \times 1$, prix des options sur le marché.
K	vecteur $N \times 1$, prix d'exercice.
F0	vecteur $N \times 1$, valeurs des futures.
tau	vecteur $N \times 1$, maturités des options.
r	vecteur $N \times 1$, taux d'intérêt.
a	scalaire, valeur de a.
b	scalaire, valeur de b.

■ **Sorties**

sigma	vecteur $N \times 1$, volatilités implicites.

Voici le code de la procédure `Volimp` :

```
1.  declare matrix _volimp_K;
2.  declare matrix _volimp_F0;
3.  declare matrix _volimp_tau;
4.  declare matrix _volimp_r;
5.  declare matrix _volimp_Prime;
6.  declare matrix _convergence = 0.001;
7.
```

```
 8.
 9.  proc Volimp(Prime,K,F0,tau,r,a,b);
10.
11.    _volimp_Prime = Prime;
12.    _volimp_K = K;
13.    _volimp_F0 = F0;
14.    _volimp_tau = tau;
15.    _volimp_r = r;
16.
17.    retp( bisection(&_CALL_Future,a,b) );
18.  endp;
19.
20.
21.  proc CALL_future(K,F0,sigma,tau,r);
22.    local a,b,d1,d2,prime;
23.
24.    a = sigma.*sqrt(tau);
25.    d1 = ln(F0./K)./a + 0.5*a;
26.    d2 = d1 - a;
27.    b = exp(-r.*tau);
28.
29.
30.    prime = F0.*b.*cdfn(d1) - K.*b.*cdfn(d2);
31.
32.    retp(prime);
33.  endp;
34.
35.
36.  proc _CALL_Future(sigma);
37.     retp(Call_Future(_volimp_K,_volimp_F0,sigma,_volimp_tau,_volimp_r)
38.           - _volimp_Prime);
39.  endp;
40.
41.
42.  proc bisection(&f,a,b);
43.    local f:proc;
44.    local ya,yb,Nobs,c,yc,indx1,indx2;
45.
46.    ya = f(a); yb = f(b);
47.    Nobs = rows(ya);
48.
49.    if ( maxc(ya) > 0 ) or ( minc(yb) < 0 );
50.      ERRORLOG "erreur : mauvaise initialisation de a et b.";
51.      end;
52.    endif;
53.
54.    do while maxc(abs(a-b)) > _convergence;
55.      c = (a+b)/2;
56.      yc = f(c);
57.      indx1 = yc.<0;
58.      indx2 = 1 - indx1;
59.      a = indx1.*c + indx2.*a;
60.      b = indx1.*b + indx2.*c;
61.    endo;
62.    c = (a+b)/2;
63.    retp(c);
64.  endp;
```

Comment est construite la procédure `Volimp` ? Elle comprend 7 entrées pour définir les caractéristiques des options et l'intervalle $[a, b]$ dans lequel on cherche les volatilités implicites. La convergence de l'algorithme est définie par une variable externe `_convergence`. Les lignes 21 à 33 définissent la prime théorique d'une option

d'achat sur future. La procédure `CALL_future` correspond donc à la formule (18). Elle est vectorisée puisque nous utilisons les opérateurs élément par élément. Les lignes 36 à 36 définissent la fonction $f(\sigma) = C(\sigma) - C^{\bullet}$. C'est une fonction à une seule variable. C'est pour cela que nous employons 5 variables externes (`_volimp_K`, `_volimp_F0`, etc.). Elles permettent de prendre en compte les valeurs de $C^{\bullet}$, K, F_0, τ et r qui sont données par l'utilisateur. Les lignes 42 à 64 définissent l'algorithme de la bisection. La programmation de cet algorithme a été vectorisée. Cela veut dire en particulier qu'un seul appel de la procédure (et non N appels) permet de calculer les volatilités implicites des N options. Nous vérifions en premier que l'intervalle $[a,b]$ encadre les solutions (lignes 49 à 52). La réduction **des intervalles** correspond aux lignes 54 à 61. La première intération transforme l'intervalle $[a,b]$ en intervalles plus petits

$$[a,b] \longrightarrow \begin{bmatrix} a_1, b_1 \\ a_2, b_2 \\ \vdots \\ a_{N-1}, b_{N-1} \\ a_N, b_N \end{bmatrix}$$

La solution `c` de la ligne 63 est donc un vecteur de dimension $N \times 1$.

L'exemple suivant considère le calcul de 10000 volatilités implicites. la convergence est fixée à 0.00001. Cela veut dire que la dernière itération de l'algorithme de la bisection vérifie que

$$|a_i - b_i| \leq 0.00001 \qquad \forall\, i = 1, \ldots, 10000$$

Les points de départ de l'algotithme sont $a = 0.001$ et $b = 0.5$.

```
new;
#include volimp.src;

load data;

Prime = data[.,1];
K = data[.,2];
F0 = data[.,3];
r = data[.,4];
tau = data[.,5];

tt = hsec;

output file = volimp.out reset;

_convergence = 0.00001;

sigma = Volimp(Prime,K,F0,tau,r,0.001,0.5);

print ftos((hsec-tt)/100,"Temps de calcul : %lf secondes",6,2);

output off;


Temps de calcul :   9.89 secondes
```

Moins de 10 secondes sont nécessaires pour le calcul des 10000 volatilités implicites. Cela est relativement rapide. Et surtout, c'est simple à programmer.

2 Modélisation des séries temporelles appliquées à la finance

TSM (**T**ime **S**eries **M**odelling) est un module GAUSS pour la modélisation (représentation, estimation et prévision) des séries temporelles multidimensionnelles dans les domaines du temps, des fréquences et du temps-fréquences (ou des ondelettes). C'est un logiciel destiné aux analystes financiers, aux économistes et aux ingénieurs (théorie du signal et contrôle des processus).

Le cadre de référence de TSM est la théorie du système linéaire. Les modèles qui sont donc principalement étudiés par TSM sont les processus markoviens linéaires d'ordre 1 (par exemple, les modèles espace état, les

processus VARMA et VARX ou les modèles structurels de Harvey). Le choix d'une modélisation linéaire peut paraître limité. Mais elle s'avère intéressante dans de nombreux cas. Tout d'abord, parce que les temps de calcul nécessaires pour l'analyse des systèmes non linéaires sont parfois prohibitifs[16]. Ensuite, comme de nombreux outils sont disponibles, l'interprétation des résultats peut aboutir à des enseignements intéressants. Cette position est celle de Sargent et Hansen dans leur ouvrage non publié "Recursive linear model of dynamic economies". Pour estimer les modèles markoviens linéaires, deux méthodes d'estimation sont proposées : celle du maximum de vraisemblance et celle des moments généralisés. Elles permettent de prendre en compte facilement des restrictions linéaires[17]. Des outils de modélisation, développés dans les trois domaines – temps, fréquences et ondelettes, complètent la librarie TSM.

2.1 Note sur les procédures de TSM

Les procédures de TSM exploitent pleinement les capacités de GAUSS. Dans certains cas, l'emploi d'algorithmes spécifiques peut fortement diminuer les temps de calcul. C'est notamment le cas de la méthode du maximum de vraisemblance dans le domaine des fréquences (ou méthode de Whittle) qui utilise un gradient analytique. C'est pourquoi l'estimation suivante du processus ARFIMA(1,d,1) à partir de 2500 observations ne prend que 25 secondes en utilisant un point de départ $(0.5, 0, -0.25, 3)$ relativement éloigné de la vraie valeur $(0.75, 0.5, 0, 2)$.

```
new;
library tsm,optmum;

rndseed 123;

Nobs = 2500;

u = rndn(Nobs,1)*2;
u = u~(0|trimr(u,0,1));
u = u*(1|-0.5);
y = recserar(u,0,0.75);    /* Simulation d'un processus ARIMA(1,0,1) */

cnt = 0|0|0|0;             /* Pas de contraintes sur les parametres  */

_tsm_Mcov = 1;
_fourier = 0;
_print = 1;

output file = arfima.out reset;

t0 = hsec;

sv = 0.5|0|-0.25|3;
{theta,stderr,Mcov,Logl} = arfima(y,1,1,sv,cnt);

print;
print ftos((hsec-t0)/100,"Temps de calcul : %lf secondes",5,3);

output off;
```

Voici le contenu du fichier *arfima.out* :

```
Total observations:                                 2500
Usable observations:                                2500
Number of parameters to be estimated:                  4
Degrees of freedom:                                 2496
Value of the maximized log-likelihood function:  -8599.01965


Parameters      estimates       std.err.      t-statistic         p-value
```

[16] Les systèmes non linéaires posent aussi des problèmes de convergence d'estimation (en particulier, le filtre de Kalman étendu).

[17] La transformation d'un système de restrictions linéaires explicites en système de restrictions linéaires implicites permet de ramener un problème de maximisation sous contraintes en un problème de maximisation libre.

```
------------------------------------------------------------------------------
phi1            0.733707        0.025945        28.279186        0.000000
theta1          0.459602        0.033994        13.520141        0.000000
d              -0.011222        0.017961        -0.624784        0.532170
sigma           1.974721        0.021819        90.506270        0.000000

Covariance matrix: inverse of the negative Hessian.

Temps de calcul : 25.650 secondes
```

Le tableau suivant indique les temps de calcul en fonction du nombre d'observations pour l'estimation FLS (flexible least squares) d'un système 1 sortie - 5 entrées.

Nombre d'observations	500	2500	5000	10000	20000
Temps de calcul (en secondes)	0.22	1.21	2.58	5.27	10.71[18]

Cette rapidité des temps de calcul est due à l'utilisation d'un algorithme spécifique pour la résolution des systèmes linéaires creux. D'autres procédures de TSM exploitent les capacités de GAUSS. C'est notamment le cas des transformées en ondelettes et en paquets d'ondelettes. Les temps de calcul (en secondes) d'une transformée discrète en ondelettes sont présentés dans le tableau ci-dessous[19].

Nombre d'observations	16384	32768	65536	131072	262144
DAUB #4	0.55	1.37	3.08	6.48	13.23
DAUB #20	0.61	1.59	3.51	7.36	15.05

Un autre avantage de TSM est sa facilité d'emploi. L'apprentissage de TSM est d'autant plus facilité que 230 exemples illustrent l'utilisation des procédures. Voyons deux exemples simples d'utilisation de commandes de TSM. Pour estimer un modèle Local Level, nous avons

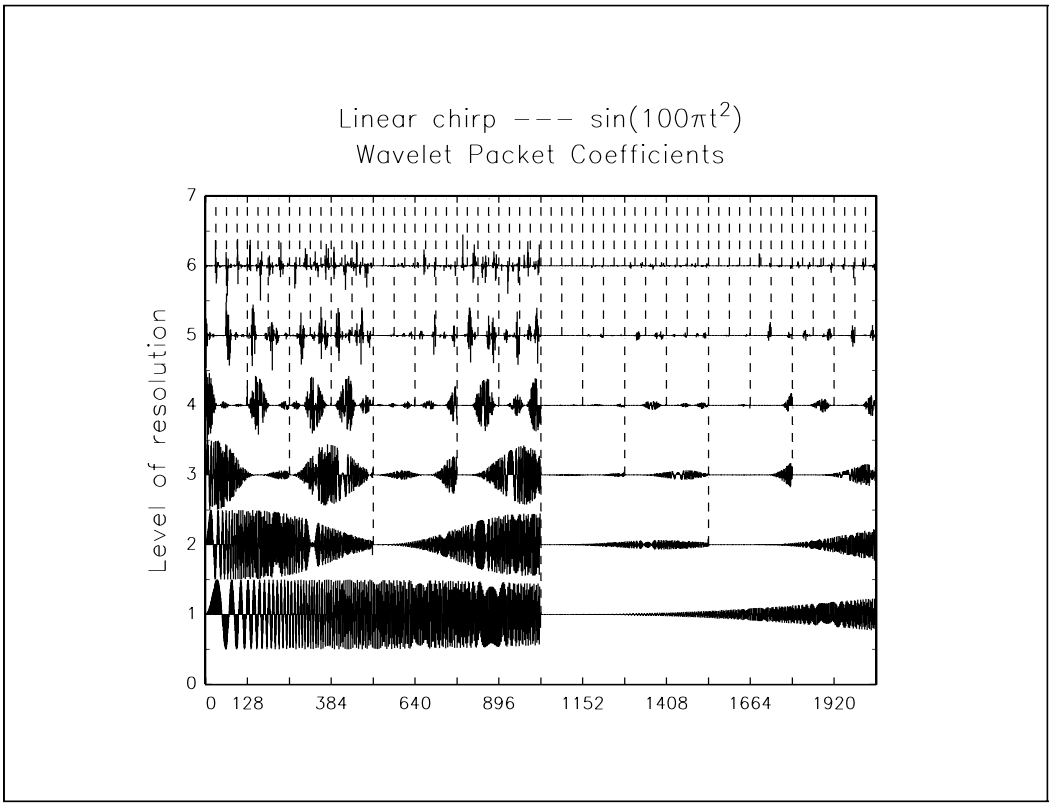
```
new;
library tsm,optmum;
TSMset;

load purse[] = purse.asc;

output file = purse.out reset;

sv = 1|1;
{beta,stderr,Mcov,Logl} =  sm_LL(purse,sv);

output off;
```

Voici le contenu du fichier *purse.out* :

```
Total observations:                                    71
Usable observations:                                   70
Number of parameters to be estimated:                   2
Degrees of freedom:                                    68
Value of the maximized log-likelihood function:  -227.49900


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------------
sig_eta         2.174190        0.713899         3.045513        0.003303
sig_epsi        5.037427        0.603238         8.350645        0.000000

Covariance matrix: inverse of the negative Hessian.
```

[18] Moins de 11 secondes suffisent pour estimer les cent mille coefficients FLS.

[19] A titre indicatif, avec 65536 observations et le filtre DAUB #4, il faut 54 secondes pour une transformation en paquet d'ondelettes, 64 secondes pour déterminer le meilleur niveau et 1 minutes 40 secondes pour définir la base optimale en utilisant l'algorithme *tree pruning* de Coifman et Wickerhauser.



Graphique 11

Le programme suivant permet de decomposer un chirp linéaire en paquets d'ondelettes et de représenter graphiquement la table des coefficients des six premiers niveaux.

```
new;
library tsm,pgraph;
TSMset;

Nobs = 2^11; N = Nobs;
t = seqa(0,2/Nobs,Nobs);
chirp = sin(100*pi*t^2);

{H,G,Htilde,Gtilde} = Coiflet(2); /* Filtre de Coiflet #2 */
pkt = wpkt(chirp,H,G,6);

graphset;
    call wpkPlot(pkt,1);
    _pnum = 2; _pdate = ""; _ptitlht = 0.20; _paxht = 0.20;
    fonts("simplex simgrma");
    title("Linear chirp --- sin(100\202p\201t[2])"\
          "\LWavelet Packet Coefficients");
    xtics(0,N-1,128,0);
    graphprt("-c=1 -cf=chirp.eps -w=5");
    draw;
```

2.2 Quelques exemples

2.2.1 Modèles espace état et Filtre de Kalman

TSM considère les modèles espace état de la forme[20]

$$\begin{cases} y_t = Z_t \alpha_t + d_t + \varepsilon_t & \forall\, t = 1, \ldots, Nobs \\ \alpha_t = T_t \alpha_{t-1} + c_t + R_t \eta_t & \forall\, t = 1, \ldots, Nobs \end{cases}$$

où y_t est un processus de dimension N et α_t un vecteur d'état de dimension m. De nombreuses procédures permettent d'estimer les paramètres du modèle (`KF_ml` définit la vraisemblance du processus d'innovation, `SSM_ic` calcule les conditions initiales selon la méthode de Jones), d'appliquer le filtre de Kalman (`KFiltering` et `KF_matrix`) ou encore d'obtenir un lissage (`KSmoothing`). La procédure `KF_gain` calcule les matrices de gain K_t, nécessaires à la représentation sous forme d'innovations

$$\begin{cases} y_t = Z_t E_{t-1}\left[\alpha_t\right] + d_t + \left(y_t - E_{t-1}\left[y_t\right]\right) \\ E_t\left[\alpha_{t+1}\right] = T_{t+1} E_{t-1}\left[\alpha_t\right] + c_{t+1} + K_t\left(y_t - E_{t-1}\left[y_t\right]\right) \end{cases}$$

Cela permet notamment d'étudier la convergence des anticipations rationnelles. L'algorithme du bootstrap développé par Stoffer et Wall [1991][21] correspond à la procédure `bootstrap_SSM`. Des procédures spécifiques permettent d'étudier les modèles invariants dans le temps (`KForecasting` pour la prévision, `ARE` pour la solution de l'équation algébrique de Riccati, `sgf_SSM` pour la fonction de densité spectrale, etc).

Considérons un premier exemple d'estimation par le filtre de Kalman d'un modèle invariant dans le temps. Dans son papier "The second bank of the United States in its early years: lessons form a transition towards an inter-regional monetary union", M-A. Sénégas s'intéresse au processus de convergence des taux d'inflation régionaux américains (New-York, Philadelphia, Charleston, New Orleans et Cincinnati) au début du 19ème siècle (1817-1480). Il utilise notamment une décomposition composante commune/composantes spécifiques :

$$\begin{bmatrix} \pi_1(t) \\ \pi_2(t) \\ \pi_3(t) \\ \pi_4(t) \\ \pi_5(t) \end{bmatrix} = \begin{bmatrix} \beta_1 & 1 & 0 & 0 & 0 & 0 \\ \beta_2 & 0 & 1 & 0 & 0 & 0 \\ \beta_3 & 0 & 0 & 1 & 0 & 0 \\ \beta_4 & 0 & 0 & 0 & 1 & 0 \\ \beta_5 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C(t) \\ \mu_1(t) \\ \mu_2(t) \\ \mu_3(t) \\ \mu_4(t) \\ \mu_5(t) \end{bmatrix}$$

avec $\pi_i(t)$ le taux d'inflation de la région i, $\mu_i(t)$ la composante spécifique de la région i et $C(t)$ la composante commune. β_i mesure l'influence de la composante commune sur le taux d'inflation de la région i. Il suppose de plus que les composantes spécifiques et la composante commune suivent une marche aléatoire. Le programme estime les paramètres du modèle espace état associé, ainsi que la composante inobservable $C(t)$.

```
new;
library tsm,optmum,pgraph;
TSMset;

load y = pigen;

y = packr(y);

d = zeros(5,1);
H = zeros(5,5);
T = eye(6);
c = zeros(6,1);
R = eye(6);

a0 = 0|y[1,.]';
P0 = zeros(6,6);

/* Compute the log-likelihood in the time domain */
```

[20] Harvey, A.C. [1990], Forecasting, Structural Time Series Models and the Kalman Filter, Cambridge University Press, Cambridge

[21] Stoffer, D.S. et K.D. Wall [1991], Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter, *Journal of the American Statistical Association*, **86**, 1024-1033

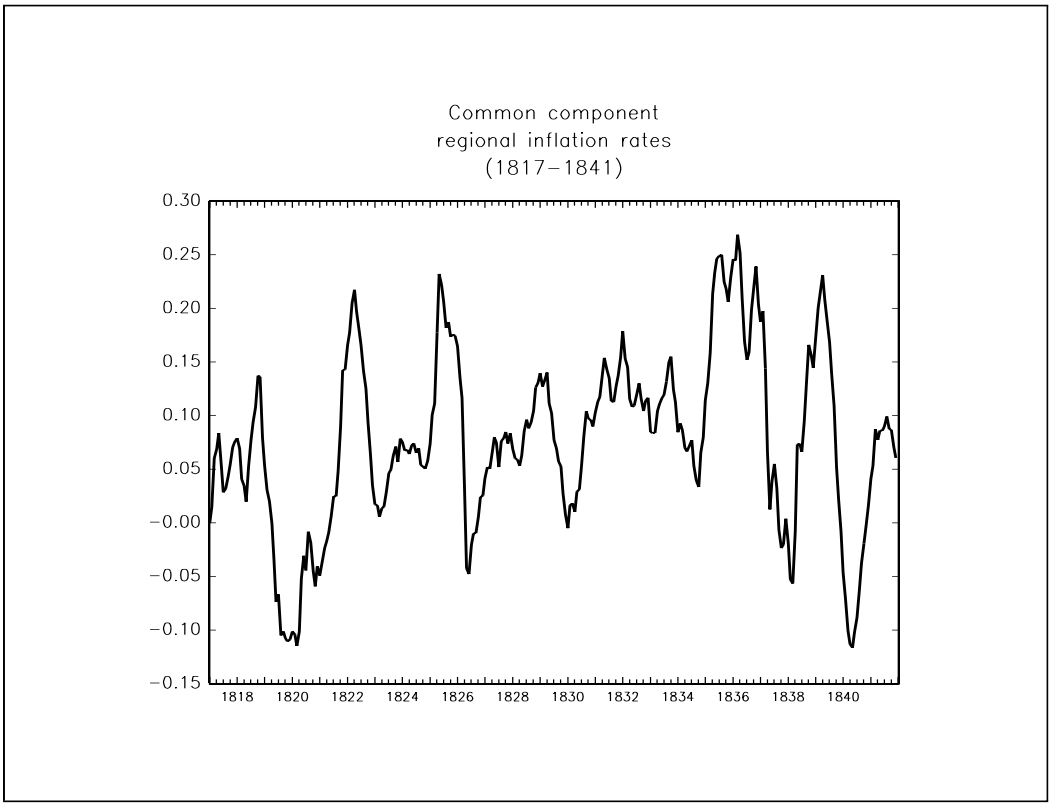
```
proc ml(theta);
  local beta,sigma,Z,Q,logl;

  beta = theta[1:5];
  sigma = theta[6:11]^2;

  Z = beta~eye(5);
  Q = diagrv(zeros(6,6),sigma);

  call SSM_build(Z,d,H,T,c,R,Q,0); /* Build the state space model */
  call KFiltering(y,a0,P0);        /* Run the Kalman filter       */
  logl = KF_ml;              /* Compute the log-likelihood vector */

  retp(logl);
endp;


/* Estimation of the parameters vector */

sv = ones(11,1);
_tsm_Mcov = 1;
{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

beta = theta[1:5];
sigma = theta[6:11]^2;

Z = beta~eye(5);
Q = diagrv(zeros(6,6),sigma);

call SSM_build(Z,d,H,T,c,R,Q,0); /* Build the state space model */
call KFiltering(y,a0,P0);        /* Run the Kalman filter       */
at = KF_matrix(3);               /* Read the state vectors a(t) */

graphset;
  _pdate = ""; _pnum = 2;
  lab=" [1817] [1818] [1819] [1820] [1821] "\
      " [1822] [1823] [1824] [1825] [1826] [1827] [1828] "\
      " [1829] [1830] [1831] [1832] [1833] [1834] [1835] "\
      " [1836] [1837] [1838] [1839] [1840] [1841] ";
  asclabel(lab,0);
  xtics(1,300,12,4);
  title("Common component"\
        "\Lregional inflation rates"\
        "\L(1817-1841)");
  _pltype = 6;
  _plwidth = 5;
  graphprt("-c=1 -cf=common.eps -w=5");
  xy(seqa(1,1,rows(at)),at[.,1]);
```

Considérons maintenant un modèle espace-état variant dans le temps (c'est-à-dire que les matrices définissant le modèle ne sont plus constantes) :

$$\begin{cases} y_t = x_t \beta(t) + u_t \\ \beta(t) = \alpha\beta(t-1) + \delta z_t + v_t \end{cases}$$

Dans l'exemple suivant, nous avons simulé un processus (avec $\alpha = 0.8$, $\delta = 0.25$, $u_t \sim \mathcal{N}(0,2)$ et $v_t \sim \mathcal{N}(0,1.5)$). Nous avons ensuite estimé les paramètres du modèle $(\alpha, \delta, \sigma_u, \sigma_v)$ et la trajectoire du coefficient $\beta(t)$.

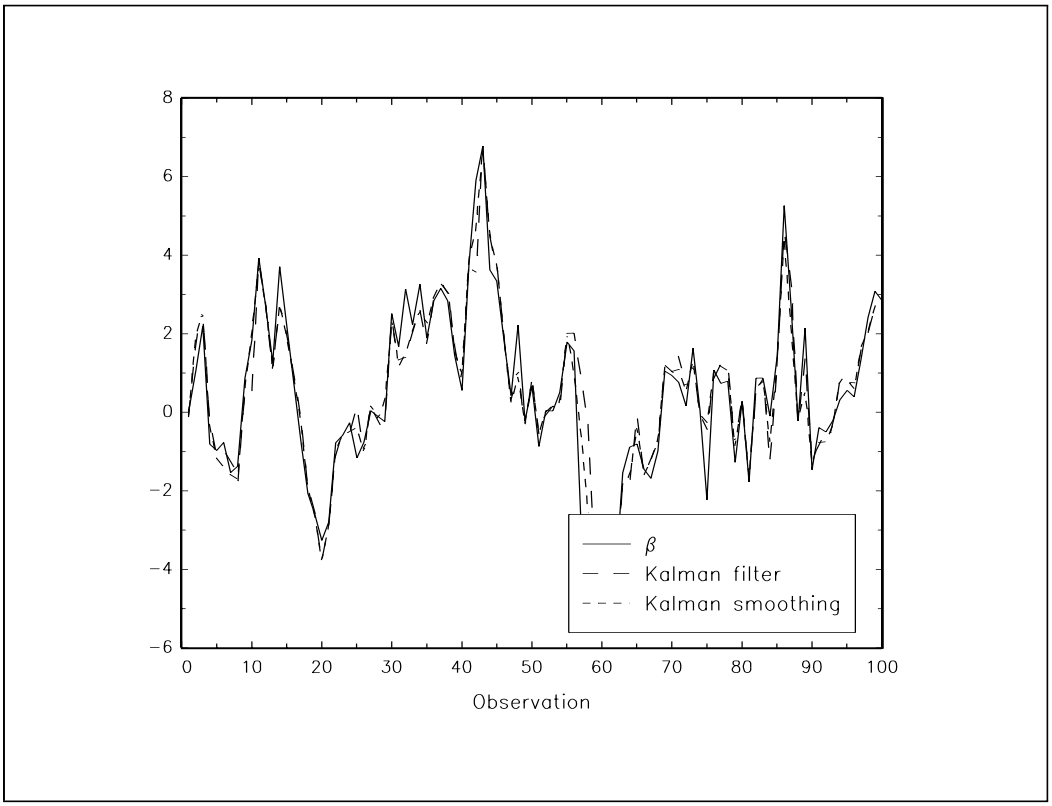
```
/*
**  y(t) = b(t)*x(t) + u(t)
**  b(t) = alpha*b(t-1) + delta*z(t) + v(t)
*/
```



Graphique 12

```
new;
library tsm,optmum,pgraph;
declare external sigma;
declare external delta;
declare external alpha;

rndseed 123456;

Nobs = 100;
s = seqa(1,1,100);
x = rndu(Nobs,1)*10;
z = rndu(Nobs,1);
v = rndn(Nobs,1)*1.5;
b = recserar(v+0.25*z,0,0.8);
u = rndn(Nobs,1)*2;
y = b.*x + u;

proc ZZ(i);
  retp(x[i]);
endp;

proc d(i);
  retp(0);
endp;

proc T(i);
  retp(alpha);
endp;
```

```
proc c(i);
  retp(delta*z[i]);
endp;

proc R(i);
  retp(1);
endp;

proc H(i);
  local w;
  w = sigma[1]^2;
  retp(w);
endp;

proc Q(i);
  local w;
  w = sigma[2]^2;
  retp(w);
endp;

a0 = 0; P0 = 0;

proc ml(theta);
  local LogL;
  alpha = theta[1];
  delta = theta[2];
  sigma = theta[3 4];
  call SSM_build(&ZZ,&d,&H,&T,&c,&R,&Q,1);
  call KFiltering(y,a0,P0);
  LogL = KF_ml;
  retp(LogL);
endp;

output file = varying.out reset;

sv = ones(4,1);
_tsm_parnm = "alpha"|"delta"|"sig_u"|"sig_v";
{theta,stderr,Mcov,LogL} = TD_ml(&ml,sv);

output off;

alpha = theta[1]; delta = theta[2]; sigma = theta[3 4];
call SSM_build(&ZZ,&d,&H,&T,&c,&R,&Q,1);
call KFiltering(Y,a0,P0);

yc = KF_matrix(1);     /* y[t|t-1]      */
v = KF_matrix(2);      /* v[t]          */
a = KF_matrix(3);      /* a[t]          */
P = KF_matrix(4);      /* P[t]          */
ac = KF_matrix(5);     /* a[t|t-1]      */
Pc = KF_matrix(6);     /* P[t|t-1]      */
F = KF_matrix(7);      /* F[t|t-1]      */
invF = KF_matrix(8);   /* F[t|t-1]^(-1) */

{as,Ps} = Ksmoothing;

graphset;
  _pdate = ""; _pnum = 2; fonts("simplex simgrma");
  xlabel("Observation");
  _plegstr = "\202b\201\000Kalman filter\000Kalman smoothing";
```



Graphique 13

```
_plegctl = {2 6 5 1};
_pltype = 6|1|3;
graphprt("-c=1 -cf=varying.eps -w=5");
xy(s,b~a~as);
```

```
Total observations:                                    100
Usable observations:                                   100
Number of parameters to be estimated:                    4
Degrees of freedom:                                     96
Value of the maximized log-likelihood function:    -325.15847
```

Parameters	estimates	std.err.	t-statistic	p-value
alpha	0.730533	0.078219	9.339549	0.000000
delta	0.270691	0.260028	1.041004	0.300488
sig_u	1.975484	0.400561	4.931796	0.000003
sig_v	1.434785	0.142434	10.073341	0.000000

```
Covariance matrix: inverse of the negative Hessian.
```

2.2.2 Analyse dans le domaine des fréquences

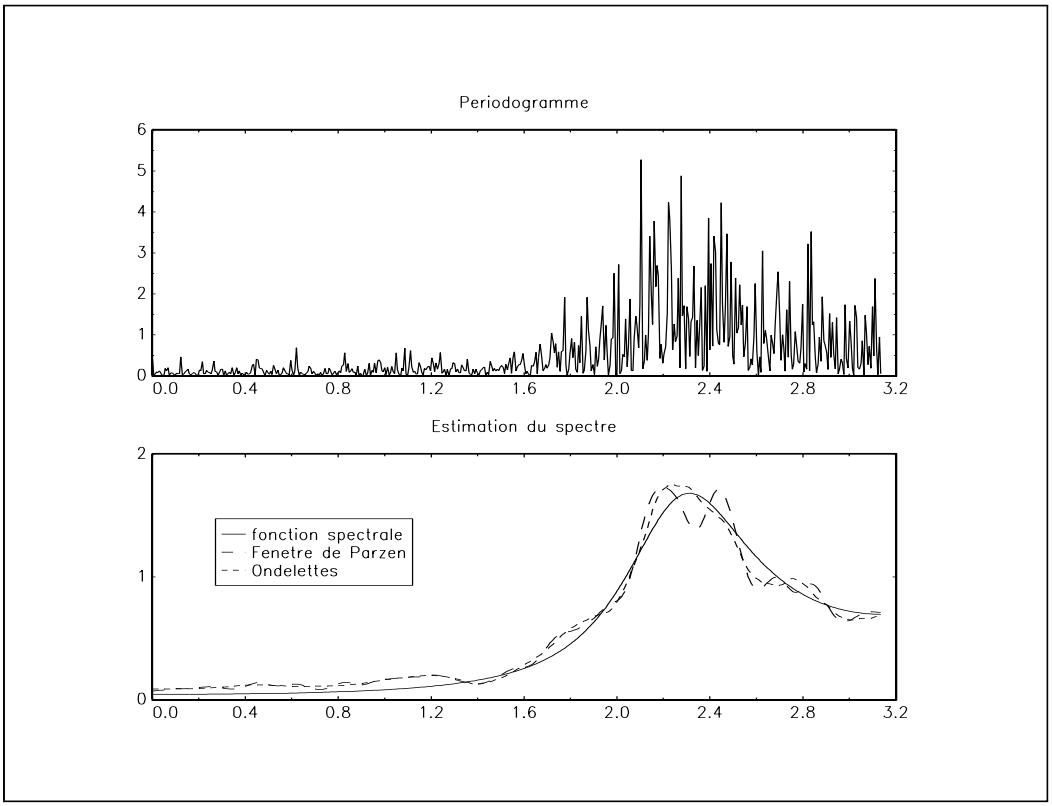
Le domaine des fréquences est très developpé dans TSM. Il existe des procédures pour le calcul du périodogramme, l'estimation du spectre par lissage (fenêtres rectangulaire, de Bartlett, Daniell, Tukey, Parzen et Bartlett-Priestley), pour l'analyse croisée et l'analyse multidimensionnelle. Dans l'exemple suivant, nous simulons un modèle arma univarié, puis nous calculons le périodogramme. Nous estimons ensuite le spectre par lissage spectral et par ondelettes[22].

[22] Moulin, P. [1994], wavelet thresholding techniques for power spectrum estimation, *IEEE transactions on Signal Processing*, **42**, 3126-3136

```
new;
library tsm,optmum,pgraph;
TSMset;

rndseed 123456;

t = seqa(1,1,1024);
beta = -0.9|-0.5;
sigma = 0.5;
x = recserar(rndn(1024,1)*sigma,0|0,beta);
x = x + rndn(1024,1)*0.25;

_fourier = 0;
{lambda,I} = PDGM(x);

_smoothing = 25|5|0|0.23;   /* Parzen lag window with bandwidth = 25 */
I2 = smoothing(I);
sgf = sgf_arfima(beta|0,2,0,sigma,lambda);

{H,G,Htilde,Gtilde} = Daubechies(8);
_wcenter = 0;

w = wt(I,H,G,0);
w = extract(w,11|10|9|8|7|6);
y = iwt(w,Htilde,Gtilde,0);

q = trunc(rows(lambda)/2);

graphset;
  _pdate = ""; _pnum = 2; _pnumht = 0.25; _ptitlht = 0.25;
  _pltype = 6|1|3|5;

  begwind;
  window(2,1,0);
  setwind(1);
    title("Periodogramme");
    xy(lambda[1:q],I[1:q]);
  nextwind;
    title("Estimation du spectre");
    ytics(0,2,1,0);
    _plegstr = "fonction spectrale\0Fenetre de Parzen\0Ondelettes";
    _plegctl = {2 5 1 3};
    xy(lambda[1:q],sgf[1:q]~I2[1:q]~y[1:q]);
    graphprt("-c=1 -cf=spectral.eps -w=5");
  endwind;
```

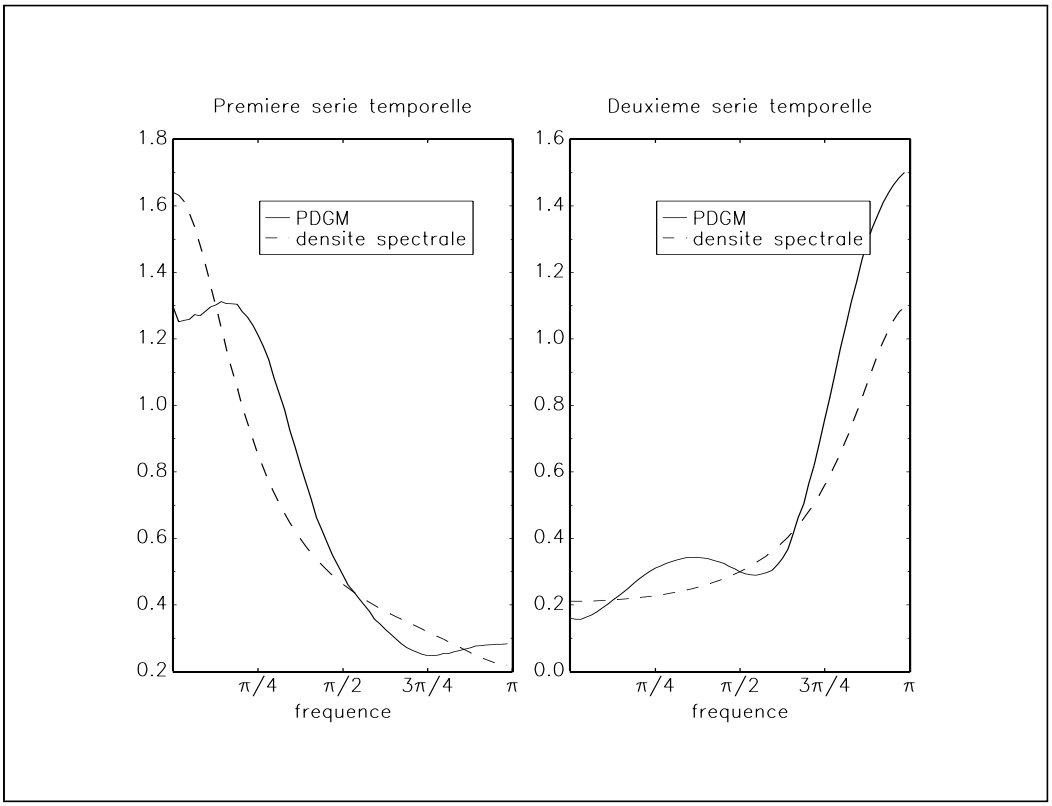
Dans l'exemple suivant, nous simulons un modèle espace état (un système à 2 entrées et 2 sorties). Nous calculons ensuite le périodogramme multidimensionnel des séries simulées et nous le comparons au spectre théorique du modèle espace état.

```
new;
library tsm,optmum,pgraph;
TSMset;

rndseed 123456;

_fourier = 0;

Z = eye(2); d = 0|0;
let H[2,2] = 0.2 0 0 0.1;
let T[2,2] = 0.5 0.3 0 -0.5;
```



Graphique 14

```
c = 0|0; R = 1|1; Q = 0.25;

call SSM_build(Z,d,H,T,c,R,Q,0);
{y,a} = RND_SSM(0|0,100);

{lambda,I} = PDGM2(y);

_smoothing = 15|4|1|0.23;
I1 = smoothing(I[.,1]);
I2 = smoothing(I[.,4]);

G = sgf_SSM(lambda);

q = trunc(rows(lambda)/2);

graphset;
  _pdate = ""; _pnum = 2;
  _ptitlht = 0.30; _pnumht = 0.30; _paxht = 0.30;
  fonts("simplex simgrma");
  begwind;
  window(1,2,0);
  setwind(1);
    title("Premiere serie temporelle");
    xlabel("frequence");
    xtics(0,pi,pi/4,0);
    lab = " 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201";
    asclabel(lab,0);
    _plegstr = "PDGM\0densite spectrale"; _plegctl = {2 5 3 5};
    xy(lambda[1:q],I1[1:q]~G[1:q,1]);
  nextwind;
```



Graphique 15

```
    title("Deuxieme serie temporelle");
    graphprt("-c=1 -cf=pdgm.eps -w=5");
    xy(lambda[1:q],I2[1:q]~G[1:q,4]);
  endwind;
```

La procédure `sgf_SSM` permet de calculer le spectre d'un modèle espace état multidimensionnel. Cela permet notamment de pouvoir estimer les modèles multivariés dans le domaine des fréquences.

```
new;
library tsm,optmum,pgraph;
TSMset;

rndseed 123456;

_fourier = 0;

Z = eye(2); d = 0|0;
let H[2,2] = 0.2 0 0 0.1;
let T[2,2] = 0.5 0.3 0 -0.5;
c = 0|0; R = 1|1; Q = 0.25;

call SSM_build(Z,d,H,T,c,R,Q,0);
{y,a} = RND_SSM(0|0,128);

{lambda,Iy} = PDGM2(y);

sv = 0.5|0.3|-0.5|sqrt(0.25|0.2|0.1);

/*
** Procedure pour calculer la densite spectrale
** du modele espace etat
```

```
*/

proc sgf(theta,lambda);
  local T,Q,H,Z,d,c,R;
  local g;

  T = (theta[1]~theta[2])|(0~theta[3]);
  Q = theta[4];
  Q = Q^2;
  H = (theta[5]~0)|(0~theta[6]);
  H = H^2;
  Z = eye(2); d = 0|0;
  c = {0,0}; R = {1,1};

  call SSM_build(Z,d,H,T,c,R,Q,0);
  g = sgf_SSM(lambda);
  retp(g);
endp;


/*
** Vraisemblance d'un processus multidimensionnel
*/

proc ml(theta);
  local Gy,Nstar,logl,j,Ij,Gj;

  Gy = sgf(theta,lambda);
  Nstar = rows(lambda);

  logl = zeros(Nstar,1);

  j = 1;
  do until j > Nstar;
   Ij = xpnd2(Iy,j);
   Gj = xpnd2(Gy,j);
   logl[j] = -0.5*ln(det(Gj)) -0.5*sumc(diag((inv(Gj)*Ij)));
   j = j + 1;
  endo;

  logl = real(logl);

  retp(logl);
endp;

output file = fdml.out reset;

_tsm_Mcov = 1;
{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;
```

Voici le contenu de *fdml.out* :

```
Total observations:                                   128
Usable observations:                                  128
Number of parameters to be estimated:                   6
Degrees of freedom:                                   122
Value of the maximized log-likelihood function:   5.70812
```

```
Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01             0.572158        0.075523         7.575998        0.000000
P02             0.187763        0.129585         1.448954        0.149916
P03            -0.568451        0.083228        -6.830014        0.000000
P04             0.507772        0.040336        12.588464        0.000000
P05             0.454139        0.040628        11.177860        0.000000
P06             0.294961        0.035467         8.316431        0.000000

Covariance matrix: inverse of the negative Hessian.
```

2.2.3 Les différentes méthodes d'estimation

Les différentes méthodes d'estimation de TSM sont le maximum de vraisemblance dans le domaine du temps, le maximum de vraisemblance dans le domaine des fréquences (ou méthode de Whittle) et la méthode des moments généralisés. Pour chacune de ces méthodes, nous pouvons imposer des restrictions linéaires implicites. Pour les restrictions linéaires explicites, nous pouvons employer la procédure `Explicit_to_Implicit` pour les transformer en restrictions implicites. L'exemple suivant concerne l'estimation d'un modèle *Local Linear Trend*. Nous utilisons les deux méthodes du maximum de vraisemblance.

```
new;
library tsm,optmum,pgraph;
TSMset;

load gnp[] = gnp.asc;
Nobs = rows(gnp);

output file = llt.out reset;

/* Methode de Whittle */

sv = 5|5|5;

print "Methode de Whittle";
print "------------------";

_tsm_optmum = 0;   /* Algorithme du score */
{beta,stderr,Mcov,Logl} =  sm_LLT(gnp,sv);

/* Methode du maximum de vraisemblance dans le domaine du temps */

print; print;
print "Methode du maximum de vraisemblance dans le domaine du temps";
print "-----------------------------------------------------------";


Z = 1~0; d = 0;
T = {1 1,0 1}; c = 0|0; R = {1 0,0 1};
a0 = gnp[1]|0; P0 = zeros(2,2);

proc ml(theta);
  local H,Q,LogL;
  theta = theta^2;  /* sigma^2 */
  H = theta[3];
  Q = (theta[1]~0)|(0~theta[2]);
  call SSM_build(Z,d,H,T,c,R,Q,0);
  call KFiltering(gnp,a0,P0);
  LogL = KF_ml;
  retp(LogL);
endp;
```

```
_tsm_optmum = 0;    /* algorithme BHHH */
_tsm_Mcov = 3;      /* Heteroskedasticity covariance */

{beta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;

Methode de Whittle
------------------

Total observations:                                      61
Usable observations:                                     59
Number of parameters to be estimated:                     3
Degrees of freedom:                                      56
Value of the maximized log-likelihood function:      -273.00725


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
sig_eta        21.732459        3.862894         5.625953        0.000001
sig_zeta        1.601452        0.825478         1.940030        0.057418
sig_epsi        7.112489        5.283081         1.346277        0.183638

Covariance matrix: inverse of the negative Hessian.


Methode du maximum de vraisemblance dans le domaine du temps
------------------------------------------------------------

Total observations:                                      61
Usable observations:                                     61
Number of parameters to be estimated:                     3
Degrees of freedom:                                      58
Value of the maximized log-likelihood function:      -281.05421


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
sig_eta        20.935297        3.821497         5.478297        0.000001
sig_zeta        1.865094        0.598828         3.114575        0.002861
sig_epsi        7.544870       10.754893         0.701529        0.485778

Covariance matrix: White Heteroskedastic matrix.
```

Nous pouvons estimer un modèle ARCH par maximum de vraisemblance ou par méthode des moments généralisés.

```
new;
library tsm,optmum;

Nobs = 250;
alpha0 = 0.5; alpha1 = 0.6;

/* Simulation d'un processus ARCH */

u = zeros(Nobs,1);
h = zeros(Nobs,1);
i = 2;
do until i > Nobs;
  h[i] = sqrt(alpha0^2 + alpha1^2*u[i-1]^2);
  u[i] = rndn(1,1)*h[i];
```

```
  i = i+1;
endo;

x =  3*rndu(Nobs,1);
y = 2 + 3*x + u;

proc ml(theta);
  local u,u2,h2,logl;

  u = y-theta[1]-theta[2]*x;

  u2 = u.*u;
  h2 = theta[3]^2 + theta[4]^2*lag1(u2);
  h2[1] = theta[3]^2;

  logl = -0.5*ln(2*pi)-0.5*ln(h2)-0.5*u2./h2;

  retp(logl);
endp;

proc ARCH(theta);
  local M,u,h2,u2,i;

  M = zeros(Nobs,4);

  u = y-theta[1]-theta[2]*x;

  /* Premier moment */

  M[.,1] = u;

  /* calcul h(t)^2 */

  u2 = u.*u;
  h2 = theta[3]^2 + theta[4]^2*lag1(u2);
  h2[1] = theta[3]^2;

  /* Deuxieme moment */

  M[.,2] = u2-h2;

  /* u(t) non correle avec x(t) */

  M[.,3] = x.*u;

  /* u(t)^2-h(t)^2 non correle avec u(t-i)^2, i = 1 */

  M[.,4] = (u2-h2).*lagn(u2,1);

  retp(M);
endp;

output file = arch.out reset;

sv = 2|3|0.5;

/* theta[4] = 0, Pas d'effet arch */

RR = design(1|2|3|0); r = zeros(4,1);
_tsm_parnm = "beta1"|"beta2"|"alpha0"|"alpha1";
```

```
/* Methode du maximum de vraisemblance */

print "Methode du maximum de vraisemblance";
print "-----------------------------------";

{theta1,stderr,Mcov,Qmin} = TD_cml(&ml,sv,RR,r);

_gmm_RR = RR;
_gmm_r = r;

/* Methode des moments generalises */

print; print;
print "Methode du moments generalises";
print "------------------------------";

{theta2,stderr,Mcov,Qmin} = gmm(&arch,sv);

output off;
```

```
Methode du maximum de vraisemblance
-----------------------------------

Total observations:                                    250
Usable observations:                                   250
Number of parameters to be estimated:                    3
Degrees of freedom:                                    247
Value of the maximized log-likelihood function:  -234.57747


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
beta1           1.922025        0.075797        25.357392        0.000000
beta2           3.021310        0.045081        67.019108        0.000000
alpha0          0.618396        0.027656        22.360410        0.000000
alpha1          0.000000        0.000000                .               .

Covariance matrix: inverse of the negative Hessian.


Methode du moments generalises
------------------------------

Total observations:                                    250
Usable observations:                                   249
Number of parameters to be estimated:                    4
Degrees of freedom:                                    246
Number of moment conditions:                             4
Value of the criterion function:                   0.02446


Hansen's test of overidentifying restrictions:     6.09049
p-value:                                           0.01359


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
beta1           1.932011        0.076772        25.165430        0.000000
beta2           3.018102        0.044413        67.955276        0.000000
alpha0          0.582556        0.028398        20.514202        0.000000
```

```
alpha1          0.000000        0.000000                .               .
```

2.2.4 Systèmes approximativement linéaires

Les algorithmes FLS[23] et GFLS[24] sont implémentés dans TSM. Ils permettent d'estimer des systèmes approximativement linéaires. Considérons un modèle linéaire avec des coefficients variant dans le temps. Dans l'exemple suivant, le premier coefficient est une constante, le second est une fonction sinusoïdale bruitée et le troisième suit un processus AR(1). Nous avons simulé une série, puis nous avons estimé les coefficients par OLS et FLS. Le graphique (16) représente la vraie valeur du troisième coefficient et les estimateurs OLS et FLS. La méthode FLS apparaît très performante.

```
new;
library tsm,pgraph;
TSMset;

rndseed 123;
Nobs = 500;
x = floor(rndu(Nobs,3)*100);   /* 3 entrees */
b1 = ones(Nobs,1);
b2 = sin(seqa(0,2*pi/Nobs,Nobs)) + rndn(Nobs,1)*0.25;
b3 = recserar(rndn(Nobs,1),10,0.9);
B = b1~b2~b3;
y = sumc(x'.*B') + rndn(Nobs,1);   /* variable expliquee */

{Bfls,u,r2M,r2D} = FLS(y,x,1);            /* Estimation FLS */
Bols = invpd(x'x)*x'y;                    /* Estimation OLS */

graphset;
  _pltype = 6|3|1; _pdate = ""; _pnum = 2;
  title("3[eme] coefficient"); xlabel("Observation");
  _plegstr = "vraie valeur\0FLS\0OLS"; _plegctl = 1;
  Bols = Bols'.*ones(Nobs,1);
  graphprt("-c=1 -cf=fls1.eps -w=5");
  xy(seqa(1,1,Nobs),b3~Bfls[.,3]~Bols[.,3]);
```

Considérons maintenant le modèle suivant

$$y_t = x_t\beta + u_t$$

avec

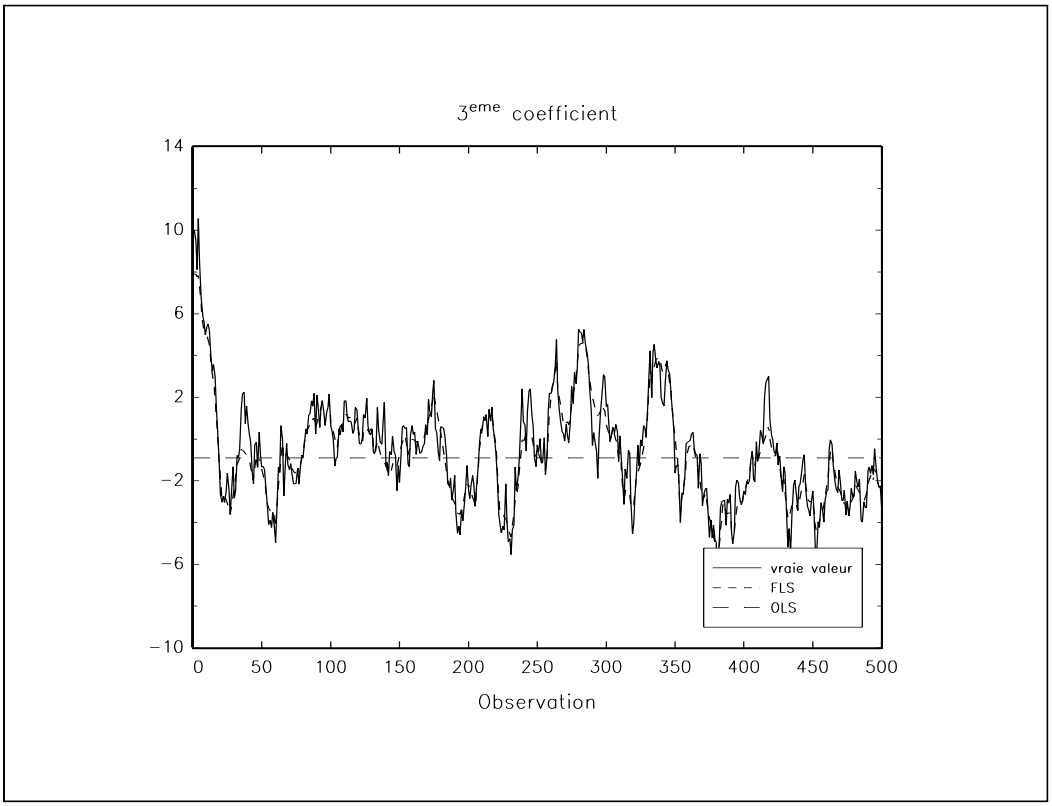
$$\beta_t = \begin{cases} z & \text{si } t \leq S \\ w & \text{si } t > S \end{cases}$$

Le graphique 17 montre que le filtre FLS peut détecter le déplacement non anticipé de z à w.

```
new;
library tsm,pgraph;
TSMset;

rndseed 123;

Nobs = 500; S = 250;
z = 2; w = 5;

x = floor(rndu(Nobs,1)*100);
b = ones(S,1)*z|ones(Nobs-S,1)*w;
y = 3 + x.*b + rndn(Nobs,1);
x = ones(Nobs,1)~x;

_print = 0;
```

[23] Kalaba, R. et L. Tesfatsion [1989], Time-varying linear regression via flexible least squares, *Computers & Mathematics with Applications*, **17**, 1215-1245

[24] Kalaba, R. et L. Tesfatsion [1990], Flexible least squares for approximately linear systems, *IEEE Transactions on Systems, Man and Cybernetics*, **20**, 978-989

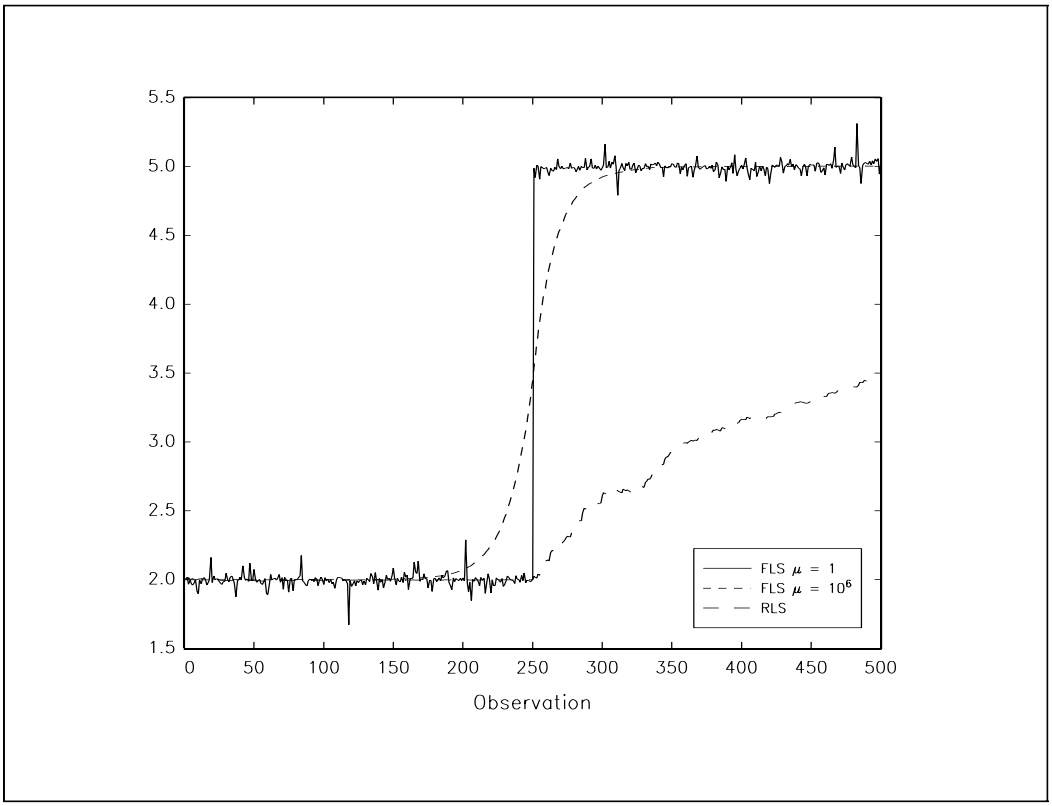


Graphique 16

```
{Bfls1,u,r2M,r2D} = FLS(y,x,1);
{Bfls2,u,r2M,r2D} = FLS(y,x,10^6);
{Brls,stderr,u,rss} = RLS(y,x);

graphset;
  _pltype = 6|3|1;
  fonts("simplex simgrma");
  _plegstr = "FLS \202m\201 = 1\0FLS \202m\201 = 10[6]\0RLS";
  _plegctl = 1;
  _pdate = ""; _pnum = 2; xlabel("Observation");
  graphprt("-c=1 -cf=fls2.eps -w=5");
  xy(seqa(1,1,Nobs),Bfls1[.,2]~Bfls2[.,2]~Brls[.,2]);
```

Considérons un système linéaire 4 entrées/1 sortie. Supposons que ce système comprenne 8 régimes. Dans le programme suivant, les régimes sont liés à la relation $2^{\text{ème}}$ entrée/$1^{\text{ère}}$ sortie. FLS détecte parfaitement ces régimes (voir graphiques 18 et 19).

```
new;
library tsm,pgraph;
TSMset;

rndseed 123;

Nobs = 1000;
z = 2|2.5|2.25|1.5|2|2.5|3|2.75;

x1 = floor(rndu(Nobs,1)*100);
x2 = floor(rndu(Nobs,2)*100);
b1 = {}; b2 = 2|3;
i = 1;
do until i > 8;
```

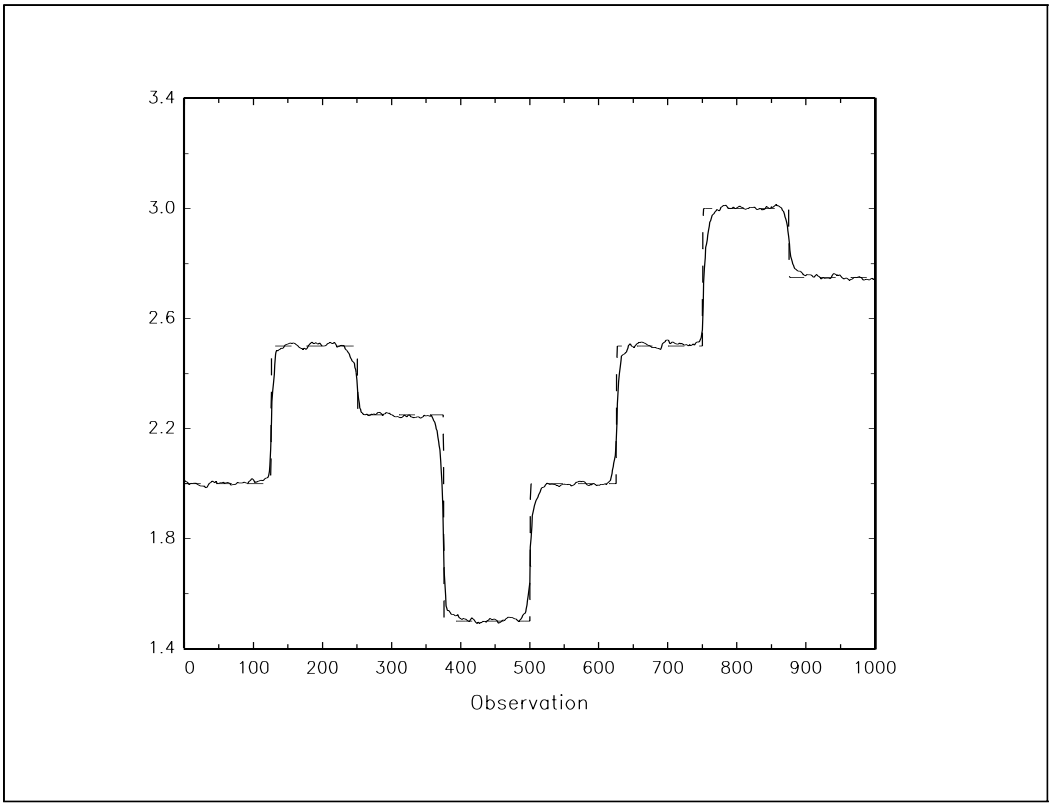


Graphique 17

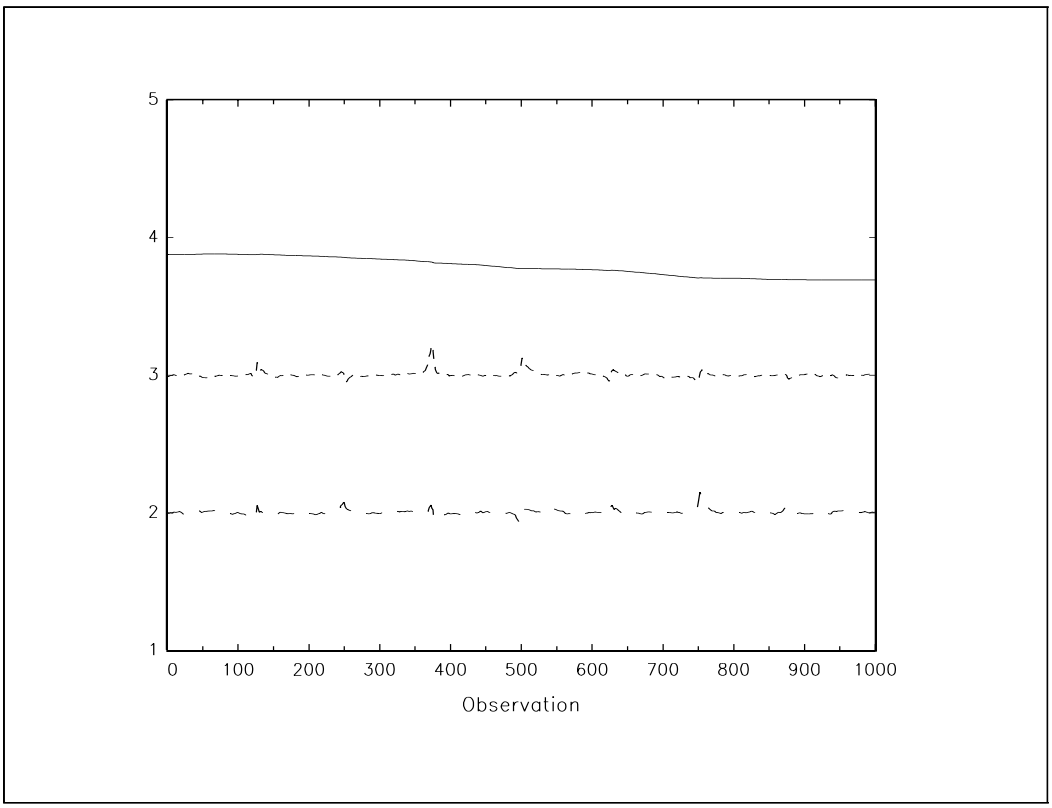
```
  b1 = b1 | z[i]*ones(125,1);
  i = i + 1;
endo;

y = 4 + x1.*b1 + x2*b2 + rndn(Nobs,1);
x = ones(Nobs,1)~x1~x2;

{Bfls,u,r2M,r2D} = FLS(y,x,10000);

graphset;
  _pdate = ""; _pnum = 2; xlabel("Observation");
  graphprt("-c=1 -cf=fls3a.eps -w=5");
  xy(seqa(1,1,Nobs),Bfls[.,2]~b1);
  graphprt("-c=1 -cf=fls3b.eps -w=5");
  ytics(1,5,1,0);
  _pltype = 6|1|3;
  xy(seqa(1,1,Nobs),Bfls[.,1 3 4]);
```

L'algorithme GFLS est une généralisation de FLS lorsque le système comprend plusieurs sorties. Dans l'exemple suivant, nous employons le filtre GFLS pour extraire la tendance d'une série.

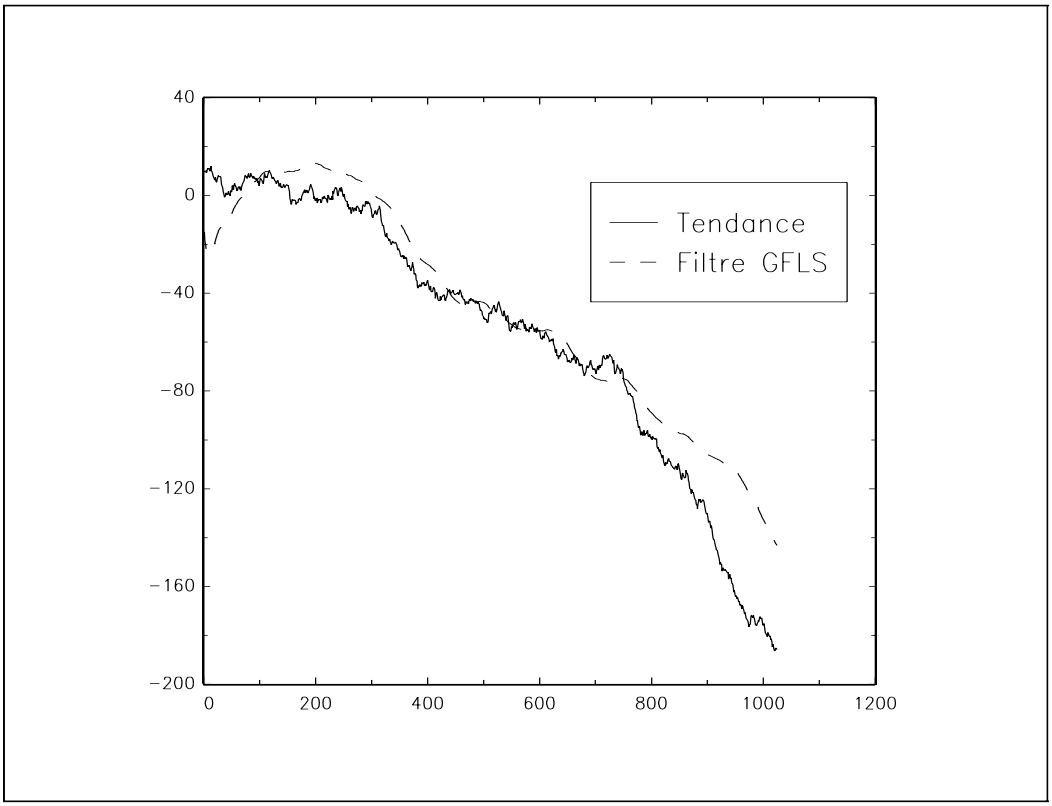
```
new;
library tsm,optmum,pgraph;
TSMset;

load data[1024,2] = tendance.asc;

y = data[.,1];
tendance = data[.,2];

/*
```



Graphique 18



Graphique 19



Graphique 20

```
** Filtre GFLS
*/

proc H(t); retp(1); endp;
proc m(t); retp(0); endp;
proc MM(t); retp(1); endp;
proc F(t); retp(1); endp;
proc d(t); retp(0); endp;
proc DD(t); retp(1); endp;

Q0 = 1; p0 = y[1];
{a_GFLS,u,as,us} = gfls(y,&H,&m,&MM,&F,&d,&DD,Q0,p0,10000);

graphset;
   _pdate = "";
  _plegstr = "Tendance\0Filtre GFLS";
  _plegctl = {2 8 5.25 4.5};
  graphprt("-c=1 -cf=tendance.eps -w=5");
  xy(seqa(1,1,1024),tendance~a_GFLS);
```

2.2.5 Exposant de Hurst et racine fractionnaire

Le programme suivant est une comparaison sur quelques simulations des estimateurs de Hurst dans le domaine du temps[25] et dans le domaine des fréquences[26].

```
new;
library tsm,optmum;
TSMset;
```

[25] Lo, A.W. [1991], Long-memory in stock market prices, *Econometrica*, **59**, 1279-1313

[26] Robinson, P.M. [1995], Gaussian semiparametric estimation of long range dependence, *The Annals of Statistics*, **23**, 1630-1661

```
#include robinson.src;
#include hurst.src;

output file = hurst.out reset;

_fourier = 0;
_print = 0;
__output = 0;

i = 1;
do until i > 10;
  x = rndn(1000,1);
  H1 = Robinson(x,0,1,0);
  {H2,RS} = HURST(x,2,5);
  print H1~H2;
  i = i + 1;
endo;

output off;
```

Voici le contenu du fichier *hurst.out* :

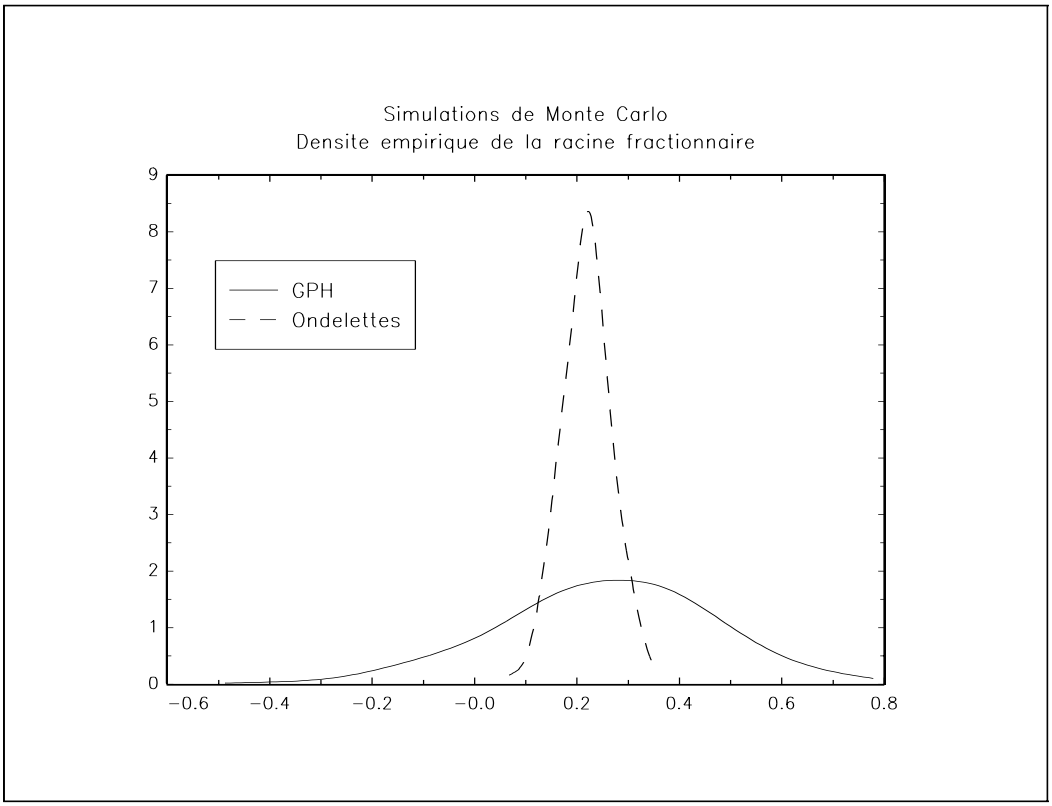
```
      0.52351927       0.59680076
      0.51591368       0.40370499
      0.53133504       0.49360975
      0.49289042       0.34145124
      0.48771991       0.76600895
      0.50909870       0.57185275
      0.47975597       0.48701289
      0.50062348       0.46374857
      0.48829980       0.44433679
      0.52284438       0.45157564
```

Plusieurs méthodes existent pour estimer la racine fractionnaire. Dans l'exemple qui suit, nous comparons l'estimateur GPH avec celui des ondelettes[27].

```
new;
library tsm,optmum,pgraph;
#include fractal.src;
TSMset;

M = 8;
Nobs = 2^8;
Nr = 500;    /* Nombre de replications */

d = 0.25;
{data,retcode} = RND_arfima(d,0,0,2,1000,Nobs,Nr);

Estimate = zeros(Nr,2);

Nbsimul = 1;
do until Nbsimul > Nr;
  locate 5,5; print ftos(Nbsimul,"Iteration n %lf",2,0);
  x = data[.,Nbsimul];
  d1 = GPH(x);
  d2 = WaveletMethod(x);
  Estimate[Nbsimul,.] = d1~d2;
  Nbsimul = Nbsimul + 1;
```

[27] WORNELL, G.W. et A. OPPENHEIM [1992], Estimation of fractal signals form noisy measurements using wavelets, *IEEE transactions on Signal Processing*, **40**, 611-623

```
endo;


x = {}; dens ={};

j = 1;
do until j > 2;
  {x_,dens_,F,retcode} = Kernel(estimate[.,j]);
  x = x~x_; dens = dens~dens_;
  j = j + 1;
endo;

graphset;
   title("Simulations de Monte Carlo"\
         "\LDensite empirique de la racine fractionnaire");
  _pdate = ""; _pnum = 2;
  _plegstr = "GPH\0Ondelettes";
  _plegctl = {2 6 1 4};
  _pltype = 6|1;
  graphprt("-c=1 -cf=fraction.eps -w=5");
  xy(x,dens);

/*
** Procedure pour la methode GPH
*/

proc (1) = gph(x);
  local lambda,I,Nobs,M,Y,w,XX,beta,d;

  _fourier = 0;
  {lambda,I} = PDGM(x);

  Nobs = rows(I);
  M = trunc(sqrt(Nobs));

  Y = ln(I[2:M+1]);
  w = 4*sin(lambda[2:M+1]/2)^2;
  XX = ones(M,1)~-ln(w);

  beta = Y/XX;
  d = beta[2];

  retp(d);
endp;


/*
** Procedure pour la methode des ondelettes
*/

proc WaveletMethod(x);
  local H,G,Htilde,Gtilde,w;
  local sv,theta,stderr,Mcov,Logl;

  M = ln(rows(x))/ln(2);

  {H,G,Htilde,Gtilde} = Daubechies(4);

  w = wt(x,H,G,0);
  _wcoeff = w;
```



Graphique 21

```
  _scale = seqa(1,1,M);
  sv = 4|0;

  __output = 0;
  _print = 0;
  _tsm_Mcov = 0;

  {theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

  retp(theta[2]);
endp;
```

2.2.6 Ré-échantillonnage et simulation

Les méthodes de simulation prennent de plus en plus d'importance en économétrie[28]. Il existe plusieurs procédures dans TSM pour simuler des processus (SSM, ARMA vectoriel et arfima). La procédure `bootstrap_SSM` implémente l'algorithme du bootstrap pour les modèles espace-état en considérant la représentation en terme d'innovations[29]. La procédure `surrogate` permet d'obtenir des données de substitution. Nous avons utilisé l'algorithme FT. Cela permet notamment de tester les nonlinéarités[30] dans une série (unidimensionnelle ou multidimensionnelle[31]). Adam Kurpiel[32] a écrit le programme suivant pour tester la nonlinéarité d'une série unidimensionnelle en combinant la technique "surrogate" et le test LWG.

```
new;
```

[28] Van Dijk, H.K., A. Monfort et B.W. Brown [1995], Econometric Inference Using Simulation Techniques, John Wiley & Sons

[29] Stoffer, D.S. et K.D. Wall [1991], Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter, *Journal of the American Statistical Association*, **86**, 1024-1033

[30] Theiler, J., S. Eubank, A. Longtin, B. Galdrikian et J.D. Farmer [1992], Testing for nonlinearity in time series: the method of surrogate data, *Physica D*, **58**, 77-94

[31] Palus, M. [1995], Detecting nonlinearity in multivariate time series, Working paper, Santa Fe Institute

[32] Kurpiel, A. [1996], Une reconsidération de l'hypothèse de l'efficience du marché des changes par la méthode des réseaux de neuronnes artificiels, Mémoire de DEA, LARE

```
library tsm,optmum;
TSMset;

/* Simulate an arma(1,1) process */

{data1,retcode} = RND_arfima(0.5|-0.2|0,1,1,1,1000,512,1);

data2 = (data1 + cos(data1) )^2;

_fourier = 0;  /* Using the fast transform */
_surrogate_random = 1; /* uniform [0,2pi] randomization of the phases */

output file = nonlin.out reset;

seed = 123;
call Test(data1,25,3,4,seed);

print;

seed = 123;
call Test(data2,25,3,4,seed);

output off;


/* {sigma,pvalue} = Test(y,Ns,order1,order2,seed} */

/* y: data                                      */
/* Ns: number of FT surrogate simulations       */
/* order1:  the linear autoregressive order     */
/* order2:  the non-linear autoregressive order */
/* seed: seed parameter                         */

/* sigma: the surrogate test                    */
/* pvalue: p-value of the surrogate test        */


proc (2) = Test(y,Ns,order1,order2,seed);
  local simul,TR2,pvalue,TR2s,s,ys,sigma;

  y = packr(y);

  if cols(y) /= 1;
    ERRORLOG "error: Not valid for multivariate data.";
    end;
  endif;

  {TR2,pvalue} = LWG(y,order1,order2,seed);

  simul = {};

  s = 1;
  do until s > Ns;
    cls; locate 4,5; printdos ftos(s,"surrogate no %lf",5,0);
    ys = surrogate(y);
    simul = simul~ys;
    s = s + 1;
  endo;

  TR2s = zeros(Ns,1);
```

```
  s = 1;
  do until s > Ns;
    {TR2s[s],pvalue} = LWG(simul[.,s],order1,order2,seed);
    s = s + 1;
  endo;

  sigma = abs(TR2 - meanc(TR2s))/stdc(TR2s);
  pvalue = erfc(sigma/sqrt(2));

  if _print == 1;

    cls;
    print "-------------------------------------------";
    print "LWG-Surrogate test for neglected non-linearity";
    print "-------------------------------------------";
    print;
    print ftos(Ns,"Number of surrogate:                %lf",1,0);
    print;
    print ftos(TR2,"LWG test for the original data:      %lf",5,5);
    print ftos(meanc(TR2s),"LWG test for the surrogate data:      %lf",5,5);
    print;
    print ftos(sigma,"Surrogate test:                      %lf",5,5);
    print ftos(pvalue,"p_value:                             %lf",5,5);

  endif;

  retp(sigma,pvalue);
endp;


proc (2) = LWG(data,p,J,seed);
  local tmin,tmax,t,Nobs,y,x,i,xi,z;
  local K,beta,u,phi,M,delta,e,R2,TR2,pvalue;

  /* target data */

  tmin = 0.1; tmax = 0.9;
  t = tmin + (tmax-tmin)*(data-minc(data)')./(maxc(data)'-minc(data)');

  Nobs = rows(t);

  /* endogeneous data */

  y = t[p+1:Nobs];

  /* exogeneous data */

  x = {};
  i = 1;
  do until i > p;
    xi = t[1+p-i:Nobs-i];
    x = x~xi;
    i = i+1;
  endo;

  /* Add a constant */

  z = ones(Nobs-p,1)~x;
```

```
  I = cols(z);    /* Input Number */
  K = cols(y);    /* Output Number */

  /* OLS: data(t) = beta0 + beta1*data(t-1) + ... + betap*data(t-p) + u(t) */

  beta = y/z;

  /* Residuals */

  u = y - z*beta;

  /* Neural Net */

  phi = Activation(z,J,seed);

  /* Form the regressor matrix */

  M = z~phi;

  /* Residuals of the Neural Net model */

  delta = u/M;
  e = u - M*delta;

  /* T*R^2 */

  R2 = 1 - (e'e)/(u'u);
  TR2 = (Nobs-p)*R2;

  /* pvalue */

  pvalue = cdfchic(TR2,J);

  retp(TR2,pvalue);
endp;


proc Net_Function(x);
  local y;
  y = 1./(1+exp(-x));
  retp(y);
endp;


proc Activation(x,J,seed);
  local Nx,u,ht,at;

  rndseed seed;

  Nx = cols(x);

  u = 4*rndu(J,Nx) - 2;  /* [-2,2] uniform random */
  ht = x*u';
  at = Net_Function(ht);

  retp(at);
endp;
```

Voici le contenu du fichier *nonlin.out* :

```
-----------------------------------------------
```

```
LWG-Surrogate test for neglected non-linearity
----------------------------------------------

Number of surrogate:                25

LWG test for the original data:     1.03595
LWG test for the surrogate data:     4.46332

Surrogate test:                      0.92943
p_value:                             0.35266

----------------------------------------------
LWG-Surrogate test for neglected non-linearity
----------------------------------------------

Number of surrogate:                25

LWG test for the original data:     9.34500
LWG test for the surrogate data:     3.45165

Surrogate test:                      2.98179
p_value:                             0.00287
```

2.3 Liste des procédures de TSM

1. Processus ARMA.
   (a) **arma_ML** : Maximum de vraisemblance conditionnelle d'un modèle ARMA vectoriel
   (b) **arma_CML** : Maximum de vraisemblance conditionnelle d'un modèle ARMA vectoriel avec restrictions linéaires
   (c) **arma_to_VAR1** : Représentation VAR(1) d'un processus ARMA vectoriel
   (d) **arma_roots** : Racines du polynôme caractéristique associé à la représentation VAR(1) d'un processus ARMA vectoriel
   (e) **canonical_arma** : Représentations canoniques (autorégressive et moyenne-mobile) d'un processus ARMA vectoriel
   (f) **arma_autocov** : Autocovariances et autocorrélations d'un processus ARMA vectoriel
   (g) **arma_impulse** : Réponses aux erreurs de prévisions d'un processus ARMA vectoriel
   (h) **arma_orthogonal** : Réponses aux impulsions orthogonales d'un processus ARMA vectoriel
   (i) **arma_fevd** : Décomposition de la variance de l'erreur de prévision d'un processus ARMA vectoriel
   (j) **arma_to_SSM** : Forme espace état d'un modèle ARMA vectoriel
   (k) **Hankel** : Matrice de Hankel d'une série temporelle multivariée
2. Processus VARX.
   (a) **varx_LS** : Estimation d'un processus VARX par moindres carrés multivariés
   (b) **varx_CLS** : Estimation d'un processus VARX avec restrictions linéaires par moindres carrés multivariés
   (c) **varx_ML** : Estimation d'un processus VARX par maximum de vraisemblance
   (d) **varx_CML** : Estimation d'un processus VARX avec restrictions linéaires par maximum de vraisemblance
3. Analyse spectrale.
   (a) **fourier** : Transformée de Fourier
   (b) **inverse_fourier** : Transformée inverse de Fourier
   (c) **fourier2** : Transformée de Fourier de deux séries temporelles réelles
   (d) **PDGM** : Périodogramme d'une série temporelle univariée
   (e) **PDGM2** : Périodogramme d'une série temporelle multivariée

(f) **CPDGM** : Périodogramme croisé

(g) **CSpectrum** : Cohérences, amplitudes croisées et phases spectrales

(h) **Smoothing** : Lissage (Data windowing) dans le domaine des fréquences

4. Estimation par maximum de vraisemblance.

   (a) Estimation dans le domaine du temps.

      i. **TD_ml** : Estimation MV dans le domaine du temps

      ii. **TD_cml** : Estimation MV avec restrictions linéaires dans le domaine du temps

      iii. **TDml_derivatives** : Calcule le jacobien, le gradient, la matrice hessienne dans le domaine du temps

   (b) Estimation dans le domaine des fréquences des processus univariés.

      i. **FD_ml** : Estimation MV dans le domaine des fréquences

      ii. **FD_cml** : Estimation MV avec restrictions linéaires dans le domaine des fréquences

      iii. **FDml_derivatives** : Calcule le jacobien, le gradient et la matrice hessienne et la matrice d'information dans le domaine des fréquences

5. Modèles univariés.

   (a) **sm_LL** : Estimation du modèle *local level/random walk plus noise model*

   (b) **sm_LLT** : Estimation du modèle *local linear trend model*

   (c) **BSM** : Estimation du modèle *basic structural model*

   (d) **sm_cycle** : Estimation du modèle *cycle model*

   (e) **arfima** : Estimation d'un processus ARMA fractionnaire avec contraintes

   (f) **canonical_arfima** : Représentation canonique (autorégressive et moyenne-mobile) d'un processus ARMA fractionnaire

   (g) **sgf_arfima** : Fonction génératrice spectrale d'un processus ARMA fractionnaire

6. Les modèles espace état et le filtre de Kalman.

   (a) **SSM** : Affiche le modèle espace état

   (b) **SSM_build** : Initialise le modèle espace état

   (c) **SSM_ic** : Conditions initiales du modèle espace invariant dans le temps

   (d) **KFiltering** : Filtre de Kalman

   (e) **KF_matrix** : Matrices définies par le filtre de Kalman

   (f) **KF_gain** : Matrices de gain d'un modèle espace état

   (g) **KF_ml** : Vraisemblance du processus d'innovation

   (h) **KSmoothing** : Lissage

   (i) **KForecasting** : Prévision

   (j) **ARE** : Equation algébrique de Riccati

   (k) **sgf_SSM** : Fonction génératrice spectrale d'un modèle espace état invariant dans le temps

   (l) **SSM_autocov** : Autocovariances et autocorrélations d'un modèle espace état invariant dans le temps

   (m) **SSM_impulse** : Réponses aux erreurs de prévisions d'un modèle espace état invariant dans le temps

   (n) **SSM_orthogonal** : Réponses aux impulsions orthogonales d'un modèle espace état invariant dans le temps

   (o) **SSM_fevd** : Décomposition de la variance de l'erreur de prévision d'un modèle espace état invariant dans le temps

   (p) **SSM_Hankel** : Matrice de Hankel d'un modèle espace état invariant dans le temps

7. Ré-échantillonnage et simulation.

   (a) **Bootstrap** : Bootstrap d'une matrice

   (b) **SSM_Bootstrap** : Bootstrap d'un modèle espace état

(c) **surrogate** : Technique des données de substitution (Fourier Transform Surrogate data technique)

(d) **Kernel** : Estimation de la densité par la méthode du noyau

(e) **RND_arma** : Simulation d'un processus ARMA vectoriel

(f) **RND_arfima** : Simulation d'un processus ARMA fractionnaire

(g) **RND_SSM** : Simulation d'un modèle espace état

8. Outils d'estimation pour l'analyse des séries temporelles.

   (a) **FLS** : Moindres carrés flexibles

   (b) **GFLS** : Moindres carrés flexibles généralisés de Kalaba et Tesfatsion [1990]

   (c) **GFLS2** : Moindres carrés flexibles généralisés de Lütkepohl and Herwartz [1996]

   (d) **GMM** : Méthodes des moments généralisés

   (e) **RLS** : Moindres carrés récursifs

9. Analyse temps-fréquences.

   (a) Filtres miroirs en quadrature.

      i. **Coiflet** : Filtres de Coiflet

      ii. **Daubechies** : Filtres de Daubechies

      iii. **Haar** : Filtre de Haar

      iv. **Pollen** : Filtres de Pollen

   (b) Analyse par ondelettes.

      i. Transformée discrète en ondelettes.

         A. **iwt** : Transformée inverse en ondelettes d'un signal de dimension 1

         B. **iwt_matrix** : Matrice de transformation en ondelettes

         C. **wt** : Transformée en ondelettes d'un signal de dimension 1

         D. **wt_matrix** : Matrice de transformation inverse en ondelettes

      ii. Outils pour l'analyse en ondelettes.

         A. **extract** : Extraction d'une sous-bande des coefficients de décomposition en ondelettes

         B. **insert** : Insertion d'une sous-bande dans le vecteur des coefficients de décomposition en ondelettes

         C. **Scalogram** : Scalogramme (décomposition de l'énergie) des coefficients de décomposition en ondelettes

         D. **select** : Sélection d'une sous-bande des coefficients de décomposition en ondelettes

         E. **split** : Décomposition des sous-bandes des coefficients de décomposition en ondelettes

         F. **wPlot** : Graphique des coefficients de décomposition en ondelettes

   (c) Analyse par paquets d'ondelettes.

      i. Transformée discrète en paquets d'ondelettes.

         A. **iwpkt** : Transformée inverse en paquets d'ondelettes

         B. **wpkPlot** : Graphique des coefficients de décomposition en paquets d'ondelettes

         C. **wpkt** : Transformée inverse en paquets d'ondelettes

      ii. Outils de recherche d'un base optimale.

         A. **Basis** : Sélection d'une base

         B. **BasisPlot** : Localisation temps-fréquences de la base (*Time-frequency* plane tilings plot)

         C. **BestBasis** : Sélection de la meilleure base (pruning algorithm)

         D. **BestLevel** : Sélection du meilleur niveau

         E. **Entropy** : Fonction de coût (entropie de Shannon)

         F. **isBasis** : vérifie si $\mathcal{B}$ est une base

         G. **LogEnergy** : Fonction de coût (logarithme de l'énergie)

         H. **LpNorm** : Fonction de coût (norme ℓ^p)

   (d) Méthodes de débruitage.

      i. **SemiSoft** : Semi-soft shrinkage

ii. **Thresholding** : Quantile thresholding
iii. **VisuShrink** : Visu shrinkage
iv. **WaveShrink** : Wavelet shrinkage (hard and soft)

10. Opérateurs matriciels.

(a) **vech_** : opérateur vech
(b) **xpnd_** : opérateur xpnd
(c) **Elimination_** : Matrice d'élimination
(d) **Duplication_** : Matrice de duplication
(e) **Commutation_** : Matrice de commutation
(f) **xpnd2** : Procedure pour le codage des matrices carrées
(g) **Explicit_to_Implicit** : Transforme un système de restrictions linéaires explicites en un système de restrictions linéaires implicites
(h) **Implicit_to_Explicit** : Transforme un système de restrictions linéaires implicites en un système de restrictions linéaires explicites

Partie II
Quelques applications de GAUSS en finance (II)

Ce document est la suite de "Quelques Applications de GAUSS en finance". La plupart des procédures qui sont présentées dans cette deuxième partie ont été développées pour illustrer le cours *Théorie Financière III* en 3ème année de Magistère d'Économie et Finance Internationales à l'Université Montesquieu-Bordeaux IV.

3 Gestion de portefeuille

3.1 Frontière de Markowitz

Références
Cobbaut, R. [1994], Théorie Financière, Economica (chapitre 4)
Schoenberg, R. [1995], Markowitz portfolio analysis for the individual investor, document de travail

On considère N actifs. On note $\tilde{R}_i$ le rendement aléatoire de l'actif i. Soit $\tilde{R}$ le vecteur aléatoire des rendements

$$\tilde{R} = \begin{bmatrix} \tilde{R}_1 \\ \vdots \\ \tilde{R}_N \end{bmatrix}$$

Une stratégie est définie par un vecteur α représentant la proportion des actifs dans le portefeuille. L'objectif de l'agent est alors

$$\begin{aligned} &\max_{\alpha} \Phi E\left[\tilde{R}_p\right] - \sigma^2\left[\tilde{R}_p\right] \\ &\text{s.c.} \quad \begin{cases} \sum_{i=1}^{N} \alpha_i = 1 \\ \alpha_i \geq 0 \end{cases} \end{aligned} \tag{21}$$

avec Φ le coefficient de préférence pour le rendement, $E\left[\tilde{R}_p\right]$ le rendement moyen du portefeuille et $\sigma\left[\tilde{R}_p\right]$ le risque du portefeuille. Sous l'hypothèse de normalité des rendements $\tilde{R} \sim \mathcal{N}(\mu, V)$, nous avons

$$E\left[\tilde{R}_p\right] = \alpha^\top \mu$$

et

$$\sigma^2\left[\tilde{R}_p\right] = \alpha^\top V \alpha$$

GAUSS possède une commande `Qprog` qui permet de résoudre le problème de programmation quadratique suivant :

$$\begin{array}{ll} \min\limits_{x} & \frac{1}{2}x^\top Qx - x^\top R \\ \text{s.c.} & \left\{ \begin{array}{l} Ax = B \\ Cx \geq D \\ e_1 \leq x \leq e_2 \end{array} \right. \end{array} \tag{22}$$

Nous pouvons donc déterminer le portefeuille optimal avec `Qprog`, puisque nous avons

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^\top (2V)\alpha - \alpha^\top (\Phi\mu) \\ \text{s.c.} & \left\{ \begin{array}{l} \mathbf{1}^\top \alpha = 0 \\ \mathbf{I}\alpha \geq \mathbf{0} \end{array} \right. \end{array} \tag{23}$$

Markowitz

■ **Objectif**
Détermination d'un portefeuille optimal de Markowitz.

■ **Format**
{alpha,Rp,SIGMAp} = Markowitz(mu,Mcov,PHI);

■ **Entrées**

PHI	scalaire, coefficient Φ de préférence pour le rendement.
mu	vecteur $N \times 1$, vecteur μ des rendements moyens.
Mcov	matrice $N \times N$, matrice V de covariance des rendements.

■ **Sorties**

alpha	vecteur $N \times 1$, stratégie optimale.
Rp	scalaire, rendement moyen du portefeuille optimal.
SIGMAp	scalaire, risque du portefeuille optimal.

La procédure `Markowitz` permet de déterminer la stratégie optimale α solution du programme suivant

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^\top (2V)\alpha - \alpha^\top (\Phi\mu) \\ \text{s.c.} & \left\{ \begin{array}{l} A\alpha = B \\ C\alpha \geq D \\ e_1 \leq \alpha \leq e_2 \end{array} \right. \end{array} \tag{24}$$

Les matrices A, B, C, D et $e = \left[\begin{array}{ccc} e_1 & \vdots & e_2 \end{array} \right]$ sont définies par les variables externes `_Markowitz_A`, `_Markowitz_B`, `_Markowitz_C`, `_Markowitz_D` et `_Markowitz_bnds`. Par défaut, ces matrices sont initialisées de telle façon que la stratégie optimale soit celle définie par le programme (23).

markowitz.dec

```
declare matrix _Markowitz_A = 0;
declare matrix _Markowitz_B = 0;
declare matrix _Markowitz_C = 0;
declare matrix _Markowitz_D = 0;
declare matrix _Markowitz_bnds = 0;
```

markowitz.ext

```
external matrix _Markowitz_A;
external matrix _Markowitz_B;
external matrix _Markowitz_C;
external matrix _Markowitz_D;
external matrix _Markowitz_bnds;
```

markowitz.src

```
proc (3) = Markowitz(mu,Mcov,PHI);
  local Nombre_Actifs;
  local sv,Q,R,A,B,C,D,bnds;
  local alpha,u1,u2,u3,u4,retcode;
  local Rendement_du_Portefeuille,Risque_du_Portefeuille;

  Nombre_Actifs = rows(mu);
  sv = ones(Nombre_Actifs,1)/Nombre_Actifs;
  Q = 2*Mcov;
  R = PHI*mu;

  if _Markowitz_A == 0;
    A = ones(1,Nombre_Actifs);
  else;
    A = _Markowitz_A;
  endif;

  if _Markowitz_B == 0;
    B = 1;
  else;
    B = _Markowitz_B;
  endif;

  if _Markowitz_C == 0;
    C = eye(Nombre_Actifs);
  else;
    C = _Markowitz_C;
  endif;

  if _Markowitz_D == 0;
    D = zeros(Nombre_Actifs,1);
  else;
    D = _Markowitz_D;
  endif;

  if _Markowitz_bnds == 0;
    bnds = 0;
  else;
    bnds = _Markowitz_bnds;
  endif;

  {alpha,u1,u2,u3,u4,retcode} = Qprog(sv,Q,R,A,B,C,D,bnds);

  if retcode /= 0;
    retp(error(0),error(0),error(0));
  endif;

  Rendement_du_Portefeuille = alpha'mu;
  Risque_du_Portefeuille = sqrt(alpha'Mcov*alpha);

  retp(alpha,Rendement_du_Portefeuille,Risque_du_Portefeuille);
endp;
```

L'exemple suivant est issu de "Markowitz portfolio analysis for the individual investor" par SCHOENBERG [1995]. Il considère 6 actifs *Merrill Lynch*, *Paine Webber*, *Digital Equipment*, *Microsoft*, *Silicon Graphics* et *Hewlett-Packard.* Les informations sur ces actifs sont contenus dans le programme **portef1.inf** :

```
let mu[6,1] = 15.852
              14.262
              31.336
              25.775
```

```
              50.228
              14.842;

let sigma[6,1] = 37.215
                 41.773
                 33.165
                 62.009
                 60.720
                 23.757;

let Mcor[6,6]  = 1.000  0.944  0.146  0.231  0.379  0.258
                 0.944  1.000  0.109  0.239  0.413  0.223
                 0.146  0.109  1.000 -0.169 -0.229  0.691
                 0.231  0.239 -0.169  1.000  0.882 -0.256
                 0.379  0.413 -0.229  0.882  1.000 -0.284
                 0.258  0.223  0.691 -0.256 -0.284  1.000;

Mcov = Mcor.*sigma.*sigma';
```

Le programme suivant reproduit les résultats de Schoenberg [1995].

```
new;
library finance;

#include portef1.inf;

Proportion = {};
Rendement = {};
Risque = {};

PHI = 20;
do until PHI > 70;
  {alpha,RP,SigmaP} = Markowitz(mu,Mcov,PHI);
  Proportion = Proportion~alpha;
  Rendement = Rendement~RP;
  Risque = Risque~SigmaP;
  PHI = PHI + 10;
endo;

cls;

output file = markow1.out reset;

format /rd 10,3;
print "=================================================================";
print "             PREFERENCE DE L'INVESTISSEUR POUR LE RETURN          ";
print "-----------------------------------------------------------------";
print seqa(20,10,6)';
print "=================================================================";
print "                     RENDEMENT MOYEN DU PORTEFEUILLE             ";
print "-----------------------------------------------------------------";
print Rendement;
print "=================================================================";
print "                          RISQUE DU PORTEFEUILLE                 ";
print "-----------------------------------------------------------------";
print Risque;
print "=================================================================";
print "                        COMPOSITION DU PORTEFEUILLE              ";
print "-----------------------------------------------------------------";
print proportion;
print "=================================================================";
```

```
output off;

================================================================
             PREFERENCE DE L'INVESTISSEUR POUR LE RETURN
----------------------------------------------------------------
    20.000     30.000     40.000     50.000     60.000     70.000
================================================================
                    RENDEMENT MOYEN DU PORTEFEUILLE
----------------------------------------------------------------
    29.721     33.384     37.047     38.065     38.377     38.690
================================================================
                         RISQUE DU PORTEFEUILLE
----------------------------------------------------------------
    21.198     23.258     25.868     26.686     27.006     27.380
================================================================
                       COMPOSITION DU PORTEFEUILLE
----------------------------------------------------------------

     0.000     -0.000      0.000      0.000     -0.000      0.000
    -0.000      0.000     -0.000     -0.000      0.000     -0.000
     0.336      0.481      0.626      0.644      0.627      0.611
     0.000      0.000     -0.000     -0.000      0.000     -0.000
     0.264      0.300      0.336      0.356      0.373      0.389
     0.400      0.219      0.038     -0.000      0.000     -0.000
================================================================
```

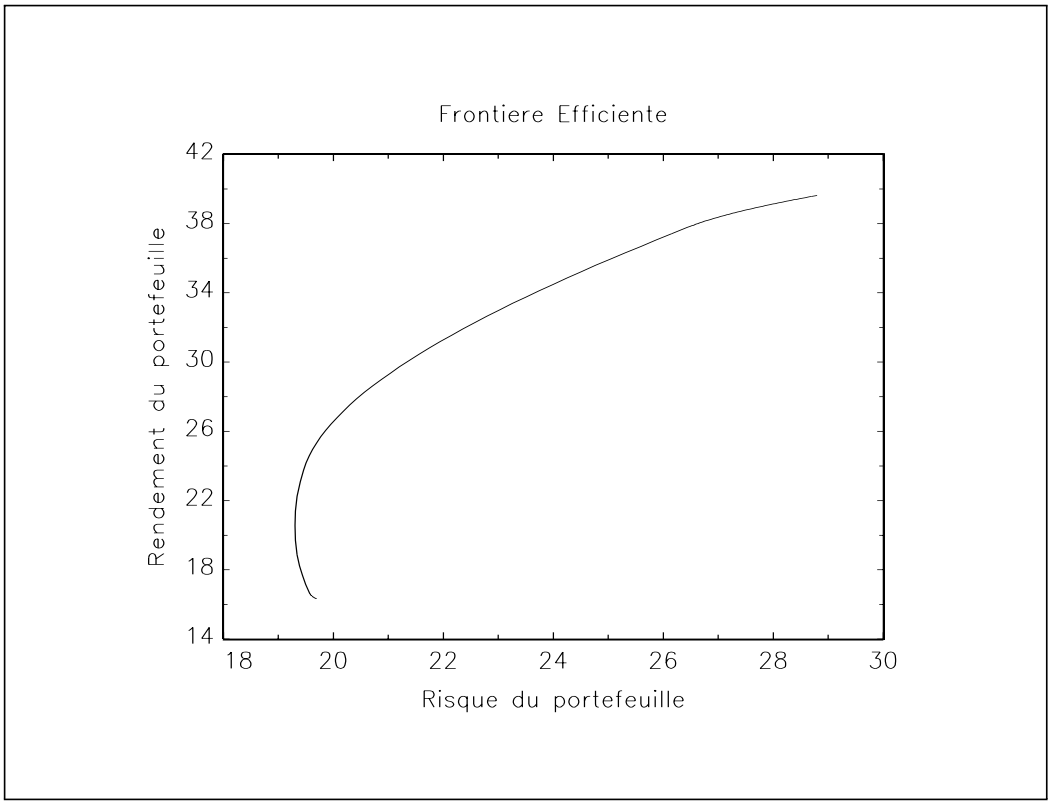
Pour obtenir la frontière efficiente, il suffit de faire varier Φ dans un intervalle $[\Phi_{\min}, \Phi_{\max}]$ et de représenter la courbe défini par les couples $\left(E\left[\tilde{R}_p\right], \sigma\left[\tilde{R}_p\right]\right)$.

```
new;
library finance,pgraph;

#include portef1.inf;

Rendement = {}; Risque = {};

PHI = -25;
do until PHI > 100;
  {alpha,RP,SigmaP} = Markowitz(mu,Mcov,PHI);
  Rendement = Rendement|RP;
  Risque = Risque|SigmaP;
  PHI = PHI + 1;
endo;


graphset;
  _pdate = ""; _pnum = 2; _paxht = 0.18; _pnumht = 0.18; _ptitlht = 0.18;
  title("Frontiere Efficiente");
  xlabel("Risque du portefeuille");
  ylabel("Rendement du portefeuille");
  xy(Risque,Rendement);
```

3.2 Prise en compte de préférences allocatives

La procédure `Markowitz` permet de prendre en compte les préférences allocatives de l'agent. Supposons par exemple que l'agent désire détenir 1/3 de son portefeuille an actions du secteur finance. Dans ce cas, la contrainte $A\alpha = B$ devient

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \alpha = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$$



Graphique 22

Si l'agent désire détenir au moins 25% de son portefeuille en actions *Paine Webber*, la contrainte $C\alpha \geq D$ devient

$$\mathbf{I}\alpha \geq \begin{bmatrix} 0 \\ 0.25 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
new;
library finance;
financeSet;

#include portef1.inf;

PHI = 20;    /* Preference de l'investisseur pour le return */

output file = markow3.out reset;

print "L'agent desire detenir 1/3 de son portefeuille en actions du secteur finance";
print "et 2/3 en actions du secteur informatique";

_Markowitz_A = {1 1 0 0 0 0,0 0 1 1 1 1};
_Markowitz_B = {0.33,0.66};
{alpha,RP,SigmaP} = Markowitz(mu,Mcov,PHI);
print alpha;


print "L'agent desire detenir au mois 25% de son portefeuille en actions Painer Webber";

_Markowitz_A = 0;
```

```
_Markowitz_B = 0;
_Markowitz_C = eye(6);
_Markowitz_D = 0|0.25|0|0|0|0;

{alpha,RP,SigmaP} = Markowitz(mu,Mcov,PHI);
print alpha;


print "L'agent desire recomposer son portefeuille";

alpha0 = 0.15|0.20|0.17|0.14|0.04|0.30;

_Markowitz_C = 0;
_Markowitz_D = 0;
_Markowitz_bnds = alpha0 + (-0.10~0.10);

{alpha,RP,SigmaP} = Markowitz(mu,Mcov,PHI);

print "Portefeuille d'origine    Nouveau portefeuille";
print alpha0~alpha;

SigmaP0 = sqrt(alpha0'*Mcov*alpha0);
RP0 = alpha0'mu;

print "Rendement moyen ";
print RP0~RP;
print "Risque";
print SigmaP0~SigmaP;

output off;


L'agent desire detenir 1/3 de son portefeuille en actions du secteur finance
et 2/3 en actions du secteur informatique

     0.330
    -0.000
     0.346
    -0.000
     0.158
     0.157


L'agent desire detenir au mois 25% de son portefeuille en actions Painer Webber

    -0.000
     0.250
     0.353
    -0.000
     0.175
     0.222


L'agent desire recomposer son portefeuille
Portefeuille d'origine    Nouveau portefeuille

     0.150      0.050
     0.200      0.100
     0.170      0.270
     0.140      0.040
```

```
     0.040      0.140
     0.300      0.400
Rendement moyen
    20.628     24.679
Risque
    23.063     20.669
```

Le dernier exemple du programme précédent considère une réallocation de portefeuille. Le portefeuille original de l'agent est

$$\alpha_0 = \begin{bmatrix} 0.15 \\ 0.20 \\ 0.17 \\ 0.14 \\ 0.04 \\ 0.30 \end{bmatrix}$$

Cet agent désire reconstruire son portefeuille, par exemple modifier sa composition de la façon suivante

$$-0.10 \leq \alpha - \alpha_0 \leq 0.10$$

Pour obtenir le nouveau portefeuille, nous utilisons donc la procédure `Markowitz` avec `_Markowitz_bnds` = $\begin{bmatrix} \alpha_0 - 0.1 & \vdots & \alpha_0 + 0.1 \end{bmatrix}$.

La procédure `Markowitz` permet aussi de prendre en compte l'existence d'un marché à terme. Ainsi, autoriser la vente de contrats à terme sur l'actif i revient à imposer la restriction $\alpha_i \leq 0$. Par exemple, si nous voulons vendre à terme les actions *Paine Webber*, la constrainte $C\alpha \geq D$ devient

$$\begin{bmatrix} 1 & & & & & \\ & -1 & & & \mathbf{0} & \\ & & 1 & & & \\ & & & 1 & & \\ & \mathbf{0} & & & 1 & \\ & & & & & 1 \end{bmatrix} \alpha \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

3.3 Temps de calcul

Dans le programme suivant, nous évaluons les temps de calcul nécessaires à la détermination de la stratégie optimale en fonction du nombre d'actifs avec un Pentium 100 Mhz.

```
new;
library finance;
financeSet;

proc (2) = Information(N);
  local mu,P,V;

  mu = rndu(N,1);
  P = lowmat(xpnd(rndn(N*(N+1)/2,1)));
  V = P*P';

  retp(mu,V);
endp;

output file = markow4.out reset;

print "Nombre d'Actifs    Temps de calcul (en secondes)";
print "-----------------------------------------------";

N = 25|50|75|100|150|200|250|300;

i = 1;
do until i > rows(N);
```

```
  {mu,V} = Information(N[i]);

  debut = hsec;
  {alpha,RP,SigmaP} = Markowitz(mu,V,20);
  Temps_de_Calcul = (hsec-debut)/100;
  str = ftos(N[i],"      %lf",4,0);
  str = str $+ ftos(Temps_de_Calcul,"                    %lf",6,3);
  print str;

  i = i + 1;
endo;

output off;
```

Nombre d'Actifs	Temps de calcul (en secondes)
25	0.000
50	0.110
75	0.270
100	0.820
150	3.240
200	7.200
250	14.060
300	25.870

3.4 Construction des informations sur les rendements des actifs

Dans l'exemple précédent, les matrices μ et V sont données. Mais, nous pouvons facilement les estimer avec **GAUSS**. Supposons que la matrice `X` de dimension $T \times N$ représente l'évolution mensuelle des prix des N actifs $P_t^{(n)}$. Pour chaque actif n, nous calculons le rendement annuel de la façon suivante

$$r_t^{(n)} = \left[1 + \frac{P_t^{(n)} - P_{t-1}^{(n)}}{P_{t-1}^{(n)}}\right]^{12} - 1$$

Nous pouvons alors obtenir une estimation des matrices μ et V très simplement :

```
proc (2) = Estimation_mu_V(X);
  local Y,Rendement_mensuel,Rendement_annuel,mu,V;

  Y = lag1(X);

  Rendement_mensuel = (X - Y)./Y;
  Rendement_annuel  = (1 + Rendement_mensuel)^12 -1;

  /* Elimination de la premiere donnee, qui est une valeur manquante */
  Rendement_annuel = trimr(Rendement_annuel,1,0);

  mu = meanc(Rendement_annuel);
  V = vcx(Rendement_annuel);

  retp(mu,V);
endp;
```

Cette procédure peut aussi être utilisée pour calculer les coefficients β des actifs. Nous avons

$$\beta_i = \frac{\text{cov}\left(\tilde{R}_i, \tilde{R}_M\right)}{\text{var}\left(\tilde{R}_M\right)}$$

avec $\tilde{R}_M$ le rendement d'un portefeuille de marché (en général, un indice). Supposons que la matrice `X` représente l'évolution mensuelle de l'indice et des prix des actifs (l'indice est placé dans la première colonne). Nous avons

```
{mu,V} = Estimation_mu_V(X);
Variance_PM = V[1,1];
Covariance_PM_Actifs = trimr(V[.,1],1,0);
beta = Covariance_PM_actifs/Variance_PM;
```

4 Une procédure de bi-section vectorisée

Il existe différents algorithmes pour résoudre l'équation

$$f(x) = 0$$

En particulier, si la fonction f est strictement croissante, nous pouvons obtenir la solution de l'équation en utilisant l'algorithme de la bi-section. Supposons deux scalaires a et b tels que

$$f(a) < 0 < f(b)$$

alors la solution de l'équation $f(x) = 0$ vérifie $a < x < b$. Considérons le point c tel que $c = \frac{a+b}{2}$. Alors si $f(c) \leq 0$, la solution appartient à l'intervalle $[c,b]$. Dans le cas contraire, la solution appartient à l'intervalle $[a,c]$. Nous avons ainsi réduit de moitié l'intervalle contenant la solution. Nous pouvons recommencer l'opération. Dans ce cas, la longueur du nouveau intervalle sera quatre fois plus petite que celle de l'intervalle original. En réitérant l'opération un certain nombre de fois, nous pouvons trouver un intervalle relativement petit qui encadre la solution.

La programmation de cet algorithme en langage **GAUSS** est très intéressante, car nous pouvons vectorisée la procédure. Cela veut dire qu'un seul appel de la procédure (et non N appels) permet de calculer la solution de plusieurs équations

$$\begin{bmatrix} f_1(x) \\ \vdots \\ f_N(x) \end{bmatrix} = 0$$

Il existe d'autres algorithmes pour résoudre une équation non linéaire (Newton-Raphson, Broyden, etc). En finance, nous avons souvent besoin de résoudre une équation non linéaire (par exemple, pour calculer un taux de rendement actuariel, la volatilité implicite d'une option, etc). La simplicité de l'algorithme de la bi-section et la vectorisation en font un outil particulièrement performant. Dans la première partie de *Quelques Applications de GAUSS en finance*, nous avions calculé la volatilité implicite d'une option sur future avec cet algorithme. **Le calcul de 10000 volatilités implicites avec un Pentium 100 Mhz avait pris moins de 10 secondes.**

bisection

■ **Objectif**
Résolution de l'équation $f(x) = 0$ par l'algorithme de la bi-section.

■ **Format**
sol = bisection(&f,a,b);

■ **Entrées**

&f	pointeur d'une procédure qui calcule la fonction $f(x)$.
a	vecteur $N \times 1$, paramètre a.
b	vecteur $N \times 1$, paramètre b.

■ **Sorties**

sol	vecteur $N \times 1$, solution de l'équation $f(x) = 0$.

secante.dec

```
declare matrix _bisection_tolerance = 0.00001;
```

secante.ext

```
external matrix _bisection_tolerance;
```

secante.src

```
proc bisection(&f,a,b);
  local f:proc;
  local ya,yb,Nobs,c,yc,indx1,indx2;
  local indx,s,diff,const;

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

  indx = (ya .< 0) .and (yb .> 0);
  if sumc(indx) == 0;
    retp(miss(0,0));
  endif;

  if sumc(indx) == Nobs;

    do while maxc(abs(a-b)) > _bisection_tolerance;
      c = (a+b)/2;
      yc = f(c);
      indx1 = yc.<0;
      indx2 = 1 - indx1;
      a = indx1.*c + indx2.*a;
      b = indx1.*b + indx2.*c;
    endo;

  else;

    s = delif(seqa(1,1,Nobs),indx);
    diff = selif(a-b,indx);
    const = miss(zeros(Nobs-sumc(indx),1),0);

    do while maxc(abs(diff)) > _bisection_tolerance;
      c = (a+b)/2;
      c[s] = const;
      yc = f(c);
      indx1 = yc.<0;
      indx2 = 1 - indx1;
      a = indx1.*c + indx2.*a;
      b = indx1.*b + indx2.*c;
      diff = selif(a-b,indx);
    endo;

  endif;

  c = (a+b)/2;

  retp(c);
endp;
```

Trouvons un point fixe pour la fonction $f_1(x) = \cos x$ et une solution à l'équation $x^2 + 2\exp x + \sin(\cos x - 2x) = 8$.

```
new;
library finance;

proc f1(x);
  local y;
  y = x - cos(x);
  retp(y);
endp;

proc f2(x);
```

```
  local y;
  y = x.*x + 2*exp(x) + sin(cos(x)-2*x) - 8;
  retp(y);
endp;

proc f(x);
  local y1,y2,y;
  y1 = f1(x[1]);
  y2 = f2(x[2]);
  y = y1|y2;
  retp(y);
endp;

sol1 = bisection(&f1,0,2);
sol2 = bisection(&f2,0,2);
sol = bisection(&f,0|0,2|2);

output file = secante.out reset;

print ftos(sol1,"solution de la premiere equation : %lf",5,5);
print ftos(sol2,"solution de la deuxieme equation : %lf",5,5);

print; print;
print "solution de l'equation multidimensionnelle :";
print sol;

output off;


solution de la premiere equation : 0.73909
solution de la deuxieme equation : 1.27411


solution de l'equation multidimensionnelle :

      0.73908615
       1.2741051
```

5 Valorisation des options

5.1 La méthode de l'approximation quadratique de Barone-Adesi et Whaley [1987]

Références

Barone-Adesi, G. et R. E. Whaley [1987], Efficient analytic approximation of american option values, *Journal of Finance*, **42**, pages 301-320

Black, F. [1976], The pricing of commodity contracts, *Journal of Financial Economics*, **3**, pages 167-179

Black, F. et M. Scholes [1973], The pricing of options and corporate liabilities, *Journal of Political Economy*, **81**, pages 637-659

Garman, M.B. et S.W. Kohlhagen [1983], Foreign currency option values, *Journal of International Money and Finance*, **2**, pages 231-237

Considérons une option de prix d'exercice K et de maturité τ. La valeur actuelle de l'actif sous-jacent est S_0. Soient b et r le paramètre "cost of carry" et le taux d'intérêt. Alors les valeurs de l'option européenne d'achat et de vente sont données par les formules suivantes

$$\begin{aligned} C_{\text{EU}}(S_0) &= S_0 e^{(b-r)\tau}\Phi(d_1) - Ke^{-r\tau}\Phi(d_2) \\ P_{\text{EU}}(S_0) &= -S_0 e^{(b-r)\tau}\Phi(-d_1) + Ke^{-r\tau}\Phi(-d_2) \end{aligned}$$

avec

$$d_1 = d_1(S_0) = \frac{\ln\left(\frac{S_0}{K}\right) + \left(b + \frac{1}{2}\sigma^2\right)\tau}{\sigma\sqrt{\tau}}$$

et

$$d_2 = d_2(S_0) = d_1 - \sigma\sqrt{\tau}$$

Nous rappelons que le paramètre "cost of carry" b prend respectivement les valeurs r, 0 et $r - r^\star$ pour les modèles Black-Scholes [1973], Black [1976] (options sur future) et Garman-Kohlhagen [1983] (options de change[33]). Barone-Adesi et Whaley [1987] montrent alors que le prix de l'option américaine peut se déduire de celui de l'option européenne. Posons

$$\begin{aligned} M &= 2\frac{r}{\sigma^2} \\ N &= 2\frac{b}{\sigma^2} \\ L &= 1 - e^{-r\tau} \end{aligned}$$

Nous avons alors pour l'option d'achat

$$C_{\text{AM}}(S_0) = \begin{cases} C_{\text{EU}}(S_0) + A_2\left(\frac{S_0}{S^\star}\right)^{q_2} & \text{si } S_0 < S^\star \\ S_0 - K & \text{si } S_0 \geq S^\star \end{cases}$$

avec

$$A_2 = \left[1 - e^{(b-r)\tau}\Phi\left(d_1(S^\star)\right)\right]\frac{S^\star}{q_2}$$

et

$$q_2 = \frac{1}{2}\left(-(N-1) - \sqrt{(N-1)^2 + 4\frac{M}{L}}\right)$$

La valeur $S^\star$ est la solution de l'équation non linéaire suivante

$$S^\star - K = C_{\text{EU}}(S^\star) + \left[1 - e^{(b-r)\tau}\Phi\left(d_1(S^\star)\right)\right]\frac{S^\star}{q_2}$$

Pour une option de vente, nous avons

$$P_{\text{AM}}(S_0) = \begin{cases} P_{\text{EU}}(S_0) + A_1\left(\frac{S_0}{S^\star}\right)^{q_1} & \text{si } S_0 > S^\star \\ K - S_0 & \text{si } S_0 \leq S^\star \end{cases}$$

avec

$$A_1 = -\left[1 - e^{(b-r)\tau}\Phi\left(-d_1(S^\star)\right)\right]\frac{S^\star}{q_1}$$

et

$$q_1 = \frac{1}{2}\left(-(N-1) + \sqrt{(N-1)^2 + 4\frac{M}{L}}\right)$$

La valeur $S^\star$ est la solution de l'équation non linéaire suivante

$$K - S^\star = P_{\text{EU}}(S^\star) - \left[1 - e^{(b-r)\tau}\Phi\left(-d_1(S^\star)\right)\right]\frac{S^\star}{q_1}$$

American_BS

■ **Objectif**
Valorisation d'une option américaine.

■ **Format**
V = American_BS(S0,K,sigma,tau,b,r);

■ **Entrées**

S0	vecteur $N \times 1$, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
sigma	vecteur $N \times 1$, volatilité.
tau	vecteur $N \times 1$, maturité de l'option.
b	vecteur $N \times 1$, paramètre "cost of carry".
r	vecteur $N \times 1$, taux d'intérêt.

■ **Sorties**

V	vecteur $N \times 1$, valeur de l'option.

[33] $r^\star$ est le taux d'intérêt étranger.

■ **Remarque**
Nous précisons le type de l'option avec la variable externe `_type_option` (`''call''` ou `''put''`).

Barone-Adesi et Whaley [1987] utilisent pour résoudre l'équation non linéaire l'algorithme de Newton-Raphson. Dans la procédure `American_BS`, nous employons l'algorithme de la bi-section. La programmation de la méthode de l'approximation quadratique est ainsi facilitée. Cela permet en plus de vectoriser la procédure.

barone.dec

```
declare string _type_option  ?= "call";
declare matrix _option_S0;
declare matrix _option_K;
declare matrix _option_tau;
declare matrix _option_sigma;
declare matrix _option_r;
declare matrix _option_b;
declare matrix _option_parameters;
```

barone.ext

```
external string _type_option;
external matrix _option_S0;
external matrix _option_K;
external matrix _option_tau;
external matrix _option_sigma;
external matrix _option_r;
external matrix _option_b;
external matrix _option_parameters;
```

barone.src

```
proc American_BS(S0,K,sigma,tau,b,r);
  local w,d1,d2,V;
  local N,M,L,q1,q2,Stilde;
  local fmin,g,retcode;
  local A1,A2,V1,V2,indx;
  local indx_,s;

  _option_S0 = S0;
  _option_K = K;
  _option_sigma = sigma;
  _option_tau = tau;
  _option_b = b;
  _option_r = r;

  /* European Pricing Option (equations 5 and 6) */

  w = sigma.*sqrt(tau);
  d1 = (ln(S0./K)+(b+0.5*sigma^2).*tau)./w;  d2 = d1 - w;

  if _type_option $== "put";
    V = -S0.*exp((b-r).*tau).*cdfn(-d1) + K.*exp(-r.*tau).*cdfn(-d2);
  else;
    V = S0.*exp((b-r).*tau).*cdfn(d1) - K.*exp(-r.*tau).*cdfn(d2);
  endif;

  /* Footnote 6, page 307 */

  if b > r;
    retp(V);
  endif;
```

```
  /* Solution of the equation 13 */

  M = 2*r./(sigma^2);
  N = 2*b./(sigma^2);
  L = 1-exp(-r.*tau);

  w = (N-1)^2 + 4*M./L; w = sqrt(w);

  q1 = (-(N-1) - w)/2;
  q2 = (-(N-1) + w)/2;

  _option_parameters = M~N~L~q1~q2;

  Stilde = bisection(&_American_Stilde,S0/10,10*S0);
  indx_ = Stilde .== miss(0,0);
  if sumc(indx_) /= 0;
    s = selif(seqa(1,1,rows(Stilde)),indx_);
    Stilde[s] = S0[s];
  endif;

  w = sigma.*sqrt(tau);
  d1 = (ln(Stilde./K)+(b+0.5*sigma^2).*tau)./w;  d2 = d1 - w;

  if _type_option $== "put";

    indx = S0 .> Stilde;
    A1 = -(Stilde./q1).*(1-exp((b-r).*tau).*cdfn(-d1));
    V1 = V + A1.*(S0./Stilde)^q1;
    V2 = K - S0;
    V = indx.*V1 + (1-indx).*V2;

  else;

    indx = S0 .< Stilde;
    A2 = (Stilde./q2).*(1-exp((b-r).*tau).*cdfn(d1));
    V1 = V + A2.*(S0./Stilde)^q2;
    V2 = S0 - K;
    V = indx.*V1 + (1-indx).*V2;

  endif;

  if sumc(indx_) /= 0;
    V[s] = miss(zeros(rows(s),1),0);
  endif;

  retp(V);
endp;


proc _American_Stilde(Stilde);
  local K,sigma,tau,b,r;
  local M,N,L,q1,q2;
  local w,d1,d2,A1,A2,V,eqtn;

  K = _option_K;
  sigma = _option_sigma;
  tau = _option_tau;
  b = _option_b;
  r = _option_r;
  M = _option_parameters[.,1];
```

```
  N = _option_parameters[.,2];
  L = _option_parameters[.,3];
  q1 = _option_parameters[.,4];
  q2 = _option_parameters[.,5];

  w = sigma.*sqrt(tau);
  d1 = (ln(Stilde./K)+(b+0.5*sigma^2).*tau)./w;  d2 = d1 - w;

  if _type_option $== "put";
    A1 = -(Stilde./q1).*(1-exp((b-r).*tau).*cdfn(-d1));
    V = -Stilde.*exp((b-r).*tau).*cdfn(-d1) + K.*exp(-r.*tau).*cdfn(-d2);
    eqtn = V + A1 - K + Stilde;
  else;
    V = Stilde.*exp((b-r).*tau).*cdfn(d1) - K.*exp(-r.*tau).*cdfn(d2);
    A2 = (Stilde./q2).*(1-exp((b-r).*tau).*cdfn(d1));
    eqtn = -(V + A2 + K - Stilde);
  endif;

  retp(eqtn);
endp;
```

Le programme suivant reproduit les tables I-V de l'article de Barone-Adesi et Whaley [1987].

```
new;
library finance;

K = 100;
e = ones(5,1);
r = (0.08|0.12|0.08|0.08).*.e;
sigma = (0.20|0.20|0.40|0.20).*.e;
tau = (0.25|0.25|0.25|0.50).*.e;
S0 = ones(4,1).*.seqa(80,10,5);

output file = barone.out reset;

debut = hsec;

/* Table I */

b = -0.04*ones(20,1);

_type_option = "call";
C = American_BS(S0,K,sigma,tau,b,r);
_type_option = "put";
P = American_BS(S0,K,sigma,tau,b,r);

Print "                          Table I       ";
call Print_Table(b,r,sigma,tau,S0,C,P);


/* Table II */

b = 0.04*ones(20,1);

_type_option = "call";
C = American_BS(S0,K,sigma,tau,b,r);
_type_option = "put";
P = American_BS(S0,K,sigma,tau,b,r);

Print "                          Table II       ";
call Print_Table(b,r,sigma,tau,S0,C,P);
```

```
/* Table III */

b = 0.00*ones(20,1);

_type_option = "call";
C = American_BS(S0,K,sigma,tau,b,r);
_type_option = "put";
P = American_BS(S0,K,sigma,tau,b,r);

Print "                          Table III       ";
call Print_Table(b,r,sigma,tau,S0,C,P);


/* Table IV */

b = r;

C = miss(zeros(20,1),0);
_type_option = "put";
P = American_BS(S0,K,sigma,tau,b,r);

Print "                          Table IV       ";
call Print_Table(b,r,sigma,tau,S0,C,P);

/* Table V */

b = (-0.04|0.00|0.04|0.08).*.e;
r = (0.08|0.08|0.08|0.08).*.e;
sigma = (0.20|0.20|0.20|0.20).*.e;
tau = (3|3|3|3).*.e;

_type_option = "call";
C = American_BS(S0,K,sigma,tau,b,r);
_type_option = "put";
P = American_BS(S0,K,sigma,tau,b,r);

Print "                          Table V       ";
call Print_Table(b,r,sigma,tau,S0,C,P);

fin = hsec;

print ftos((fin-debut)/100,"Temps de calcul : %lf secondes",5,3);


output off;


proc (0) = Print_Table(b,r,sigma,tau,S0,C,P);
  local mask,fmt,omat;

  mask = 1;
  let fmt[7,3]=   "*.*lf" 8 2 "*.*lf" 8 2  "*.*lf" 8 2
                  "*.*lf" 8 2 "*.*lf" 8 0 "*.*lf" 15 3  "*.*lf" 15 3;

  omat = b~r~sigma~tau~S0~C~P;
  print "     b       r     sigma    tau       S0          Call           Put";
  print "-------------------------------------------------------------------"\
        "-----";
```

```
  call printfm(omat,mask,fmt);
  print;  print;
  retp;
endp;
```

Table I

b	r	sigma	tau	S0	Call	Put
-0.04	0.08	0.20	0.25	80	0.032	20.419
-0.04	0.08	0.20	0.25	90	0.590	11.251
-0.04	0.08	0.20	0.25	100	3.525	4.397
-0.04	0.08	0.20	0.25	110	10.315	1.118
-0.04	0.08	0.20	0.25	120	20.000	0.184
-0.04	0.12	0.20	0.25	80	0.032	20.248
-0.04	0.12	0.20	0.25	90	0.587	11.146
-0.04	0.12	0.20	0.25	100	3.506	4.355
-0.04	0.12	0.20	0.25	110	10.288	1.107
-0.04	0.12	0.20	0.25	120	20.000	0.183
-0.04	0.08	0.40	0.25	80	1.067	21.463
-0.04	0.08	0.40	0.25	90	3.284	13.927
-0.04	0.08	0.40	0.25	100	7.411	8.274
-0.04	0.08	0.40	0.25	110	13.502	4.523
-0.04	0.08	0.40	0.25	120	21.233	2.297
-0.04	0.08	0.20	0.50	80	0.229	20.982
-0.04	0.08	0.20	0.50	90	1.387	12.645
-0.04	0.08	0.20	0.50	100	4.724	6.372
-0.04	0.08	0.20	0.50	110	10.955	2.650
-0.04	0.08	0.20	0.50	120	20.000	0.919

Table II

b	r	sigma	tau	S0	Call	Put
0.04	0.08	0.20	0.25	80	0.052	20.000
0.04	0.08	0.20	0.25	90	0.849	10.183
0.04	0.08	0.20	0.25	100	4.441	3.544
0.04	0.08	0.20	0.25	110	11.662	0.798
0.04	0.08	0.20	0.25	120	20.898	0.118
0.04	0.12	0.20	0.25	80	0.052	20.000
0.04	0.12	0.20	0.25	90	0.841	10.161
0.04	0.12	0.20	0.25	100	4.397	3.525
0.04	0.12	0.20	0.25	110	11.547	0.794
0.04	0.12	0.20	0.25	120	20.694	0.118
0.04	0.08	0.40	0.25	80	1.289	20.528
0.04	0.08	0.40	0.25	90	3.823	12.927
0.04	0.08	0.40	0.25	100	8.350	7.456
0.04	0.08	0.40	0.25	110	14.797	3.958
0.04	0.08	0.40	0.25	120	22.716	1.954
0.04	0.08	0.20	0.50	80	0.414	20.000
0.04	0.08	0.20	0.50	90	2.180	10.706
0.04	0.08	0.20	0.50	100	6.496	4.772
0.04	0.08	0.20	0.50	110	13.425	1.760
0.04	0.08	0.20	0.50	120	22.060	0.546

Table III

b	r	sigma	tau	S0	Call	Put
0.00	0.08	0.20	0.25	80	0.040	20.000
0.00	0.08	0.20	0.25	90	0.702	10.578

0.00	0.08	0.20	0.25	100	3.927	3.927
0.00	0.08	0.20	0.25	110	10.811	0.940
0.00	0.08	0.20	0.25	120	20.016	0.146
0.00	0.12	0.20	0.25	80	0.040	20.000
0.00	0.12	0.20	0.25	90	0.698	10.526
0.00	0.12	0.20	0.25	100	3.900	3.900
0.00	0.12	0.20	0.25	110	10.754	0.934
0.00	0.12	0.20	0.25	120	20.001	0.145
0.00	0.08	0.40	0.25	80	1.169	20.927
0.00	0.08	0.40	0.25	90	3.534	13.394
0.00	0.08	0.40	0.25	100	7.844	7.844
0.00	0.08	0.40	0.25	110	14.085	4.226
0.00	0.08	0.40	0.25	120	21.856	2.116
0.00	0.08	0.20	0.50	80	0.303	20.040
0.00	0.08	0.20	0.50	90	1.723	11.481
0.00	0.08	0.20	0.50	100	5.478	5.478
0.00	0.08	0.20	0.50	110	11.902	2.149
0.00	0.08	0.20	0.50	120	20.335	0.703

Table IV

b	r	sigma	tau	S0	Call	Put
0.08	0.08	0.20	0.25	80	.	20.000
0.08	0.08	0.20	0.25	90	.	10.013
0.08	0.08	0.20	0.25	100	.	3.220
0.08	0.08	0.20	0.25	110	.	0.681
0.08	0.08	0.20	0.25	120	.	0.097
0.12	0.12	0.20	0.25	80	.	20.000
0.12	0.12	0.20	0.25	90	.	10.000
0.12	0.12	0.20	0.25	100	.	2.925
0.12	0.12	0.20	0.25	110	.	0.578
0.12	0.12	0.20	0.25	120	.	0.079
0.08	0.08	0.40	0.25	80	.	20.248
0.08	0.08	0.40	0.25	90	.	12.514
0.08	0.08	0.40	0.25	100	.	7.100
0.08	0.08	0.40	0.25	110	.	3.712
0.08	0.08	0.40	0.25	120	.	1.807
0.08	0.08	0.20	0.50	80	.	20.000
0.08	0.08	0.20	0.50	90	.	10.235
0.08	0.08	0.20	0.50	100	.	4.193
0.08	0.08	0.20	0.50	110	.	1.446
0.08	0.08	0.20	0.50	120	.	0.424

Table V

b	r	sigma	tau	S0	Call	Put
-0.04	0.08	0.20	3.00	80	2.524	26.245
-0.04	0.08	0.20	3.00	90	4.966	20.641
-0.04	0.08	0.20	3.00	100	8.670	15.990
-0.04	0.08	0.20	3.00	110	13.882	12.221
-0.04	0.08	0.20	3.00	120	20.882	9.235
0.00	0.08	0.20	3.00	80	4.203	22.395
0.00	0.08	0.20	3.00	90	7.537	16.498
0.00	0.08	0.20	3.00	100	12.030	12.030
0.00	0.08	0.20	3.00	110	17.641	8.687
0.00	0.08	0.20	3.00	120	24.297	6.222
0.04	0.08	0.20	3.00	80	6.965	20.325
0.04	0.08	0.20	3.00	90	11.621	13.563

```
0.04    0.08    0.20    3.00    100        17.399         9.108
0.04    0.08    0.20    3.00    110        24.092         6.122
0.04    0.08    0.20    3.00    120        31.491         4.115
0.08    0.08    0.20    3.00     80             .        20.000
0.08    0.08    0.20    3.00     90             .        11.634
0.08    0.08    0.20    3.00    100             .         6.962
0.08    0.08    0.20    3.00    110             .         4.257
0.08    0.08    0.20    3.00    120             .         2.640

Temps de calcul : 1.150 secondes
```

Nous avons effectué les calculs sur un Pentium 100. 1.15 secondes suffisent pour exécuter le programme précédent. Si nous éliminons les commandes d'affichage des résultats, il faut une demi seconde pour calculer 200 prix d'options américaines !

5.2 Les modèles à saut de valorisation des options de change

5.2.1 Le modèle à saut pur de Borensztein et Dooley [1987]

Références

Borensztein, E.R. et M.P. Dooley [1987], Options on foreign exchange and exchange rate expectations, *IMF Staff Papers,* **34**, pages 643-680

Dans le modèle à saut pur de valorisation des options de change, nous considèrons que la dynamique du sous-jacent $S(t)$ est

$$\begin{cases} dS(t) &= \mu S(t)\, dt + (\phi - 1)\, S(t)\, dN(t) \\ S(t_0) &= S_0 \end{cases}$$

avec $N(t)$ un processus de Poisson d'intensité λ. Borensztein et Dooley [1987] montrent alors que la prime d'une option de change européenne d'achat de prix d'exercice K est donnée par la formule suivante

$$C_{\text{EU}}(S_0) = S_0 e^{-r^\star \tau} \Psi(d_1, n^\star) - K e^{-r\tau} \Psi(d_2, n^\star)$$

avec

$$n^\star = \left[\frac{\ln \frac{K}{S_0} - \mu\tau}{\ln \phi} \right]$$

$$d_1 = \frac{\phi}{\phi - 1} (r - r^\star - \mu)\, \tau$$

et

$$d_2 = \frac{1}{\phi - 1} (r - r^\star - \mu)\, \tau$$

La fonction $\Psi(d, n^\star)$ est définie de la façon suivante

$$\Psi(d, n^\star) = \sum_{n=n^\star}^{\infty} e^{-d} \frac{d^n}{n!}$$

Borensztein_Dooley

■ **Objectif**

Valorisation d'une option européenne de change (modèle de Borensztein et Dooley [1987]).

■ **Format**

C = Borensztein_Dooley(S0,K,mu,tau,phi,r,rstar);

■ **Entrées**

S0	vecteur $N \times 1$, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
mu	vecteur $N \times 1$, paramètre μ.
tau	vecteur $N \times 1$, maturité de l'option.
phi	vecteur $N \times 1$, paramètre ϕ.
r	vecteur $N \times 1$, taux d'intérêt national.
rstar	vecteur $N \times 1$, taux d'intérêt étranger.

■ **Sorties**

C vecteur $N \times 1$, valeur de l'option européenne d'achat.

■ **Remarque**

La variable externe _dooley_Max permet d'approcher la fonction $\Psi(d, n^\star)$ par la formule $\sum_{n=n^\star}^{n+\_\text{dooley\_Max}} e^{-d}\frac{d^n}{n!}$.

dooley.dec

```
declare matrix _dooley_Max = 5;
```

dooley.ext

```
external matrix _dooley_Max;
```

dooley.src

```
proc _dooley_PSI(d,nstar);
  local n;

  nstar = nstar'; d = d';
  n = nstar + seqa(0,1,_dooley_Max);

  retp( sumc(exp(-d).*(d.^n)./(n!)) );
endp;

proc Borensztein_Dooley(S0,K,mu,tau,phi,r,rstar);
  local lambda,nstar,d2,d1,C;

  lambda = (r-rstar-mu)./(phi-1);
  nstar = ceil( (ln(K./S0)-mu.*tau)./ln(phi) );
  nstar = nstar.*(nstar .>= 0);
  d2 = lambda.*tau;
  d1 = phi.*d2;

  C = S0.*exp(-rstar.*tau).*_dooley_PSI(d1,nstar)
      - K.*exp(-r.*tau).*_dooley_PSI(d2,nstar);

  retp(C);
endp;
```

5.2.2 Le modèle de Merton [1976]

Références

Garman, M.B. et S.W. Kohlhagen [1983], Foreign currency option values, *Journal of International Money and Finance*, **2**, pages 231-237

Lamberton, D. et B. Lapeyre [1991], Introduction au Calcul Stochastique Appliqué à la Finance, Ellipses

Merton, R. [1976], Option pricing when underlying stock returns are discontinuous, *Journal of Financial Economics*, **3**, pages 125-144

Lamberton et Lapeyre [1991] analyse la valorisation d'une option européenne lorsque l'actif sous-jacent est décrit par un processus de saut. Ce modèle est très proche de celui de Merton [1976]. Considérons la dynamique suivante du cours de change

$$\left\{ \begin{array}{rcl} dS(t) & = & \mu S(t)\, dt + \sigma S(t)\, dW(t) + (\theta - 1) S(t)\, dN(t) \\ S(t_0) & = & S_0 \end{array} \right.$$

avec $W(t)$ un processus de Wiener et $N(t)$ un processus de Poisson d'intensité λ. Nous pouvons alors montrer que la valeur de la prime de l'option d'achat est

$$C_{\text{EU}}(S_0) = \sum_{n=0}^{\infty} e^{-\lambda\tau}\frac{(\lambda\tau)^n}{n!}\,\text{GK}\left(S_0 e^{-\lambda(\theta-1)\tau}\theta^n, K, \sigma, \tau, r, r^\star\right)$$

où GK est la formule de Garman et Kohlhagen [1983]

$$\mathrm{GK}\left(S_0, K, \sigma, \tau, r, r^\star\right) = S_0 e^{-r^\star \tau} \Phi\left(d_1\right) - K e^{-r\tau} \Phi\left(d_2\right)$$

avec

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r - r^\star\right)\tau}{\sigma\sqrt{\tau}} + \frac{1}{2}\sigma\sqrt{\tau}$$

et

$$d_2 = d_1 - \sigma\sqrt{\tau}$$

Le modèle de Garman et Kohlhagen [1983] est donc un cas particulier du modèle de Merton [1976]. En effet, pour $\lambda = 0$ et $\theta = 1$ (absence de saut), nous vérifions bien que

$$C_{\mathrm{EU}}\left(S_0\right) = \mathrm{GK}\left(S_0, K, \sigma, \tau, r, r^\star\right)$$

Garman_Kohlhagen

■ **Objectif**
Valorisation d'une option européenne de change (modèle de Garman et Kohlhagen [1983]).

■ **Format**
C = Garman_Kohlhagen(S0,K,sigma,tau,r,rstar);

■ **Entrées**

S0	vecteur $N \times 1$, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
sigma	vecteur $N \times 1$, volatilité.
tau	vecteur $N \times 1$, maturité de l'option.
r	vecteur $N \times 1$, taux d'intérêt national.
rstar	vecteur $N \times 1$, taux d'intérêt étranger.

■ **Sorties**

C	vecteur $N \times 1$, valeur de l'option européenne d'achat.

Merton

■ **Objectif**
Valorisation d'une option européenne de change (modèle de Merton [1976]).

■ **Format**
C = Merton(S0,K,sigma,tau,r,rstar,theta,lambda,Nstar);

■ **Entrées**

S0	vecteur $N \times 1$, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
sigma	vecteur $N \times 1$, volatilité.
tau	vecteur $N \times 1$, maturité de l'option.
r	vecteur $N \times 1$, taux d'intérêt national.
rstar	vecteur $N \times 1$, taux d'intérêt étranger.
theta	vecteur $N \times 1$, paramètre θ.
lambda	vecteur $N \times 1$, paramètre λ.
Nstar	scalaire, Nombre de sauts considérés pour évaluer la prime de l'option.

■ **Sorties**

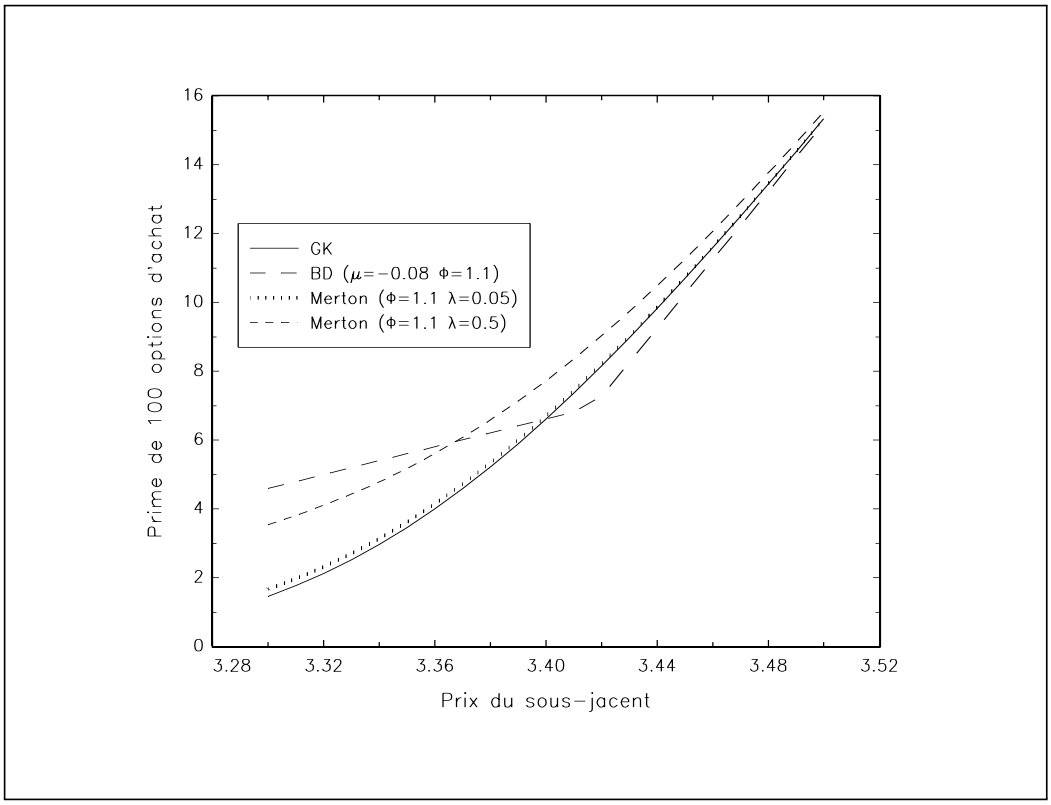
C	vecteur $N \times 1$, valeur de l'option européenne d'achat.

merton.src

```
proc Garman_Kohlhagen(S0,K,sigma,tau,r,rstar);
  local w,d1,d2,C;

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

  retp(C);
endp;

proc Merton(S0,K,sigma,tau,r,rstar,theta,lambda,Nstar);
  local n,Stilde,prob,C;

  if Nstar == 0;
    Nstar = 10;
  endif;

  n = seqa(0,1,Nstar); n = n';

  Stilde = S0.*exp(-lambda.*(theta-1).*tau).*(theta^n);
  prob = exp(-lambda.*tau).*((lambda.*tau)^n)./(n!);

  C = prob.*Garman_Kohlhagen(Stilde,K,sigma,tau,r,rstar);
  C = sumc(C');

  retp(C);
endp;
```

Dans le programme suivant, nous comparons la valorisation d'une option de change en fonction du modèle (Garman et Kohlhagen [1983], Borensztein et Dooley [1987] et Merton [1976]). Il est intéressant de remarquer que la prise en compte d'un saut pur par rapport au modèle de Garman et Kohlhagen [1983] influence différement les primes des options dans la monnaie et celles des options en dehors de la monnaie. Borensztein et Dooley [1987] ont constaté que le modèle de Garman et Kohlhagen [1983] avait tendance à sous-évaluer les prix des options d'achat en dehors de la monnaie. Ils ont utilisé le modèle à saut pur, non pas comme un modèle de valorisation, mais comme un outil permettant de révéler les anticipations de change, c'est-à-dire un outil de lecture d'information.

```
new;
library finance,pgraph;

r = 3.50/100;       /* Taux d'interet national */
rstar = 3.10/100;   /* Taux d'interet etranger */
K = 3.35;           /* Prix d'exercice         */
sigma = 0.05;       /* Volatilite              */
tau = 90/365;       /* Maturite 90 jours       */

S0 = seqa(3.30,0.01,21);

GK = 100*Garman_Kohlhagen(S0,K,sigma,tau,r,rstar);
M1 = 100*Merton(S0,K,sigma,tau,r,rstar,1.1,0.05,10);
M2 = 100*Merton(S0,K,sigma,tau,r,rstar,1.1,0.5,10);
BD = 100*Borensztein_Dooley(S0,K,-0.08,tau,1.1,r,rstar);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _plwidth = 0|0|10|0;
  ylabel("Prime de 100 options d'achat");
  xlabel("Prix du sous-jacent");
  _plegstr = "GK\0BD (\202m\201=-0.08 \202F\201=1.1)"\
     "\0Merton (\202F\201=1.1 \202l\201=0.05)"\
```



Graphique 23

```
   "\0Merton (\202F\201=1.1 \202l\201=0.5)";
_plegctl = {2 5 1.25 4};
graphprt("-c=1 -cf=merton.eps -w=5");
xy(S0,GK~BD~M1~M2);
```

6 Lecture de l'information sur les marchés dérivés

Références

Banque De France [1995], Implications macro-économiques et de politique monétaire du développement des marchés dérivés, *Bulletin de la Banque de France*, **14**, 89-129

6.1 Les options sur future

Références

Black, F. [1976], The pricing of commodity contracts, *Journal of Financial Economics*, **3**, 167-179

Nous notons K le prix d'exercice de l'option, F_0 la valeur du future, σ la volatilité, τ la maturite et r le taux d'intérêt. Dans le modèle de Black [1976], le prix de l'option d'achat est donné par la formule suivante

$$C = F_0 e^{-r\tau}\Phi\left(d_1\right) - Ke^{-r\tau}\Phi\left(d_2\right)$$

avec Φ le fonction de répartition de la loi normale centrée et réduite et

$$\begin{aligned} d_1 &= \frac{1}{\sigma\sqrt{\tau}}\ln\frac{F_0}{K} + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= \frac{1}{\sigma\sqrt{\tau}}\ln\frac{F_0}{K} - \frac{1}{2}\sigma\sqrt{\tau} \end{aligned}$$

Pour une option de vente, nous avons

$$P = -F_0 e^{-r\tau}\Phi\left(-d_1\right) + Ke^{-r\tau}\Phi\left(-d_2\right)$$

Black

■ **Objectif**
Valorisation d'une option européenne sur future (modèle de Black [1976]).

■ **Format**
Prime = Black(F0,K,sigma,tau,r);

■ **Entrées**

F0	vecteur $N \times 1$, prix du future.
K	vecteur $N \times 1$, prix d'exercice.
sigma	vecteur $N \times 1$, volatilité.
tau	vecteur $N \times 1$, maturité de l'option.
r	vecteur $N \times 1$, taux d'intérêt.

■ **Sorties**

Prime	vecteur $N \times 1$, valeur de l'option européenne d'achat.

■ **Remarque**
Nous précisons le type de l'option avec la variable externe `_type_option` (`''call''` ou `''put''`).

black.src

```
proc Black(F0,K,sigma,tau,r);
  local w,d1,d2,V;

  w = sigma.*sqrt(tau);

  d1 = ln(F0./K)./w + 0.5*w;
  d2 = d1 - w;

  if _type_option $== "put";
    V = -F0.*exp(-r.*tau).*cdfn(-d1)+K.*exp(-r.*tau).*cdfn(-d2);
  else;
    V = F0.*exp(-r.*tau).*cdfn(d1)-K.*exp(-r.*tau).*cdfn(d2);
  endif;

  retp(V);
endp;
```

6.1.1 Volatilité implicite des options américaines

Références
Gamas, M-N. [1997], Capacité informationnelle de la volatilité implicite : application au marché monétaire français, XIV$^{\text{è}}$ journées internationales d'économie monétaire et bancaire, Orléans, 5 et 6 juin 1997

Soient K le prix d'exercice de l'option, F_0 la valeur du future, σ la volatilité, τ la maturite et r le taux d'intérêt. Dans le modèle de valorisation d'une option américaine sur future, le prix **théorique** de l'obligation est fonction de ces 5 paramètres. Ce prix dépend donc de 4 paramètres parfaitement objectif (K et τ sont définis par les termes du contrat, et F_0 et r sont des prix observés sur les marchés au comptant) et d'un paramètre *subjectif* σ. Soit $P^\bullet$ la valeur **observée** de l'option sur le marché. La volatilité implicite correspond alors à la valeur de σ telle que le prix théorique soit égal au prix observé :

$$P\left(F_0, K, \sigma_{\text{imp}}, \tau, r\right) = P^\bullet$$

Pour résoudre cette équation, nous pouvons par exemple employer l'algorithme de la bisection. Dans ce cas, nous devons préciser a et b tels que nous avons

$$P\left(F_0, K, a, \tau, r\right) - P^\bullet \leq 0$$

et

$$P\left(F_0, K, b, \tau, r\right) - P^\bullet \geq 0$$

volimp_Future

■ **Objectif**
Calcul de la volatilité implicite d'une option américaine d'achat sur future.

■ **Format**
sigma = volimp_Future(F0,K,tau,r,Prime,a,b);

■ **Entrées**

F0	vecteur $N \times 1$, prix du future.
K	vecteur $N \times 1$, prix d'exercice.
tau	vecteur $N \times 1$, maturité de l'option.
r	vecteur $N \times 1$, taux d'intérêt.
Prime	vecteur $N \times 1$, prime de l'option.
a	vecteur $N \times 1$, valeur de a.
b	vecteur $N \times 1$, valeur de b.

■ **Sorties**

sigma	vecteur $N \times 1$, volatilité implicite.

■ **Remarque**
Nous précisons le type de l'option avec la variable externe `_type_option` (`''call''` ou `''put''`).

volimp.dec

```
declare matrix _volimp_K;
declare matrix _volimp_F0;
declare matrix _volimp_tau;
declare matrix _volimp_r;
declare matrix _volimp_Prime;
```

volimp.ext

```
external matrix _volimp_K;
external matrix _volimp_F0;
external matrix _volimp_tau;
external matrix _volimp_r;
external matrix _volimp_Prime;
```

volimp.src

```
proc _volimp_Future(sigma);

   retp(American_BS(_volimp_F0,_volimp_K,sigma,_volimp_tau,0,_volimp_r)
        - _volimp_Prime);
endp;


proc volimp_Future(F0,K,tau,r,Prime,a,b);
  _volimp_Prime = Prime;
  _volimp_K = K;
  _volimp_F0 = F0;
  _volimp_tau = tau;
  _volimp_r = r;

  retp( bisection(&_volimp_Future,a,b) );
endp;
```

Dans le programme suivant, nous estimons la volatilité implicite des différentes options d'achat du 02/01/1992 d'échéance Mars 1992 sur le contrat à terme PIBOR.

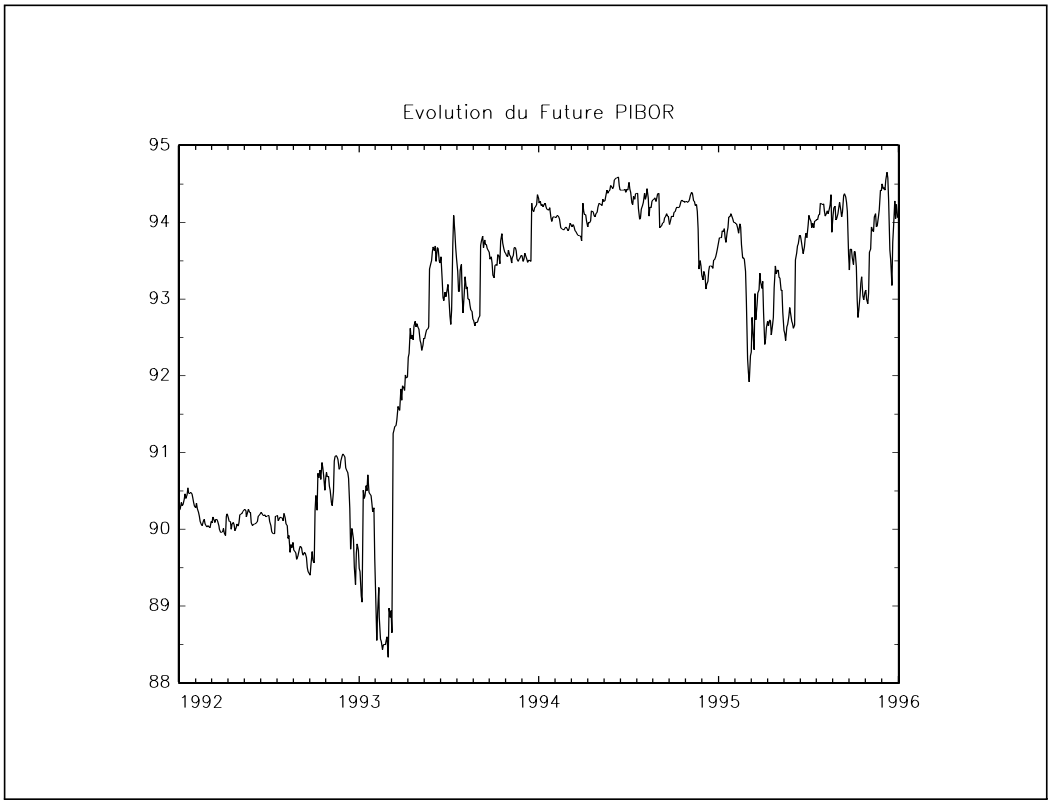
```
new;
library finance;
```

```
F0 = 100;
K = 90|100;
tau = 60/360;
r = 0.10;

/* Options d'achat du 02/01/1992 */

let Option[9,5] =
/*
Echeance       Prime           Taux d'interet
   Prix d'Exercice   Prix du Future
*/
9203    90.3    0.15    90.25   .101875
9203    90.4    0.11    90.25   .101875
9203    90.5    0.075   90.25   .101875
9203    90.7    0.035   90.25   .101875
9203    91.1    0.01    90.25   .101875
9206    90.4    0.37    90.59   .101875
9206    90.5    0.31    90.59   .101875
9206    91.5    0.025   90.59   .101875
9209    91.2    0.20    90.95   .101875;

K = Option[.,2];
C = Option[.,3];
F0 = Option[.,4];
r = Option[.,5];
tau = ( 75 + 30*(Option[.,1]%9203) )/360;

Vimp = volimp_Future(F0,K,tau,r,C,0.005*ones(9,1),0.50*ones(9,1));

output file = volimp.out reset;

print "        Option N.     Volat-imp.";
print "---------------------------------------";

call printfmt(seqa(1,1,9)~Vimp,1~1);

output off;
```

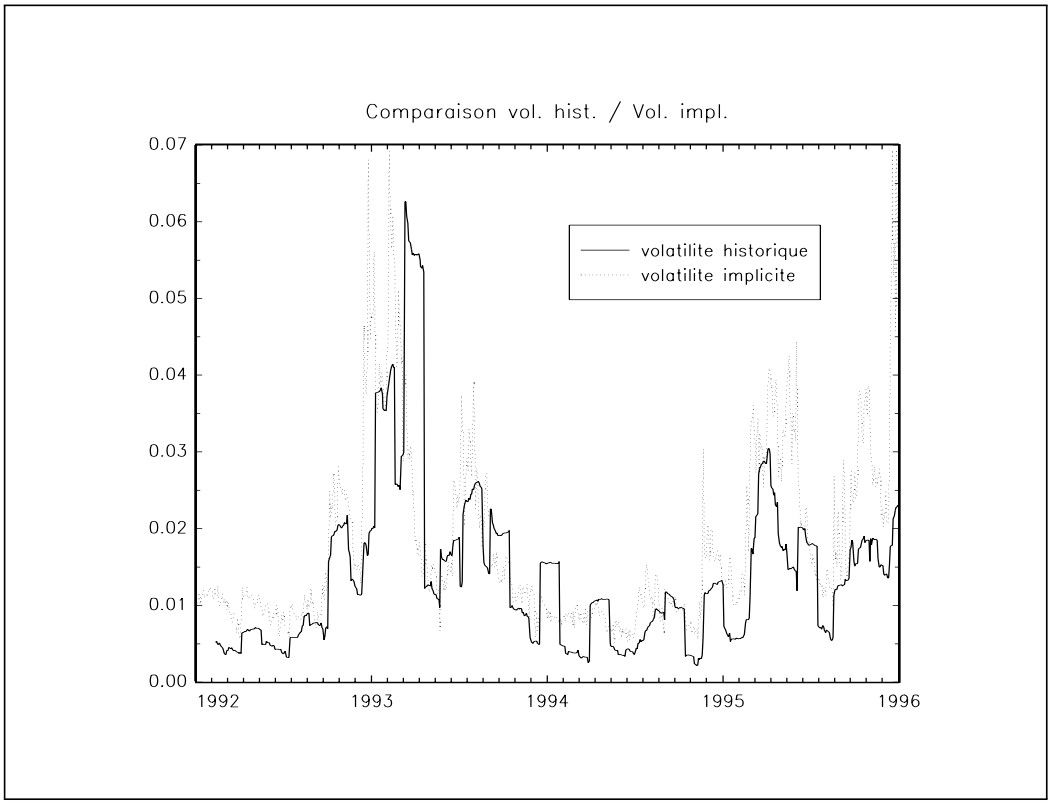
```
        Option N.     Volat-imp.
---------------------------------------
               1      0.010729027
               2      0.010751686
               3      0.010532646
               4      0.010698814
               5      0.012322731
               6      0.011318169
               7      0.011159554
               8      0.010744133
               9      0.010449562
```

L'évolution de cette volatilité implicite (comparée à celle de la volatilité historique) permet d'appréhender les anticipations des agents sur la variabilité du sous-jacent. Elle permet donc d'extraire une information, qui n'est pas prise en compte sur le marché au comptant. Les graphiques (24) et (25) sont tirés de l'étude de Gamas [1997]. Il est clair que la volatilité implicite "devance" la volatilité historique.

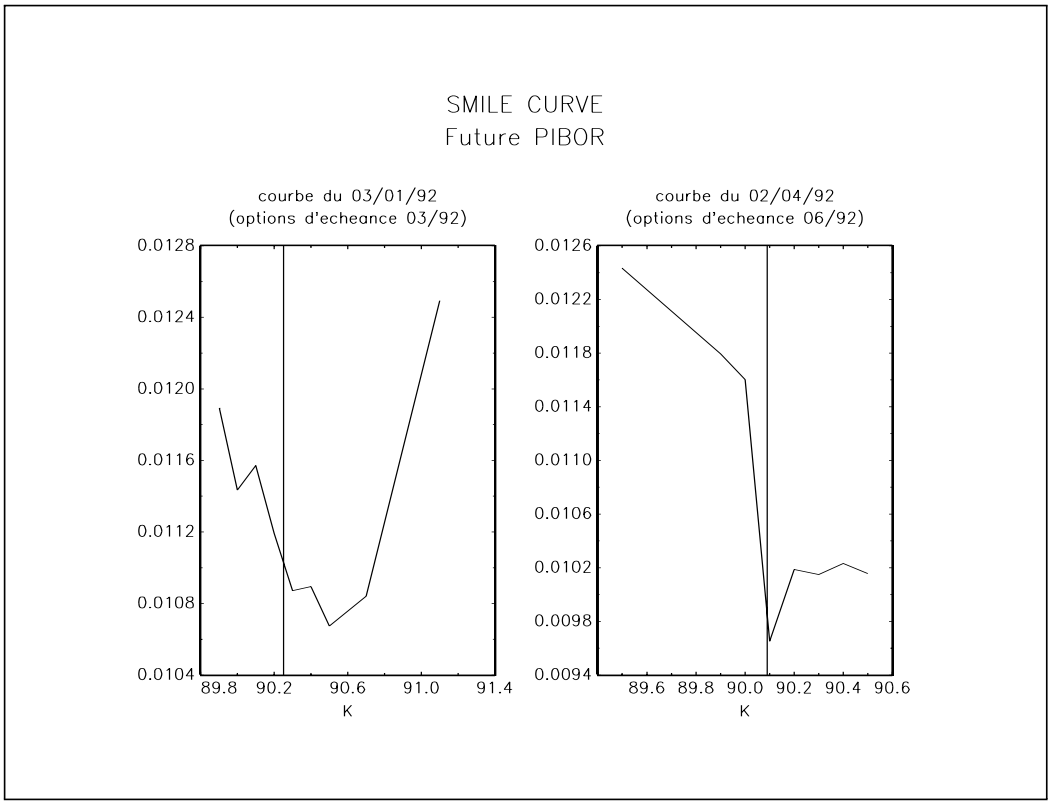
La volatilité implicite permet aussi de construire les courbes de *volatility smile*. Si nous représentons la volatilité implicite en fonction du prix d'exercice K pour les options uniquement hors de la monnaie (donc pour $K \leq F_0$, nous considérons les volatilités implicites des options de vente et pour $K > F_0$, nous considérons les volatilités implicites des options d'achat), nous devons observé "le sourire de la volatilité" pour les périodes "normales". Le graphique (26) représente la *smile curve* pour les options sur contrat à terme PIBOR pour deux dates différentes.



Graphique 24



Graphique 25



Graphique 26

6.1.2 La prime de Skewness

La relation de parité CALL-PUT dans le modèle de Black [1976] est

$$C - P = e^{-r\tau} F_0 - e^{-r\tau} K$$

Si nous considérons des options à la monnaie ($K = F_0$), nous vérifions alors l'égalité suivante

$$\frac{C}{P} - 1 = 0$$

Notons SK la prime de skewness. Celle-ci est définie de la façon suivante

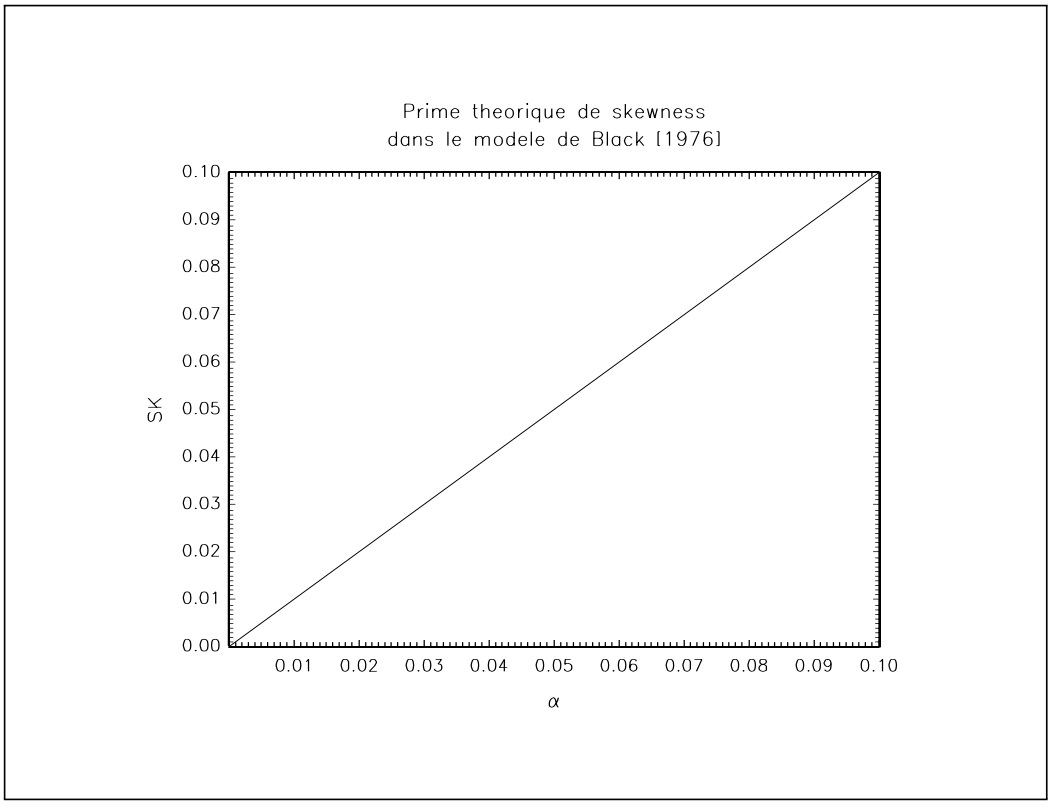
$$SK = \frac{C\left(F_0, K_c, \sigma, \tau, r\right)}{C\left(F_0, K_p, \sigma, \tau, r\right)} - 1$$

Cela veut dire que l'option d'achat et l'option de vente ont les mêmes caractéristiques à l'exception du prix d'exercice. Nous devons vérifier que cette prime de skewness est proche de 0 si nous considérons des options d'achat et de vente qui sont symétriquement hors de la monnaie, c'est-à-dire des options pour lesquelles les prix d'exercice sont $K_c = (1 + \alpha) F_0$ et $K_p = \frac{1}{1+\alpha} F_0$ avec α un scalaire positif relativement petit. En fait, nous pouvons montrer que

$$SK = \frac{C\left(F_0, K_c, \sigma, \tau, r\right)}{C\left(F_0, K_p, \sigma, \tau, r\right)} - 1 \simeq \alpha$$

Ce résultat peut s'étendre aux cas des options américaines. Le graphique (27) illustre ce résultat dans le cas des options européennes.

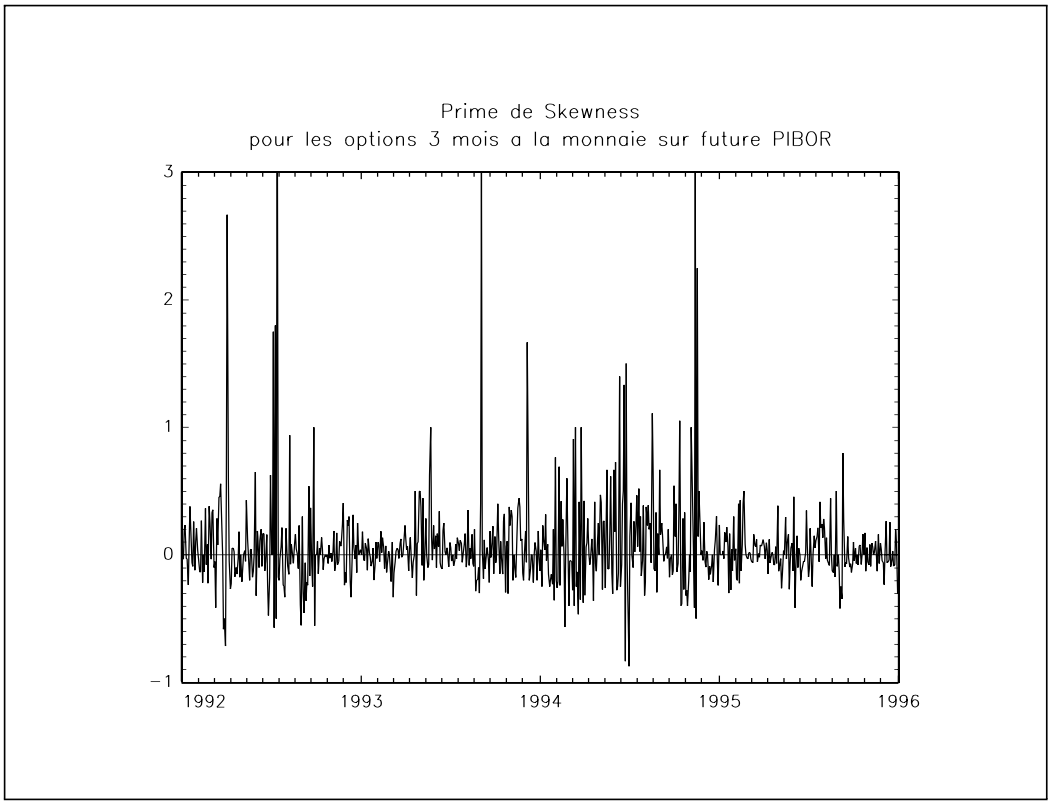
```
new;
library finance,pgraph;

F0 =  100;
sigma = 0.15;
tau = 90/360;
```



Graphique 27

```
r = 0.10;

alpha = seqa(0,1/100,11);

_type_option = "call";
K = F0*(1+alpha);
Prix_Call = Black(F0,K,sigma,tau,r);

_type_option = "put";
K = F0./(1+alpha);
Prix_Put = Black(F0,K,sigma,tau,r);

Skewness = (Prix_Call./Prix_Put) - 1;

graphset;
  _pdate = ""; _pnum = 2;
  fonts("simplex simgrma");
  title("Prime theorique de skewness"\
       "\Ldans le modele de Black @[1976@]");
  xlabel("\202a\201"); ylabel("SK");
  xtics(0,0.10,0.01,10); ytics(0,0.10,0.01,10);
  graphprt("-c=1 -cf=skew1.eps -w=5");
  xy(alpha,Skewness);
```

Comment interpréter la prime de skewness ? Si cette prime diffère très fortement de 0 pour les options à la monnaie, alors nous pouvons penser que le modèle de valorisation n'est pas valable, puisqu'il y a possibilité d'arbitrage (la relation de parité CALL-PUT n'est pas vérifiée). Le graphique (28) représente l'évolution de cette prime pour les options 3 mois sur contrat à terme PIBOR. Nous remarquons en particulier que cette prime est proche de 0 pour l'année 1995. Si cette prime est négative pour les options hors de la monnaie, on peut penser que les agents anticipent une baisse du prix du sous-jacent du fait de la relation entre la prime et le pay-off de l'option. Nous remarquons sur le graphique (29) que la prime de skweness est systématiquement négative pour les périodes



Graphique 28

février-juin 1995 et septembre-décembre 1995. On peut donc penser que les agents anticipaient une remontée des taux.

6.1.3 Estimation de la mesure de probabilité neutre au risque

Références

Breeden, D. et R. Litzenberger [1978], State contingent prices implicit in option prices, *Journal of Business*, **51**, 621-651

Söderlind, P. et L.E.O. Svensson [1997], New techniques to extract market expectations from financial instruments, *NBER*, **5877**

Sous la mesure de probabilité neutre au risque $\mathbb{P}'$, le prix d'une option d'achat sur future est égal à l'espérance de gain actualisée, c'est-à-dire que nous avons

$$C(K) = e^{-r\tau} E'\left[(F(T) - K)_+\right]$$

Nous pouvons alors en déduire que

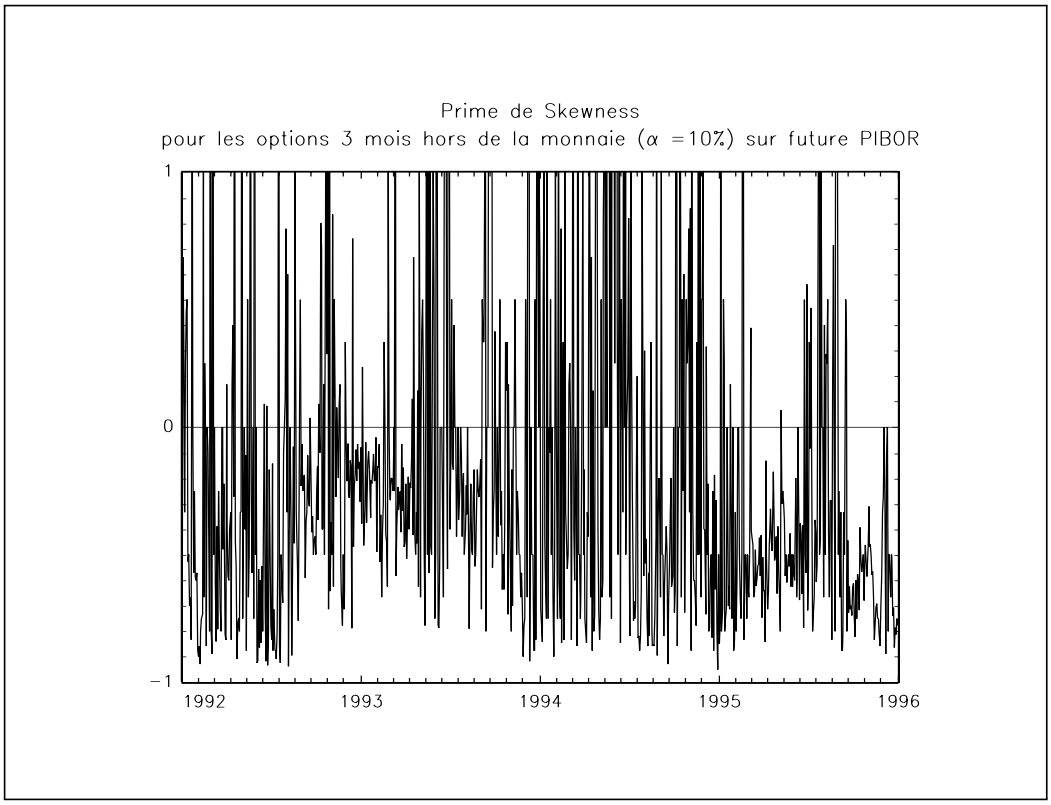
$$\Pr\{F(T) \leq K\} = 1 + e^{r\tau} \frac{\partial C(K)}{\partial K} \tag{25}$$

Ce résultat est indépendant du modèle considéré. Nous pouvons notamment le vérifier avec le modèle de Black [1976]. Dans ce cas, nous savons que

$$\Pr\{F(T) \leq K\} = \Phi\left(\frac{\ln K - \ln F_0 + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}}\right)$$

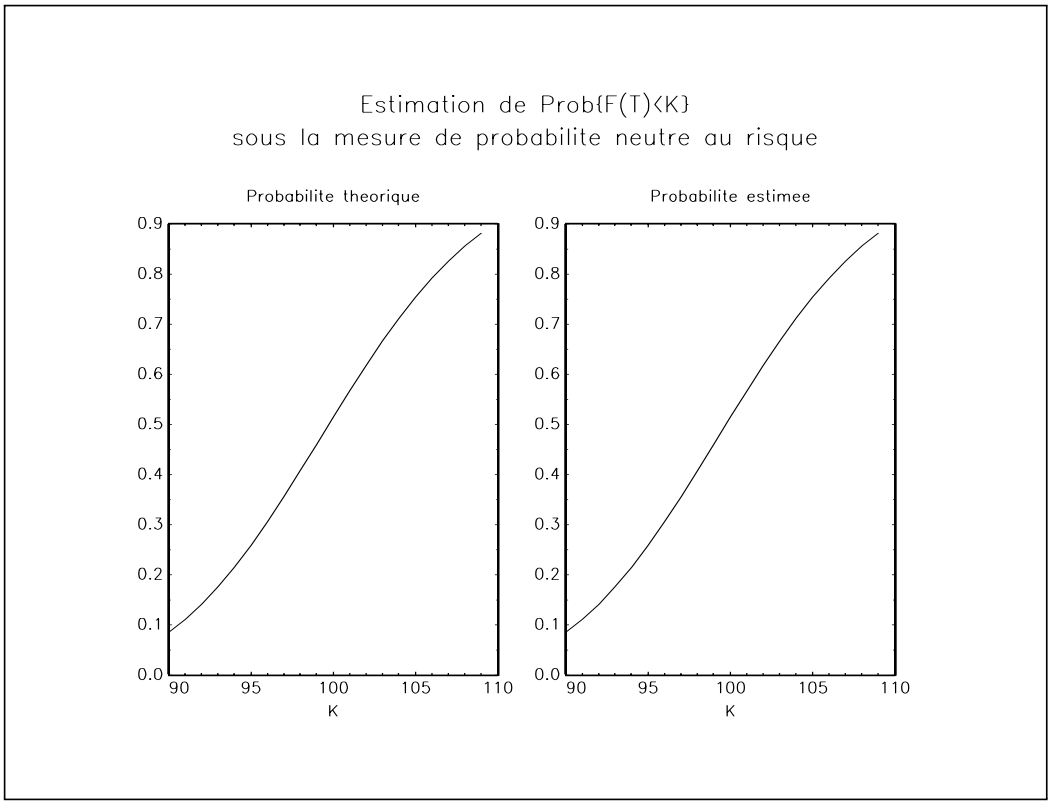
Sur le graphique (30), nous représentons la probabilité théorique $\Pr\{F(T) \leq K\}$ et la probabilité estimée à partir de la dérivée $\frac{\partial C(K)}{\partial K}$. Nous remarquons qu'elles sont parfaitement égales. La formule (25) est intéressante, car elle permet de calculer la probabilité $\Pr\{F(T) \leq K\}$ à partir des données du marché. Cette information peut alors être considérée comme l'*opinion du marché*.

```
new;
```



Graphique 29

```
library tsm,optmum,finance,pgraph;

F0 =  100;
sigma = 0.15;
tau = 90/360;
r = 0.10;

K = seqa(90,1,20);

/* Probabilite theorique dans le modele de Black [1976] */

mu = ln(F0) - 0.5*sigma^2.*tau;
V = sigma^2.*tau;
x = (ln(K)-mu)./sqrt(V);
Probabilite_Theorique = cdfn(x);

/* Probabilite estimee a partir de la derivee premiere
   de la prime du call par rapport au prix d'exercice  */

proc _call(K);
  local C;
  _type_option = "call";
  C = Black(F0,K,sigma,tau,r);
  retp(C);
endp;

Probabilite_Estimee = 1 + exp(r*tau)*diag(gradp(&_call,K));


graphset;
```



Graphique 30

```
begwind;
window2(1,2,0,1);
setwind(1);

  _pdate = ""; _ptitlht = 1.25; _pnum = 0; _paxes = 0;
  title("Estimation de Prob{F(T)<K}"\
        "\Lsous la mesure de probabilite neutre au risque");
  draw;

setwind(2);
  graphset;
  _pnum = 2; _pnumht = 0.25; _paxht = 0.25; _ptitlht = 0.25;
  title("Probabilite theorique");
  xlabel("K");
  xtics(90,110,5,5);
  xy(K,Probabilite_Theorique);
setwind(3);
  title("Probabilite estimee");
  xy(K,Probabilite_Estimee);
  graphprt("-c=1 -cf=probimp1.eps -w=5");
endwind;
```

Comme nous avons

$$\Pr\left\{F\left(T\right)\leq K\right\}=1+e^{r\tau}\frac{\partial C\left(K\right)}{\partial K}=\int_{-\infty}^{K}x\,d\mathbb{P}^{\prime}\left(x\right)$$

alors nous pouvons montrer assez facilement que la fonction de densité, sous la mesure de probabilité neutre au risque, de $\ln F\left(T\right)$ correspond à

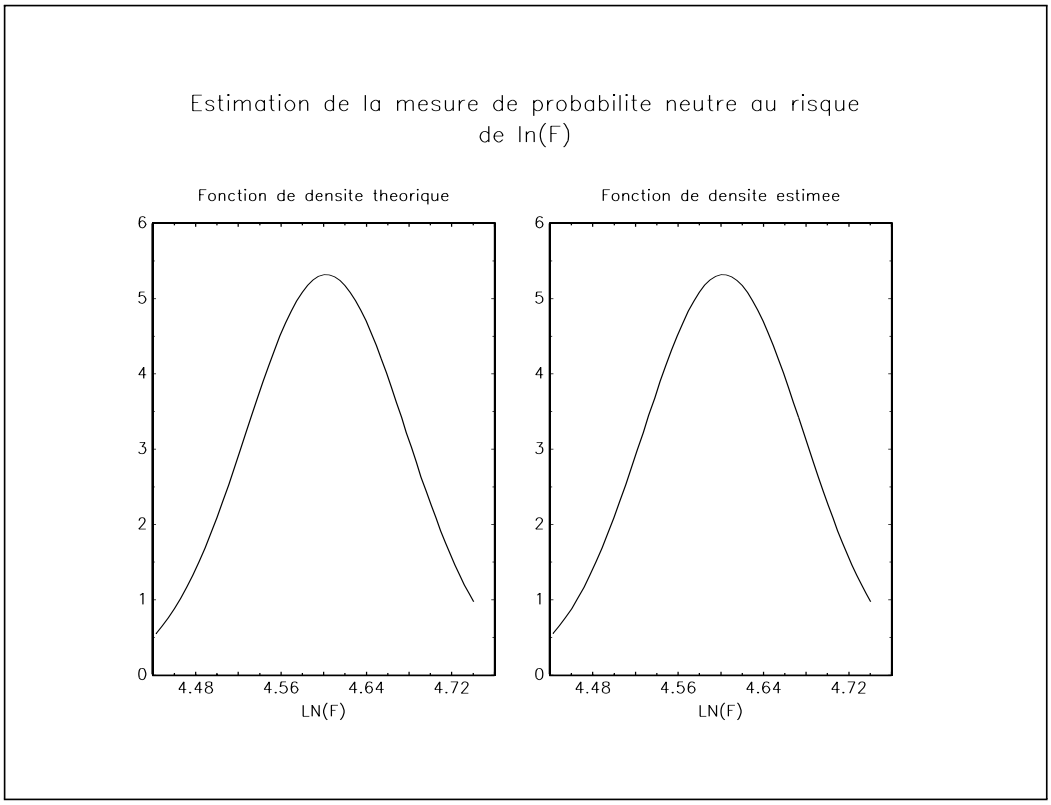
$$f\left(x\right)=e^{r\tau+x}\frac{\partial^{2}C\left(e^{x}\right)}{\partial K^{2}} \tag{26}$$

Ce résultat est aussi indépendant du modèle considéré. Pour le modèle de Black [1976], nous devons vérifier que

$$f(x) = \frac{1}{\sigma\sqrt{2\pi\tau}} \exp -\frac{1}{2}\left(\frac{x - \ln F_0 + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}}\right)^2$$

Dans le programme suivant, nous vérifions que la fonction de densité de $\ln F(T)$ dans le modèle de Black [1976] peut être estimée par la relation (26) (voir le graphique (31)).

```
new;
library tsm,optmum,finance,pgraph;

F0 = 100;
K = 110;
sigma = 0.15;
tau = 90/360;
r = 0.10;

Future = seqa(85,0.5,60);

/* Fonction de densite de ln(F) theorique dans le modele de Black [1976] */

mu = ln(F0) - 0.5*sigma^2.*tau;
V = sigma^2.*tau;
x = (ln(Future)-mu)./sqrt(V);
pdf_Theorique = pdfn(x)./sqrt(V);

/* Fonction de densite de ln(F) estimee a partir de la derivee seconde
   de la prime du call par rapport au prix d'exercice                  */

proc _call(K);
  local C;
  _type_option = "call";
  C = Black(F0,K,sigma,tau,r);
  retp(C);
endp;

pdf_Estimee = zeros(60,1);
i = 1;
do until i > 60;
  pdf_Estimee[i] = exp(r*tau+ln(Future[i]))*hessp(&_call,Future[i]);
  i = i + 1;
endo;


graphset;
  begwind;
  window2(1,2,0,1);
  setwind(1);

    _pdate = ""; _ptitlht = 1.2; _pnum = 0; _paxes = 0;
    title("Estimation de la mesure de probabilite neutre au risque\Lde ln(F)");
    draw;

  setwind(2);
    graphset;
    _pnum = 2; _pnumht = 0.25; _paxht = 0.25; _ptitlht = 0.25;
    title("Fonction de densite theorique");
    xlabel("LN(F)");
    xy(ln(Future),pdf_Theorique);
  setwind(3);
    title("Fonction de densite estimee");
```



Graphique 31

```
  xy(ln(Future),pdf_Estimee);
  graphprt("-c=1 -cf=probimp2.eps -w=5");
endwind;
```

Il existe plusieurs méthodes pour estimer la fonction de densité à partir la relation (26) en utilisant les données de marché. Si nous disposons d'un nombre relativement important de prix d'options (plus de 6), nous pouvons utiliser une méthode de spline cubique pour estimer la relation $C = g(K)$. Nous estimons alors la fonction de densité de $\ln F(T)$ à partir de la dérivée seconde de la spline cubique. Pour construire la fonction de répartition, nous employons l'algorithme des trapèzes. Nous avons implémenté cette méthode dans la procédure `Estimation_PDF_Future`.

Estimation_PDF_Future

■ **Objectif**

Estimation de la fonction de densité et de la fonction de répartition de $\ln F(T)$ sous la mesure de probabilité neutre au risque.

■ **Format**

{lnF,pdf,cdf} = Estimation_PDF_Future(Prime,K,tau,r,F);

■ **Entrées**

Prime	vecteur $N \times 1$, prix des options.
K	vecteur $N \times 1$, prix d'exercice.
tau	scalaire, maturité des options.
r	scalaire, taux d'intérêt.
F	vecteur $M \times 1$, valeurs prises par la variable aléatoire $F(T)$.

■ **Sorties**

lnF	vecteur $M \times 1$, valeur des abscisses.
pdf	vecteur $M \times 1$, fonction de densité.
cdf	vecteur $M \times 1$, fonction de répartition.

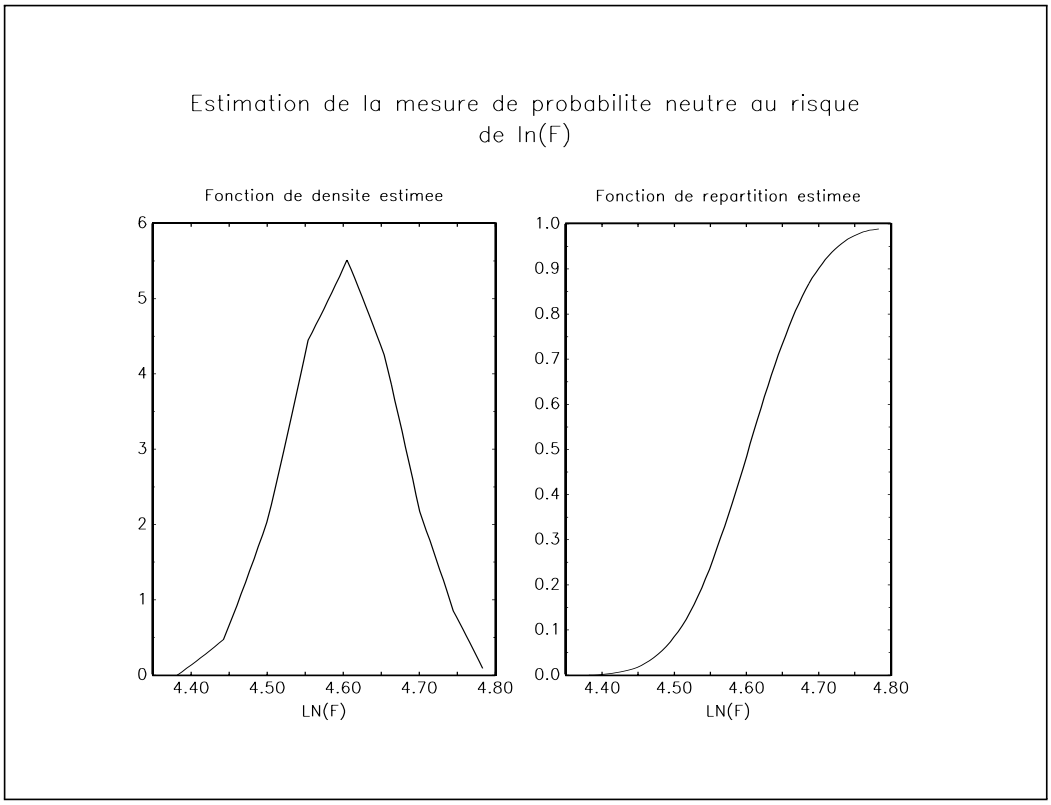
■ **Remarque**
La procédure utilise la bibliothèque du domaine public **SPLINE** de David Baird du Maf Technology of Lincoln.

probimp.src

```
proc (3) = Estimation_PDF_Future(Primes,K,r,tau,F);
  local data,Nobs_call,N,lnF,c,d,pdf,i,cdf;

  data = sortc(Primes~K,2);
  Primes = data[.,1]; K = data[.,2];
  F = sortc(F,1);

  Nobs_call = rows(Primes);
  N = rows(F);
  lnF = ln(F);

  /* Cubic spline which minimizes CV  */
  c = cvspline(K,Primes,ones(Nobs_call,1));

  /* second derivative of cubic spline */
  d = dspline(c,F,2);

  pdf = exp(r*tau+lnF).*d;

  cdf = zeros(N,1);
  i = 2;
  do until i > N;
    cdf[i] = cdf[i-1] + 0.5*(pdf[i]+pdf[i-1])*(lnF[i]-lnF[i-1]);
    i = i + 1;
  endo;

  retp(lnF,pdf,cdf);
endp;
```

Nous reprenons l'exemple précédent et nous estimons la fonction de densité à partir de 9 options d'achat. Vous pouvez comparer les graphiques (31) et (32).

```
new;
library tsm,pgraph,finance,pgraph;

/* Primes de 9 options d'achat */

K = seqa(80,5,9);
sigma = 0.15;
tau = 90/360;
r = 0.10;

_type_option = "call";
Primes = Black(100,K,sigma,tau,r);

F = seqa(80,0.5,80);

{lnF,pdf,cdf} = Estimation_PDF_Future(Primes,K,r,tau,F);

graphset;
  begwind;
  window2(1,2,0,1);
  setwind(1);

    _pdate = ""; _ptitlht = 1.2; _pnum = 0; _paxes = 0;
```



Graphique 32

```
  title("Estimation de la mesure de probabilite neutre au risque\Lde ln(F)");
  draw;

setwind(2);
  graphset;
  _pnum = 2; _pnumht = 0.25; _paxht = 0.25; _ptitlht = 0.25;
  title("Fonction de densite estimee");
  xlabel("LN(F)");
  xy(lnF,pdf);
setwind(3);
  title("Fonction de repartition estimee");
  xy(lnF,cdf);
  graphprt("-c=1 -cf=probimp3.eps -w=5");
endwind;
```

La qualité de l'estimation dépend évidemment du nombre d'options dont nous disposons. C'est pourquoi il est difficile d'estimer le fonction de densité si nous avons très peu d'options d'achat.

6.2 Les options de change

6.2.1 Estimation des probabilités de réalignement

Références
Ball, C et W. Torus [1985], On jumps in common stock prices and their impact on call option pricing, *Journal of Finance*, **40**, 155-173
Bates, D.S. [1991], The crash of '87: was it expected? The evidence from option markets, *Journal of Finance*, **46**, 1009-1044
Malz, A. M. [1996], Using option prices to estimate realignment probabilities in the European Monetary System: the case of sterling-mark, *Journal of International Money and Finance*, **15**, 717-774

S'inspirant des travaux de Ball et Torus [1985] et Bates [1991], Malz [1996] utilise un modèle à saut pour mesurer la crédibilité des parités officielles du MCE. En approximant la fonction de probabilité de Poisson par celle

de Bernoulli dans le modèle de Merton [1976], nous en déduisons que la valeur de la prime de l'option est

$$(1-\lambda\tau)\,\mathrm{GK}\left(\frac{S_0}{1+\lambda\,(\theta-1)\,\tau},K,\sigma,\tau,r,r^\star\right)+\lambda\tau\,\mathrm{GK}\left(\frac{\theta S_0}{1+\lambda\,(\theta-1)\,\tau},K,\sigma,\tau,r,r^\star\right)$$

où GK est la formule de Garman et Kohlhagen [1983]. λ est la probabilité d'occurence du saut pour la période de référence (en général, l'année) et $(\theta-1)$ est l'ampleur du saut.

_Malz

■ **Objectif**
Valorisation d'une option européenne de change (modèle de Malz [1996]).

■ **Format**
Prime = _Malz(S0,K,sigma,tau,r,rstar,theta,lambda,type_option);

■ **Entrées**

S0	vecteur $N \times 1$, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
sigma	vecteur $N \times 1$, volatilité.
tau	vecteur $N \times 1$, maturité de l'option.
r	vecteur $N \times 1$, taux d'intérêt national.
rstar	vecteur $N \times 1$, taux d'intérêt étranger.
theta	vecteur $N \times 1$, paramètre θ.
lambda	vecteur $N \times 1$, paramètre λ.
type_option	vecteur $N \times 1$, type de l'option (1 pour un call et 0 pour un put).

■ **Sorties**

Prime	vecteur $N \times 1$, prime de l'option européenne.

Malz [1996] utilise alors la formulation précédente pour estimer les coefficients σ, θ et λ. Le critère utilisé est la minimisation de la somme des carrés des erreurs entre la prime théorique et la prime observée pour différentes options. Avec la procédure `Malz`, l'utilisateur peut imposer des contraintes de la forme

$$C\begin{bmatrix}\sigma\\ \theta\\ \lambda\end{bmatrix}=c$$

Nous pouvons ainsi estimer dans un premier temps la volatilité implicite σ_{imp} en imposant le système auivant de constraintes

$$\begin{bmatrix}0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}\sigma\\ \theta\\ \lambda\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}$$

puis estimer les deux autres paramètres θ et λ en imposant la restriction

$$\begin{bmatrix}1&0&0\end{bmatrix}\begin{bmatrix}\sigma\\ \theta\\ \lambda\end{bmatrix}=\sigma_{\mathrm{imp}}$$

Ainsi, nous ne tenons compte que de la partie non expliquée par la volatilité implicite.

Malz

■ **Objectif**
Estimation des paramètres σ, θ et λ (modèle de Malz [1996]).

■ **Format**
{sigma,theta,lambda} = Malz(S0,K,tau,r,rstar,type_option,Prime,CC,c);

■ **Entrées**

S0	scalaire, prix de l'actif sous-jacent.
K	vecteur $N \times 1$, prix d'exercice.
tau	scalaire, maturité des options.
r	scalaire, taux d'intérêt national.
rstar	scalaire, taux d'intérêt étranger.
type_option	vecteur $N \times 1$, type des options (1 pour un call et 0 pour un put).
Prime	vecteur $N \times 1$, prime des options.
CC	matrice $g \times 3$, matrice C des restrictions.
c	vecteur $g \times 1$, vecteur c des restrictions.

■ **Sorties**

sigma	scalaire, estimation de σ.
theta	scalaire, estimation de θ.
lambda	scalaire, estimation de λ.

■ **Remarque**

`CC` prend la valeur 0 s'il n'a pas de restrictions.

malz.dec

```
declare matrix _malz_S0;
declare matrix _malz_K;
declare matrix _malz_tau;
declare matrix _malz_r;
declare matrix _malz_rstar;
declare matrix _malz_type_option;
declare matrix _malz_prime;
```

malz.ext

```
external matrix _malz_S0;
external matrix _malz_K;
external matrix _malz_tau;
external matrix _malz_r;
external matrix _malz_rstar;
external matrix _malz_type_option;
external matrix _malz_prime;
```

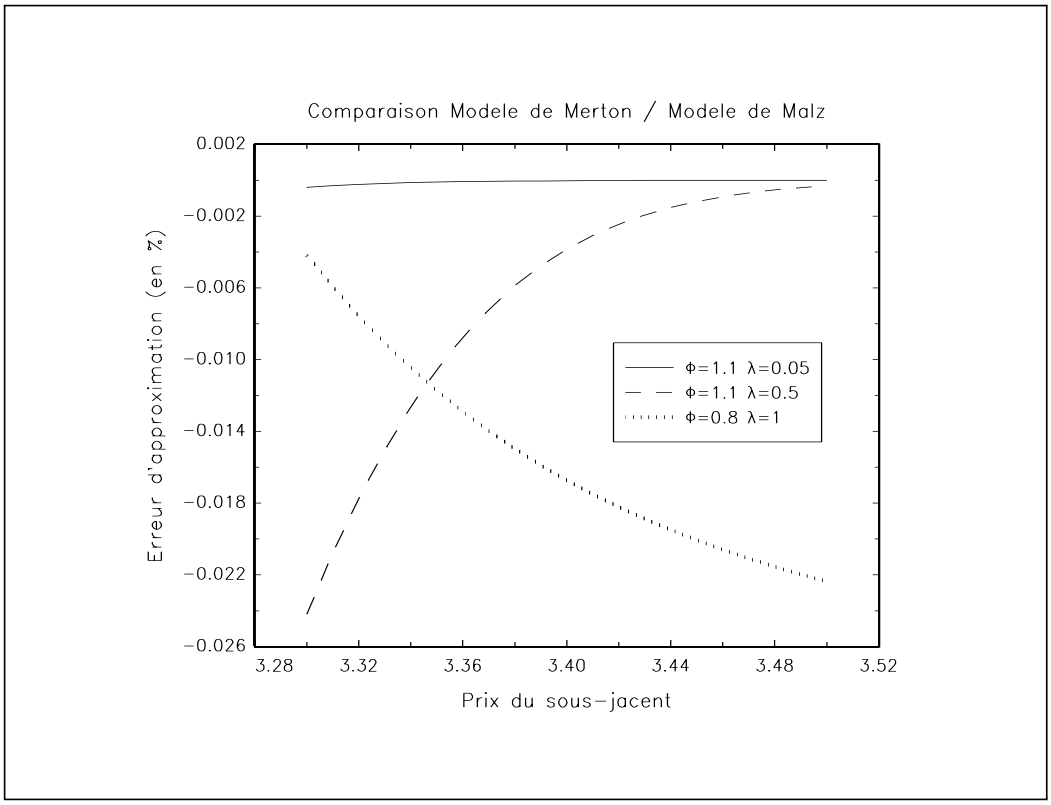
malz.src

```
proc _Garman_Kohlhagen(S0,K,sigma,tau,r,rstar,type_option);
  local w,d1,d2,C,P,Prime;

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

  C = S0.*exp(-rstar.*tau).*cdfn(d1)-K.*exp(-r.*tau).*cdfn(d2);

  P = -S0.*exp(-rstar.*tau).*cdfn(-d1)+K.*exp(-r.*tau).*cdfn(-d2);

  Prime = C.*type_option + P.*(1-type_option);

  retp(Prime);
endp;


proc _Malz(S0,K,sigma,tau,r,rstar,theta,lambda,type_option);
  local prob,Stilde,Prime;

  prob = 1 - lambda.*tau;
  Stilde = S0./(1+lambda.*(theta-1).*tau);
```

```
  Prime = prob.*_Garman_Kohlhagen(Stilde,K,sigma,tau,r,rstar,type_option) +
    (1-prob).*_Garman_Kohlhagen(Stilde.*theta,K,sigma,tau,r,rstar,type_option);

  retp(Prime);
endp;

proc __Malz(coeff);
  local u;

  u = _Malz_Prime - _Malz(_Malz_S0,_Malz_K,coeff[1],_Malz_tau,_malz_r,
      _Malz_rstar,coeff[2],coeff[3],_Malz_type_option);

  retp(u'u);
endp;


proc (3) = Malz(S0,K,tau,r,rstar,type_option,Prime,CC,c);
  local sv,coeff,fmin,grd,retcode,sigma,theta,lambda;

  _malz_S0 = S0;
  _malz_K = K;
  _malz_tau = tau;
  _malz_r = r;
  _malz_rstar = rstar;
  _malz_type_option = type_option;
  _malz_Prime = Prime;

  sv = 0.25|1.1|0.1;
  if CC == 0;
    {coeff,fmin,grd,retcode} = optmum(&__Malz,sv);
  else;
    {coeff,fmin,grd,retcode} = optmum2(&__Malz,sv,CC,c);
  endif;

  sigma = coeff[1];
  theta = coeff[2];
  lambda = coeff[3];

  retp(sigma,theta,lambda);
endp;
```

Le modèle de MALZ [1996] approxime assez bien le modèle de MERTON [1976]. Sur le graphique (33), nous avons représenté l'erreur d'approximation pour différentes valeurs de λ et θ. Celle-ci est généralement inférieure à 10%, sauf pour des valeurs très fortes de λ.

```
new;
library tsm,optmum,finance,pgraph;

r = 3.50/100;       /* Taux d'interet national */
rstar = 3.10/100;   /* Taux d'interet etranger */
K = 3.35;           /* Prix d'exercice         */
sigma = 0.05;       /* Volatilite              */
tau = 90/365;       /* Maturite 90 jours       */

S0 = seqa(3.30,0.01,21);

Merton1 = 100*Merton(S0,K,sigma,tau,r,rstar,1.1,0.05,10);
Merton2 = 100*Merton(S0,K,sigma,tau,r,rstar,1.1,0.5,10);
Merton3 = 100*Merton(S0,K,sigma,tau,r,rstar,0.8,1,10);
```



Graphique 33

```
Malz1 = 100*_Malz(S0,K,sigma,tau,r,rstar,1.1,0.05,1);
Malz2 = 100*_Malz(S0,K,sigma,tau,r,rstar,1.1,0.5,1);
Malz3 = 100*_Malz(S0,K,sigma,tau,r,rstar,0.8,1,1);


err1 = Malz1./Merton1 - 1;
err2 = Malz2./Merton2 - 1;
err3 = Malz3./Merton3 - 1;


graphset;
  fonts("simplex simgrma");
  title("Comparaison Modele de Merton / Modele de Malz");
  _pdate = ""; _pnum = 2; _plwidth = 0|0|10;
  ylabel("Erreur d'approximation (en %)");
  xlabel("Prix du sous-jacent");
  _plegstr = "\202F\201=1.1 \202l\201=0.05"\
             "\0\202F\201=1.1 \202l\201=0.5"\
             "\0\202F\201=0.8 \202l\201=1";
  _plegctl = {2 5 5.5 3};
  graphprt("-c=1 -cf=malz1.eps -w=5");
  xy(S0,err1~err2~err3);
```

Le programme suivant illustre l'utilisation de la procédure `Malz`.

```
new;
library finance,tsm,optmum,pgraph;

rndseed 123;

r = 3.50/100;       /* Taux d'interet national */
```

```
rstar = 3.10/100;   /* Taux d'interet etranger */
vol = 0.05;         /* Volatilite              */
tau = 90/365;       /* Maturite 90 jours       */

Nobs = 25;
K = 3.25 + 0.20*rndu(Nobs,1);
S0 = 3.25 + 0.20*rndu(Nobs,1);
type_option = rndu(Nobs,1) .< 0.5;
Prime = _Garman_Kohlhagen(S0,K,vol,tau,r,rstar,type_option);

let parname = "sigma" "theta" "lambda";

output file = malz2.out reset;

{sigma,theta,lambda} = Malz(S0,K,tau,r,rstar,type_option,Prime,0,0);

print "Estimation non contrainte";
print;
omat = parname~(vol|1|0)~(sigma|theta|lambda);
print "              val. theorique    estimation ";
print "-----------------------------------------";
call printfmt(omat,0~1~1);
print;
print ftos(lambda*(theta-1),"Ampleur anticipee du saut : %lf",5,3);


CC = 0~1~0; c = 1.025;
{sigma,theta,lambda} = Malz(S0,K,tau,r,rstar,type_option,Prime,CC,c);

print; print;
print "Estimation contrainte";
print;
omat = parname~(vol|1|0)~(sigma|theta|lambda);
print "              val. theorique    estimation ";
print "-----------------------------------------";
call printfmt(omat,0~1~1);
print;
print ftos(lambda*(theta-1),"Ampleur anticipee du saut : %lf",5,3);

output off;
```

```
Estimation non contrainte

              val. theorique    estimation
-----------------------------------------
   sigma             0.05      0.049874463
   theta                1        1.0150009
  lambda                0      0.058105809

Ampleur anticipee du saut : 0.001


Estimation contrainte

              val. theorique    estimation
-----------------------------------------
   sigma             0.05       0.04943159
   theta                1            1.025
  lambda                0      0.098205101
```

```
Ampleur anticipee du saut : 0.002
```

6.2.2 Estimation de la mesure de probabilité neutre au risque

Références

Bahra, B. [1996], Probability distributions of future asset prices implied by option prices, *Bank of England Quaterly Bulletin*, **August**, 299-311

Sherrick, B.J., P. Garcia et V. Tirupattur [1996], Recovering probabilistic information from option markets: Tests of distributional assumptions, *Journal of Futures Markets*, **16**, 545-560

Nous supposons en général que le rendement d'un actif est gaussien, ce qui implique que le prix de l'actif est une variable aléatoire log-normale. Nous rappelons que les fonctions de densité et de répartition de la loi log-normale LN (μ,σ) sont

$$\text{pdf}\,(x) = \frac{1}{\sigma x\sqrt{2\pi}}\exp -\frac{1}{2}\left(\frac{\ln x-\mu}{\sigma}\right)^2$$

$$\text{cdf}\,(x) = \Phi\left(\frac{\ln x-\mu}{\sigma}\right)$$

Cette hypothèse de log-normalité des prix des actifs est cependant peu justifiée pour rendre compte de certains faits empiriques. Sherrick, Garcia et Tirupattur [1996] utilisent la loi de probabilité Burr III pour mieux rendre compte de certains effets de skewness et de kurtosis. La fonction de densité de Burr3 (a,λ,τ) est

$$\text{pdf}\,(x) = \frac{a\lambda x^{a\,\lambda-1}\tau^a}{\left(\tau^a+x^a\right)^{\lambda+1}}$$

La fonction de répartition est donnée par l'expression suivante

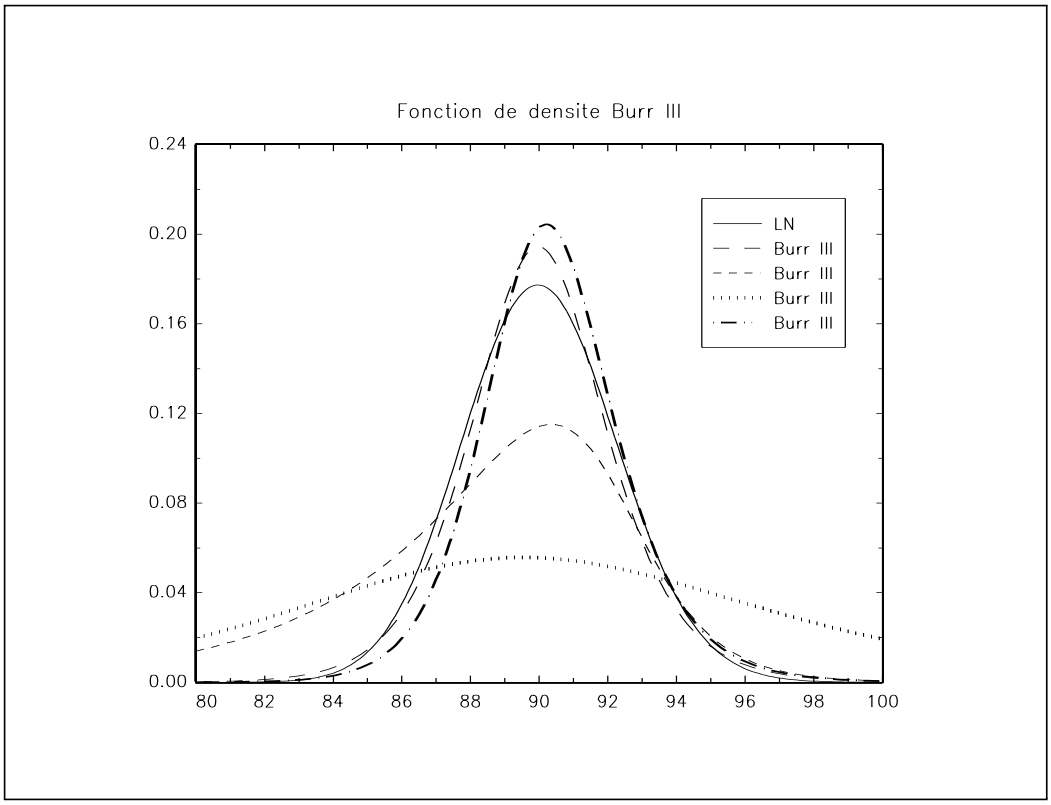
$$\text{cdf}\,(x) = \left(1-\frac{1}{\left(1+\left(\frac{x}{\tau}\right)^a\right)}\right)^{\lambda}$$

burr3.src

```
proc pdf_ln(x,mu,sigma);
  retp( pdfn((ln(x)-mu)./sigma)./(sigma.*x) );
endp;

proc cdf_ln(x,mu,sigma);
  retp( cdfn((ln(x)-mu)./sigma));
endp;

proc pdf_Burr3(x,a,lambda,tau);
  local w1,w2,w3,w4,w5,w6,w7;
  w1 = x^a;
  w2 = tau^(a);
  w3 = a.*lambda;
  w4 = x^(w3-1);
  w6 = (tau^a + w1);
  w7 = w6^(lambda+1);
  retp( w3.*w4.*w2./w7 );
endp;

proc cdf_Burr3(x,a,lambda,tau);
  retp( (1 - 1./(1+((x./tau)^a)))^lambda );
endp;
```

Le graphique (34) présente différents exemples de densité de la loi Burr III. Elle permet d'approximer assez bien la loi log-normale. Mais elle permet surtout de rendre compte d'effets d'asymétrie.

```
new;
library finance,pgraph;
```



Graphique 34

```
x = seqa(80,0.1,200);

pdf1 = pdf_ln(x,4.5,0.025);
pdf2 = pdf_burr3(x,70,1,90);
pdf3 = pdf_burr3(x,70,0.30,92);
pdf4 = pdf_burr3(x,20,1,90);

graphset;
  title("Fonction de densite Burr III");
  _pdate = ""; _pnum = 2; _pltype = 6|1|3|2; _plwidth = 0|0|0|10;
  _plegstr = "LN\0Burr III\0Burr III\0Burr III";
  _plegctl = {2 5 6.5 4};
  graphprt("-c=1 -cf=burr3.eps -w=5");
  xy(x,pdf1~pdf2~pdf3~pdf4);
```

Considérons N_1 options d'achat et N_2 options de vente. Sherrick, Garcia et Tirupattur [1996] propose alors d'estimer la fonction de densité neutre au risque de $S(T)$. Pour cela, nous estimons le vecteur θ des coefficients de la loi de densité en minimisant l'expression suivante

$$\sum_{i=1}^{N_1}\left(C_i - e^{-r_i\tau_i}\int_{K_i}^{\infty}(x-K_i)\,\mathrm{pdf}\left(x|\,\theta\right)\,dx\right)^2 + \sum_{j=1}^{N_2}\left(P_j - e^{-r_j\tau_j}\int_{0}^{K_j}(K_j-x)\,\mathrm{pdf}\left(x|\,\theta\right)\,dx\right)^2$$

avec $\theta = \begin{bmatrix} \mu \\ \sigma \end{bmatrix}$ pour la loi log-normale et $\theta = \begin{bmatrix} a \\ \lambda \\ \tau \end{bmatrix}$ pour la loi de Burr III. La procédure `optmum` a besoin de valeurs de départ pour optimiser la fonction précédente. Pour la loi de densité log-normale, nous pouvons choisir

$$sv = \begin{bmatrix} \ln S_0 + \left(b - \frac{1}{2}\sigma_{\mathrm{imp}}^2\right)\tau \\ \sigma_{\mathrm{imp}}\sqrt{\tau} \end{bmatrix}$$

avec b, le paramètre "cost of carry" des options, S_0 la valeur du sous-jacent, σ_{imp} la volatilité implicite des options et τ la maturité des options. La procédure `Sherrick_Garcia_Tirupattur` estime dans un premier temps les coefficients de la fonction de densité neutre au risque log-normale, puis cherche les valeurs de départ pour la loi Burr III en approximant la fonction de répartition neutre au risque log-normale par une Burr III et enfin estime les coefficients de la fonction de densité neutre au risque Burr III.

Sherrick_Garcia_Tirupattur

■ **Objectif**
Estimation des paramètres de la fonction de densité neutre au risque de $S(T)$ (lois log-normale et Burr III).

■ **Format**
{coeff_LN,coeff_Burr3} = Sherrick_Garcia_Tirupattur(Prime,K,r,tau,type_option,sv)

■ **Entrées**

Prime	vecteur $N \times 1$, prime des options.
K	vecteur $N \times 1$, prix d'exercice.
tau	vecteur $N \times 1$, maturité des options.
r	vecteur $N \times 1$, taux d'intérêt national.
type_option	vecteur $N \times 1$, type des options (1 pour un call et 0 pour un put).
sv	vecteur 2×1, valeurs de départ pour la procédure d'optimisation.

■ **Sorties**

coeff_LN	vecteur 2×1, paramètres estimés pour la loi log-normale.
coeff_Burr3	vecteur 3×1, paramètres estimés pour la loi Burr III.

■ **Remarque**
Pour approximer $\int_0^K f(x)\,dx$ et $\int_K^\infty f(x)\,dx$ par $\int_{c_1}^K f(x)\,dx$ et $\int_K^{c_2} f(x)\,dx$, nous employons deux variables externes `_Sherrick_zero` (c_1) et `_Sherrick_infini` (c_2).

sherrick.dec

```
declare matrix _Sherrick_infini = 0.1;
declare matrix _Sherrick_zero = 10000;

declare matrix _Sherrick_Prime;
declare matrix _Sherrick_K;
declare matrix _Sherrick_r;
declare matrix _Sherrick_tau;
declare matrix _Sherrick_type_option;
declare matrix _Sherrick_pdf;
declare matrix _Sherrick_coeff;
declare matrix _Sherrick_Exercice;
declare matrix _Burr3_x;
declare matrix _Burr3_cdf;
```

sherrick.ext

```
external matrix _Sherrick_infini;
external matrix _Sherrick_zero;

external matrix _Sherrick_Prime;
external matrix _Sherrick_K;
external matrix _Sherrick_r;
external matrix _Sherrick_tau;
external matrix _Sherrick_type_option;
external matrix _Sherrick_pdf;
external matrix _Sherrick_coeff;
external matrix _Sherrick_Exercice;
external matrix _Burr3_x;
external matrix _Burr3_cdf;
```

sherrick.src

```
proc (2) = Sherrick_Garcia_Tirupattur(Prime,K,r,tau,type_option,sv);
  local coeff_LN,sv_Burr3,coeff_Burr3,fmin,grd,retcode;

  _Sherrick_Prime = Prime;
  _Sherrick_K = K;
  _Sherrick_r = r;
  _Sherrick_tau = tau;
  _Sherrick_type_option = type_option;

  _Sherrick_pdf = "LN";
  {coeff_LN,fmin,grd,retcode} = optmum(&_Sherrick_critere,sv);

  _Sherrick_pdf = "Burr3";
  sv_Burr3 = _approximation_Burr3(coeff_LN);
  {coeff_Burr3,fmin,grd,retcode} = optmum(&_Sherrick_critere,sv_Burr3);

  retp(abs(coeff_LN),abs(coeff_Burr3));
endp;

proc _Sherrick_Critere(coeff);
  local Prime,K,r,tau,type_option;
  local Nobs,i,Prime_theorique,u;

  coeff = abs(coeff);

  Prime = _Sherrick_Prime;
  K = _Sherrick_K;
  r = _Sherrick_r;
  tau = _Sherrick_tau;
  type_option = _Sherrick_type_option;

  Nobs = rows(Prime);
  Prime_theorique = zeros(Nobs,1);

  i = 1;
  do until i > Nobs;
    Prime_Theorique[i] = __Sherrick_Prime(coeff,K[i],type_option[i]);
    i = i + 1;
  endo;

  u = Prime - exp(-r.*tau).*Prime_Theorique;

  retp(u'u);
endp;

proc __Sherrick_Prime(coeff,K,type_option);
  local Prime;

  _Sherrick_coeff = coeff;
  _Sherrick_Exercice = K;

  if type_option == 1;
    Prime = intsimp(&__Sherrick_CALL,_sherrick_infini|K,1e-6);
  else;
    Prime = intsimp(&__Sherrick_PUT,K|_sherrick_zero,1e-6);
  endif;

  retp(Prime);
endp;
```

```
proc __sherrick_CALL(S);
  local coeff,K,fct;
  coeff = _Sherrick_coeff;
  K = _Sherrick_Exercice;

  if _Sherrick_pdf $== "LN";
    fct =  pdf_LN(S,coeff[1],coeff[2]).*(S-K);
  else;
    fct =  pdf_Burr3(S,coeff[1],coeff[2],coeff[3]).*(S-K);
  endif;

  retp(fct);
endp;

proc __sherrick_PUT(S);
  local coeff,K,fct;
  coeff = _Sherrick_coeff;
  K = _Sherrick_Exercice;

  if _Sherrick_pdf $== "LN";
    fct =  pdf_LN(S,coeff[1],coeff[2]).*(K-S);
  else;
    fct =  pdf_Burr3(S,coeff[1],coeff[2],coeff[3]).*(K-S);
  endif;

  retp(fct);
endp;

proc _approximation_Burr3(coeff_LN);
  local mu,sigma,moy,x,cdf,coeff_Burr3,fmin,grd,retcode;

  mu = coeff_LN[1];
  sigma = coeff_LN[2];

  moy = exp(mu+0.5*sigma^2);
  x = seqa(0.2*moy,4.8*moy/100,100);

  cdf = cdf_LN(x,mu,sigma);

  _Burr3_x = x;
  _Burr3_cdf = cdf;

  {coeff_Burr3,fmin,grd,retcode} = optmum(&_Burr3_critere,1|1|1);


  retp(abs(coeff_Burr3));
endp;

proc _Burr3_critere(coeff);
  local u;

  coeff = abs(coeff);
  u = _Burr3_cdf - cdf_Burr3(_Burr3_x,coeff[1],coeff[2],coeff[3]);

  retp(u'u);
endp;
```

La procédure `Sherrick_Garcia_Tirupattur` peut être utilisée pour estimer les fonctions de densité neutre au risque des cours de change, des contrats à terme, etc. Dans le programme suivant, nous comparons les fonctions de densité estimée avec celle théorique (voir le graphique (35)).

```
new;
library optmum,finance,pgraph;

F0 = 100;
sigma = 0.15;
tau = 90/360;
r = 0.10;
K = seqa(90,5,5);

_type_option = "call";
C = Black(F0,K,sigma,tau,r);
_type_option = "put";
P = Black(F0,K,sigma,tau,r);

Prime = C|P;
Exercice = K|K;
type_option = ones(5,1)|zeros(5,1);

_sherrick_zero = 0.1;
_sherrick_infini = 1000;

sv = 4|.1;

{coeff_LN,coeff_Burr3} =
Sherrick_Garcia_Tirupattur(Prime,Exercice,r,tau,type_option,sv);

Future = seqa(80,1,40);
pdf = pdf_LN(Future,ln(F0)-0.5*sigma^2*tau,sigma*sqrt(tau));
pdf1 = pdf_Ln(Future,coeff_LN[1],coeff_LN[2]);
pdf2 = pdf_Burr3(Future,coeff_Burr3[1],coeff_Burr3[2],coeff_Burr3[3]);

graphset;
  _pdate = ""; _pnum = 2; _pltype = 6|1|3;
  title("Estimation de la fonction de densite neutre au risque");
  _plegstr = "pdf theorique\0pdf LN estimee\0pdf Burr III estimee";
  _plegctl = {2 5 3 1};
  graphprt("-c=1 -cf=sherrick.eps -w=5");
  xy(Future,pdf~pdf1~pdf2);
```

Nous avons rencontré des problèmes de convergence lors de l'utilisation de la procédure `Sherrick_Garcia_Tirupattur` avec des données réelles. C'est pourquoi nous proposons une autre procédure `Bahra`. Bahra [1996] propose la même méthode que Sherrick, Garcia et Tirupattur [1996] pour estimer la fonction de densité neutre au risque. La seule différence est la loi de probabilité qu'il suppose être un mélange de deux lois log-normales

$$\theta \,\mathrm{LN}\,(\mu_1, \sigma_1) + (1-\theta)\,\mathrm{LN}\,(\mu_2, \sigma_2)$$

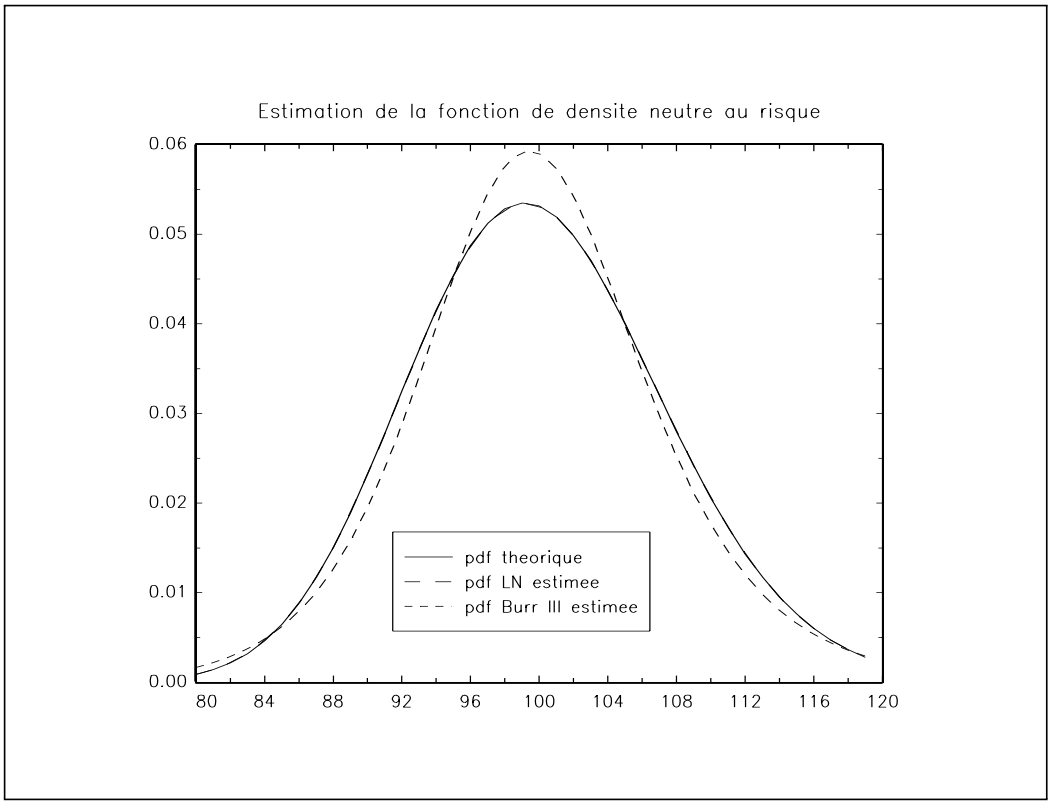
La fonction de densité dépend donc de 5 paramètres

$$\alpha = \begin{bmatrix} \mu_1 \\ \sigma_1 \\ \mu_2 \\ \sigma_2 \\ \theta \end{bmatrix}$$

Bahra

■ **Objectif**
Estimation des paramètres de la fonction de densité neutre au risque de $S\,(T)$.

■ **Format**
coeff = Bahra(Prime,K,r,tau,type_option,sv,CC,c)



Graphique 35

■ **Entrées**

Prime	vecteur $N \times 1$, prime des options.
K	vecteur $N \times 1$, prix d'exercice.
tau	vecteur $N \times 1$, maturité des options.
r	vecteur $N \times 1$, taux d'intérêt national.
type_option	vecteur $N \times 1$, type des options (1 pour un call et 0 pour un put).
sv	vecteur 5×1, valeurs de départ pour la procédure d'optimisation.
CC	matrix $g \times 5$, matrice C du sytème de restrictions $C\alpha = c$.
c	vecteur $g \times 1$, vecteur c du sytème de restrictions $C\alpha = c$.

■ **Sorties**

coeff	vecteur 5×1, paramètres estimés.

■ **Remarque**

Pour approximer $\int_0^K f(x)\,dx$ et $\int_K^\infty f(x)\,dx$ par $\int_{c_1}^K f(x)\,dx$ et $\int_K^{c_2} f(x)\,dx$, nous employons deux variables externes `_Bahra_zero` (c_1) et `_Bahra_infini` (c_2). La variable externe `_Bahra_int` permet de choisir l'algorithme d'intégration (`''simpson''` ou `''intquad1''`). `_Bahra_tol` controle la convergence de l'algorithme de Simpson. Pour imposer que $\theta \in [0,1]$, nous initialisons `_Bahra_theta` à 1.

bahra.dec

```
declare matrix _Bahra_infini = 0.1;
declare matrix _Bahra_zero = 10000;
declare matrix _Bahra_tol = 1e-6;
declare matrix _Bahra_int ?= "simpson";
declare matrix _Bahra_theta = 0;

declare matrix _Bahra_Prime;
declare matrix _Bahra_K;
declare matrix _Bahra_r;
declare matrix _Bahra_tau;
```

```
declare matrix _Bahra_type_option;
declare matrix _Bahra_coeff;
declare matrix _Bahra_Exercice;
```

bahra.ext

```
external matrix _Bahra_infini;
external matrix _Bahra_zero;
external matrix _Bahra_tol;
external matrix _Bahra_int;
external matrix _Bahra_theta;

external matrix _Bahra_Prime;
external matrix _Bahra_K;
external matrix _Bahra_r;
external matrix _Bahra_tau;
external matrix _Bahra_type_option;
external matrix _Bahra_coeff;
external matrix _Bahra_Exercice;
```

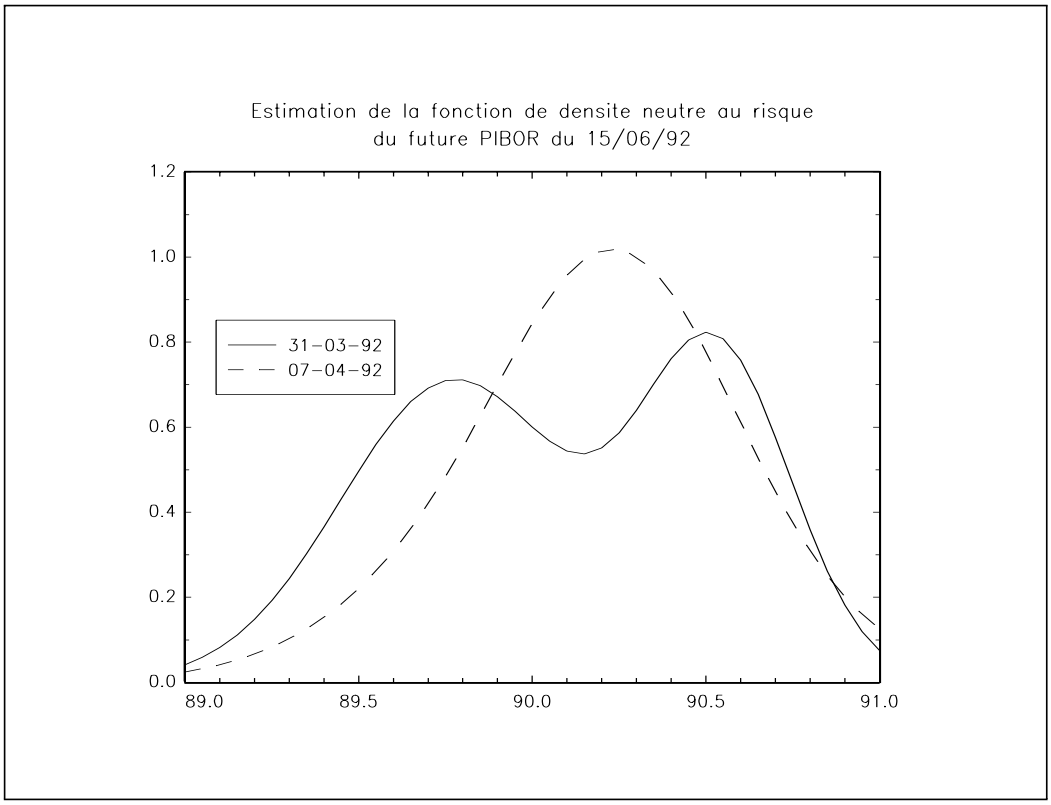
bahra.src

```
proc (1) = Bahra(Prime,K,r,tau,type_option,sv,CC,c);
  local coeff,fmin,grd,retcode;

  _Bahra_Prime = Prime;
  _Bahra_K = K;
  _Bahra_r = r;
  _Bahra_tau = tau;
  _Bahra_type_option = type_option;

  {coeff,fmin,grd,retcode} = optmum2(&_Bahra_critere,sv,CC,c);

  if _Bahra_theta == 1;
    coeff[5] = exp(-coeff[5])/(1+exp(-coeff[5]));
  endif;


  retp(abs(coeff));
endp;

proc _Bahra_Critere(coeff);
  local Prime,K,r,tau,type_option;
  local Nobs,i,Prime_theorique,u;

  coeff = abs(coeff);

  Prime = _Bahra_Prime;
  K = _Bahra_K;
  r = _Bahra_r;
  tau = _Bahra_tau;
  type_option = _Bahra_type_option;

  Nobs = rows(Prime);
  Prime_theorique = zeros(Nobs,1);

  i = 1;
  do until i > Nobs;
    Prime_Theorique[i] = __Bahra_Prime(coeff,K[i],type_option[i]);
    i = i + 1;
  endo;
```

```
  u = Prime - exp(-r.*tau).*Prime_Theorique;

  retp(u'u);
endp;

proc __Bahra_Prime(coeff,K,type_option);
  local Prime;

  _Bahra_coeff = coeff;
  _Bahra_Exercice = K;

  if type_option == 1;
    if _Bahra_int $== "simpson";
      Prime = intsimp(&__Bahra_CALL,_Bahra_infini|K,_Bahra_tol);
    else;
      Prime = intquad1(&__Bahra_CALL,_Bahra_infini|K);
    endif;
  else;
    if _Bahra_int $== "simpson";
      Prime = intsimp(&__Bahra_PUT,K|_Bahra_zero,_Bahra_tol);
    else;
      Prime = intquad1(&__Bahra_PUT,K|_Bahra_zero);
    endif;
  endif;

  retp(Prime);
endp;

proc __Bahra_CALL(S);
  local coeff,K,fct;
  coeff = _Bahra_coeff;
  K = _Bahra_Exercice;

  fct =  pdf_2LN(S,coeff[1],coeff[2],coeff[3],coeff[4],coeff[5]).*(S-K);

  retp(fct);
endp;

proc __Bahra_PUT(S);
  local coeff,K,fct;
  coeff = _Bahra_coeff;
  K = _Bahra_Exercice;

  if _Bahra_theta == 1;
    coeff[5] = exp(-coeff[5])/(1+exp(-coeff[5]));
  endif;

  fct =  pdf_2LN(S,coeff[1],coeff[2],coeff[3],coeff[4],coeff[5]).*(K-S);

  retp(fct);
endp;
```

pdf2ln.src

```
proc pdf_2ln(x,mu1,sigma1,mu2,sigma2,theta);
  local pdf;

  pdf = theta*pdf_ln(x,mu1,sigma1) + (1-theta)*pdf_ln(x,mu2,sigma2);
  retp(pdf);
endp;
```



Graphique 36

```
proc cdf_2ln(x,mu1,sigma1,mu2,sigma2,theta);
  local cdf;

  cdf = theta*cdf_ln(x,mu1,sigma1) + (1-theta)*cdf_ln(x,mu2,sigma2);
  retp(cdf);
endp;
```

7 Construction d'une courbe des taux

Références

Anderson, N., F. Breedon, M. Deacon, A. Derry et G. Murphy [1996], Estimating and Interpreting the Yield Curve, John Wiley & Sons

DeBandt, O. et J-P. Lesne [1991], Une analyse factorielle de la courbe des taux en France, *Banque de France*, Document de travail

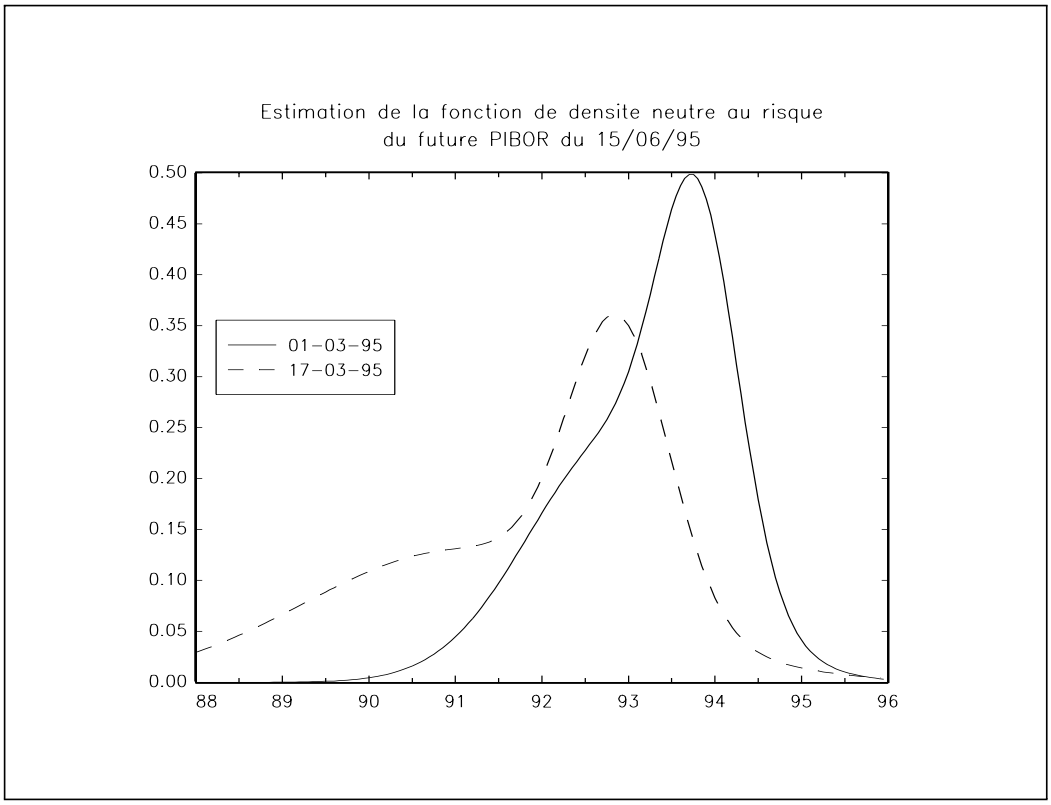
Gouriéroux, C. et O. Scaillet [1994], Estimation of the term structure from bond data, *DP Crest*, **9415**

Nelson, C.R. et A.F. Siegel [1987], Parsimonious modeling of yield curves, *Journal of Business*, **60**, 473-489

Ricart, R. et P. Sicsic [1995], Estimation d'une structure par terme des taux d'intérêt sur données françaises, *Bulletin de la Banque de France*, **22**, 117-128

Svensson, L.E.O. [1994], Estimating and interpreting forward interest rates: sweden 1992-4, *CEPR*, **1051**

Les auteurs ont proposé de nombreuses méthodes d'estimation de la courbe des taux. Elles sont décrites dans le chapitre 2 de l'ouvrage d'Anderson, Breedon, Deacon, Derry et Murphy [1996] et dans l'article de Gouriéroux et Scaillet [1994]. Ces méthodes sont très diverses, mais sont presque toutes basées sur l'estimation de la structure par terme des coupons zéros. Le nombre d'obligations à coupon zéro étant généralement faible, nous considérons les obligations d'état. Celles-ci peuvent être vue comme des titres remembrés à partir des obligations à coupon zéro. Supposons que nous disposons de N obligations d'état de prix $P_t^{(n)}$ et donc les caractéristiques sont $\left\{\left(c_j^{(n)}, \tau_j^{(n)}\right), j = 1, \ldots, J^{(n)}\right\}$ avec $J^{(n)}$ le nombre restant de coupons et $c_j^{(n)}$ la valeur du coupon de maturité



Graphique 37

$\tau_j^{(n)}$. Nous avons alors

$$P_t^{(n)} = \sum_{j=1}^{J^{(n)}} c_j^{(n)} P_t^c \left(\tau_j^{(n)}\right)$$

avec $P_t^c \left(\tau_j^{(n)}\right)$ le prix d'un coupon zéro de maturité $\tau_j^{(n)}$ et de nominal 1 Franc. Pour estimer la structure par terme des coupons zéros, nous devons estimer le vecteur $\hat{\theta}$ des paramètres du modèle

$$\hat{\theta} = \arg\min \sum_{n=1}^{N} w_n \left(P_t^{(n)}(\theta) - P_t^{(n)}\right)^2 \tag{27}$$

avec w_n le poids accordé à l'obligation n et $P_t^{(n)}(\theta)$ la valeur théorique du prix de l'obligation n donnée par le modèle considéré. Ce modèle peut être par exemple celui de Debandt et Lesne [1991], de Nelson et Siegel [1987] ou de Svensson [1994].

7.1 Procédure de lecture des caractéristiques d'une obligation

Pour estimer $\hat{\theta}$ à partir des obligations d'état et non à partir des OAT démembrées, il est nécessaire de proposer un codage de ces obligations. Nous proposons que les caractéristiques d'une obligation soient codées dans un vecteur de la manière suivante

$$\begin{bmatrix} J^{(n)} \\ P_t^{(n)} \\ \tau_1^{(n)} \\ c_1^{(n)} \\ \vdots \\ \tau_{J^{(n)}}^{(n)} \\ c_{J^{(n)}}^{(n)} \end{bmatrix}$$

La première composante du vecteur correspond au nombre de coupons $J^{(n)}$ versés par l'obligation. Le prix actuel de l'obligation $P_t^{(n)}$ est défini par la deuxième composante. Puis, nous codons successivement la maturité et la

valeur de chaque coupon. La procédure `Obligation_Maturite_Coupon` présenté ci-dessous permet d'extraire le prix de l'obligation $P_t^{(n)}$, le vecteur des maturités $\begin{bmatrix} \tau_1^{(n)} & \cdots & \tau_{J(n)}^{(n)} \end{bmatrix}^\top$ et le vecteur de la valeur des coupons $\begin{bmatrix} c_1^{(n)} & \cdots & c_{J(n)}^{(n)} \end{bmatrix}^\top$.

Obligation_Maturite_Coupon

■ **Objectif**
Extraction du prix, de la valeur et de la maturité des coupons d'une obligation.

■ **Format**
{Prix,tau,C} = Obligation_Maturite_Coupon(x);

■ **Entrées**

x	vecteur $L \times 1$, vecteur de codage de l'information de l'obligation.

■ **Sorties**

Prix	scalaire, prix actuel de l'obligation.
tau	vecteur $JN \times 1$, maturité des coupons.
C	vecteur $JN \times 1$, valeur des coupons.

■ **Remarque**
L doit être au moins égal à $2 \times JN + 2$.

titre.src

```
proc (3) = Obligation_Maturite_Coupon(x);
  local nb,Prix,y,tau,C;

  nb = x[1];   /* Nombre de coupons */
  Prix = x[2];  /* Valeur de l'obligation */

  y = x[3:2*(1+nb)];
  y = reshape(y,nb,2);
  tau = y[.,1];
  C = y[.,2];

  retp(Prix,tau,C);
endp;
```

Ce codage et la procédure de lecture des caractéritiques d'une obligation vont être très utile dans le cas où les obligations sont de nature différentes avec un nombre différent de coupons. Considérons un exemple. Nous avons pour l'obligation O_1

$$\begin{bmatrix} 5 \\ 119.95 \\ \frac{25}{360} \\ 8.55 \\ \frac{25}{360}+1 \\ 8.55 \\ \frac{25}{360}+2 \\ 8.55 \\ \frac{25}{360}+3 \\ 8.55 \\ \frac{25}{360}+4 \\ 108.55 \end{bmatrix}$$

pour l'obligation O_2

$$\begin{bmatrix} 1 \\ 94.50 \\ 1+\frac{50}{360} \\ 100 \end{bmatrix}$$

et pour l'obligation O_3

$$\begin{bmatrix} 3 \\ 115.70 \\ \frac{75}{360} \\ 10.15 \\ \frac{75}{360}+1 \\ 7.45 \\ \frac{75}{360}+2 \\ 109.35 \end{bmatrix}$$

La création d'une base de données contenant les caractéristiques de ces trois obligations ne pose plus de problèmes. Nous avons

5	1	3
119.95	94.50	115.70
$\frac{25}{360}$	$1+\frac{50}{360}$	$\frac{75}{360}$
8.55	100	10.15
$\frac{25}{360}+1$	·	$\frac{75}{360}+1$
8.55	·	7.45
$\frac{25}{360}+2$	·	$\frac{75}{360}+2$
8.55	·	109.35
$\frac{25}{360}+3$	·	·
8.55	·	·
$\frac{25}{360}+4$	·	·
108.55	·	·

Le symbole · indique une valeur quelconque. Pour lire les caractéristiques de l'obligation i, il suffit d'extraire ces caractéristiques de la i-ième colonne de la base de données avec la procédure `Obligation_Maturite_Coupon`.

titre.dat

```
5           1           3
119.95      94.50       115.70
0.069444444 1.1388889   0.20833333
8.55        100         10.15
1.069444444 .           1.20833333
8.55        .           7.45
2.069444444 .           2.20833333
8.55        .           109.35
3.069444444 .           .
8.55        .           .
4.069444444 .           .
108.55      .           .
```

titre.prg

```
new;
library finance;

load data[12,3] = titre.dat;

output file = titre.out reset;

i = 1;
do until i > 3;
  {Prix,tau,Coupon} = Obligation_Maturite_Coupon(data[.,i]);
  print "---------------------------------------";
  print ftos(i,"Obligation O%lf",1,0);
  print ftos(Prix,"Valeur actuelle : %lf",5,3);
  print "     Val. Coupon      Mat. Coupon  ";
  print Coupon~tau;
  print "---------------------------------------";
  print;
```

```
  i = i + 1;
endo;

output off;
```

titre.out

```
----------------------------------------
Obligation O1
Valeur actuelle : 119.950
     Val. Coupon      Mat. Coupon

       8.5500000      0.069444444
       8.5500000        1.0694444
       8.5500000        2.0694444
       8.5500000        3.0694444
       108.55000        4.0694444
----------------------------------------

----------------------------------------
Obligation O2
Valeur actuelle : 94.500
     Val. Coupon      Mat. Coupon
       100.00000        1.1388889
----------------------------------------

----------------------------------------
Obligation O3
Valeur actuelle : 115.700
     Val. Coupon      Mat. Coupon

       10.150000       0.20833333
       7.4500000        1.2083333
       109.35000        2.2083333
----------------------------------------
```

7.2 Le modèle de Nelson et Siegel [1987]

Nelson et Siegel [1987] supposent que le taux d'intérêt à terme instantanné $f_t(\tau)$ est solution d'une équation différentielle ordinaire du second ordre dans le cas d'une racine double :

$$F_t(\tau) = \mu_1 + \mu_2 \exp\left(-\frac{\tau}{\tau_1}\right) + \mu_3 \frac{\tau}{\tau_1} \exp\left(-\frac{\tau}{\tau_1}\right)$$

Ce modèle dépend donc de 4 paramètres. Nous avons

$$\theta = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \tau_1 \end{bmatrix}$$

Le taux d'intérêt de maturité τ est défini par

$$R_t(\tau) = \frac{1}{\tau} \int_0^\tau F_t(\theta)\, d\theta$$

Après calcul, nous obtenons

$$R_t(\tau) = \mu_1 + \mu_2 \left[\frac{1 - \exp\left(-\frac{\tau}{\tau_1}\right)}{\frac{\tau}{\tau_1}}\right] + \mu_3 \left[\frac{1 - \exp\left(-\frac{\tau}{\tau_1}\right)}{\frac{\tau}{\tau_1}} - \exp\left(-\frac{\tau}{\tau_1}\right)\right]$$

La valeur du coupon zéro est alors

$$P_t^c(\tau) = \exp(-R_t(\tau)\,\tau)$$

Soit $F_t(\tau, m)$ le taux à terme à la date t valable dans τ périodes de maturité m (c'est-à-dire que le contrat à terme couvre la période allant de $t+\tau$ à $t+\tau+m$). Nous avons

$$F_t(\tau, m) = -\frac{1}{m} \ln\left(\frac{P_t^c(\tau+m)}{P_t^c(\tau)}\right)$$

La programmation en langage **GAUSS** du modèle de NELSON et SIEGEL [1987] ne pose pas de problèmes numériques particulièrement complexes. Pour minimiser la fonction $\sum_{n=1}^{N} w_n \left(P_t^{(n)}(\theta) - P_t^{(n)}\right)^2$, nous utilisons la procédure numérique `optmum`. Il est souvent intéressant d'imposer des restrictions aux coefficients du modèle. Par exemple, le paramètre μ_1 s'interprète comme le taux long $R_t(\infty)$ et la somme $\mu_1 + \mu_2$ correspond au taux court $R_t(0)$. La procédure `NelsonSiegel` permet de prendre en compte les restrictions linéaires de la forme

$$C\theta = c$$

`NelsonSiegel` utilise alors la procédure numérique `optmum2` de la bibliothèque **TSM**.

NelsonSiegel_CourbeDesTaux

■ **Objectif**
Calcul de la courbe des taux d'intérêt $R_t(\tau)$.

■ **Format**
Taux = NelsonSiegel_CourbeDesTaux(coefficients,tau);

■ **Entrées**

coefficients	vecteur 4×1, valeur des coefficients du modèle.
tau	vecteur $T \times 1$, maturité des coupons zéros.

■ **Sorties**

taux	vecteur $T \times 1$, valeur des taux d'intérêt $R_t(\tau)$.

NelsonSiegel_CouponZero

■ **Objectif**
Calcul du prix des coupons zéros $P_t^c(\tau)$.

■ **Format**
CouponZero = NelsonSiegel_CouponZero(coefficients,tau);

■ **Entrées**

coefficients	vecteur 4×1, valeur des coefficients du modèle.
tau	vecteur $T \times 1$, maturité des coupons zéros.

■ **Sorties**

CouponZero	vecteur $T \times 1$, prix des coupons zéros $P_t^c(\tau)$.

NelsonSiegel_TauxATerme

■ **Objectif**
Calcul des taux à terme $F_t(\tau, m)$.

■ **Format**
Taux = NelsonSiegel_TauxATerme(coefficients,tau,m);

■ **Entrées**

coefficients	vecteur 4×1, valeur des coefficients du modèle.
tau	vecteur $T \times 1$ ou scalaire, valeur du paramètre τ.
m	vecteur $T \times 1$ ou scalaire, valeur du paramètre m.

■ **Sorties**

taux	vecteur $T \times 1$, valeur des taux à terme $F_t(\tau, m)$.

■ **Remarque**
Pour obtenir les taux à terme instantanné $f_t(\tau)$, il faut initialiser m à 0.

NelsonSiegel

■ **Objectif**
Estimation des coefficients du modèle de Nelson et Siegel [1987].

■ **Format**
coefficients = NelsonSiegel(Obligations,Poids,CC,c);

■ **Entrées**

Obligations	vecteur $L \times N$, base de données des obligations d'état.
Poids	vecteur $N \times 1$, vecteur des poids.
CC	matrice $g \times 4$, matrice C du système de restrictions $C\theta = c$.
c	vecteur $g \times 1$, vecteur c du système de restrictions $C\theta = c$.

■ **Sorties**

coefficients	vecteur 4×1, valeur estimée des coefficients du modèle.

■ **Remarque**
Pour une estimation sans contraintes des coefficients, il suffit d'initialiser la matrice `CC` à 0.

nelson.dec

```
declare matrix _NelsonSiegel_Obligations;
declare matrix _NelsonSiegel_Poids;
declare matrix _NelsonSiegel_coefficients;
declare matrix _NelsonSiegel_Composantes;
```

nelson.ext

```
external matrix _NelsonSiegel_Obligations;
external matrix _NelsonSiegel_Poids;
external matrix _NelsonSiegel_coefficients;
external matrix _NelsonSiegel_Composantes;
```

nelson.src

```
proc NelsonSiegel_CourbeDesTaux(coefficients,tau);
  local Nobs,mu1,mu2,mu3,tau1;
  local w,ww,www;
  local composante1,composante2,composante3,R;

  Nobs = rows(tau);

  mu1 = coefficients[1];
  mu2 = coefficients[2];
  mu3 = coefficients[3];
  tau1 = coefficients[4];

  w = tau./tau1;
  ww = exp(-w);
  www = (1-ww)./w;

  composante1 = mu1.*ones(Nobs,1);
  composante2 = mu2.*www;
  composante3 = mu3.*(www-ww);

  R = composante1 + composante2 + composante3;

  _NelsonSiegel_Composantes = composante1~composante2~composante3;

  retp(R);
endp;
```

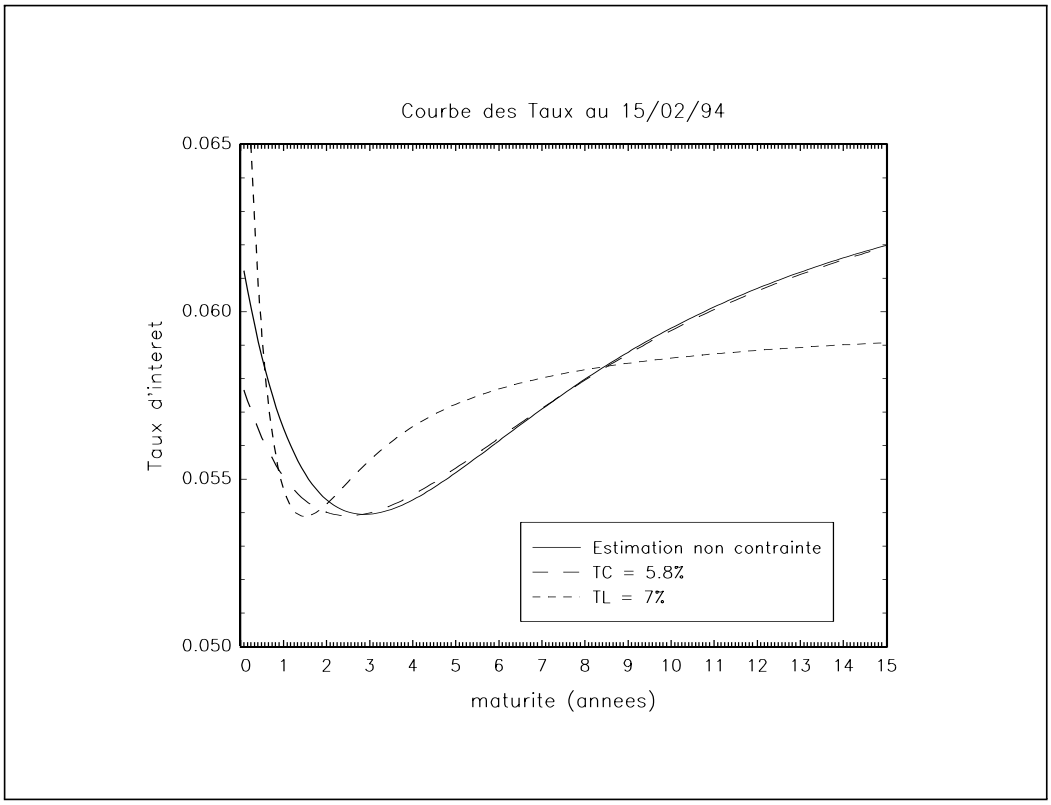
```
proc NelsonSiegel_CouponZero(coefficients,tau);
  local CouponZero;

  CouponZero = exp(-tau.*NelsonSiegel_CourbeDesTaux(coefficients,tau));

  retp(CouponZero);
endp;


proc (1) = _NelsonSiegel_ln_CouponZero(tau);
  local coefficients,tau_star,CouponZero;

  coefficients = _NelsonSiegel_Coefficients;

  CouponZero = NelsonSiegel_CouponZero(coefficients,tau);
  retp(ln(CouponZero));
endp;


proc (1) = NelsonSiegel_TauxATerme(coefficients,tau,m);
  local Taux,CZ,CZ_;

  if m == 0;

    _NelsonSiegel_Coefficients = coefficients;
    Taux = - diag(gradp(&_NelsonSiegel_ln_CouponZero,tau));

  else;

    CZ_ = NelsonSiegel_CouponZero(coefficients,tau);
    CZ = NelsonSiegel_CouponZero(coefficients,tau+m);
    Taux = -ln(CZ./CZ_)./m;

  endif;

  retp(Taux);
endp;


proc _NelsonSiegel_Critere(coefficients);
  local Obligations,Poids,NN,n,u,scr;
  local Prix_Theoriques,Prix_Observes,Prix,tau,C,R,CouponZero;

  Obligations = _NelsonSiegel_Obligations;  /* Lecture de la base de donnees */
  Poids = _NelsonSiegel_Poids;

  NN = cols(Obligations);    /* Nombre de titres */

  Prix_observes = zeros(NN,1);
  Prix_theoriques = zeros(NN,1);

  n = 1;
  do until n > NN;
    {Prix,tau,C} = Obligation_Maturite_Coupon(Obligations[.,n]);
    Prix_Observes[n] = Prix;
    CouponZero = NelsonSiegel_CouponZero(coefficients,tau);
    Prix_Theoriques[n] = sumc(C.*CouponZero);
    n = n + 1;
  endo;
```

```
  u = Prix_Observes - Prix_Theoriques;
  scr = sumc(Poids.*(u^2));

  retp(scr);
endp;


Proc (1) = NelsonSiegel(Obligations,Poids,CC,c);
  local coefficients,f,g,retcode;
  local sv;

  _NelsonSiegel_Obligations = Obligations;
  _NelsonSiegel_Poids = Poids;

  sv = 0.05|0.05|0|1;

  if CC == 0;
    {coefficients,f,g,retcode} = optmum(&_NelsonSiegel_Critere,sv);
  else;
    {coefficients,f,g,retcode} = optmum2(&_NelsonSiegel_Critere,sv,CC,c);
  endif;

  retp(coefficients);
endp;
```

Dans le programme suivant, nous estimons la courbe des taux du 15 Février 1994. La première estimation est libre. Pour la seconde estimation, nous imposons que le taux court soit égal à 5.8%. Le taux long est contraint à être égal à 6% pour la troisième estimation.

```
new;
library pgraph,tsm,optmum,finance;

load data[4,16] = CZ940215.dat;
x = seqa(1/12,1/12,180);

coefficients = NelsonSiegel(data,1,0,0);
R = NelsonSiegel_CourbeDesTaux(coefficients,x);

coefficients = NelsonSiegel(data,1,1~1~0~0,0.058);
R1 = NelsonSiegel_CourbeDesTaux(coefficients,x);

coefficients = NelsonSiegel(data,1,1~0~0~0,0.06);
R2 = NelsonSiegel_CourbeDesTaux(coefficients,x);

graphset;
  _pdate = ""; _pnum = 2;
  title("Courbe des Taux au 15/02/94");
  _pltype = 6|1|3|2;
  _plwidth = 0|0|0|10;
  xlabel("maturite (annees)"); ylabel("Taux d'interet");
  xtics(0,15,1,12);
  ytics(0.05,0.065,0.005,5);
  _plegstr = "Estimation non contrainte"\
             "\000TC = 5.8%"\
             "\000TL = 7%";
  _plegctl = {2 5 4.45 1.1};
  graphprt("-c=1 -cf=nelson1.eps -w=5");
  xy(x,R~R1~R2);
```

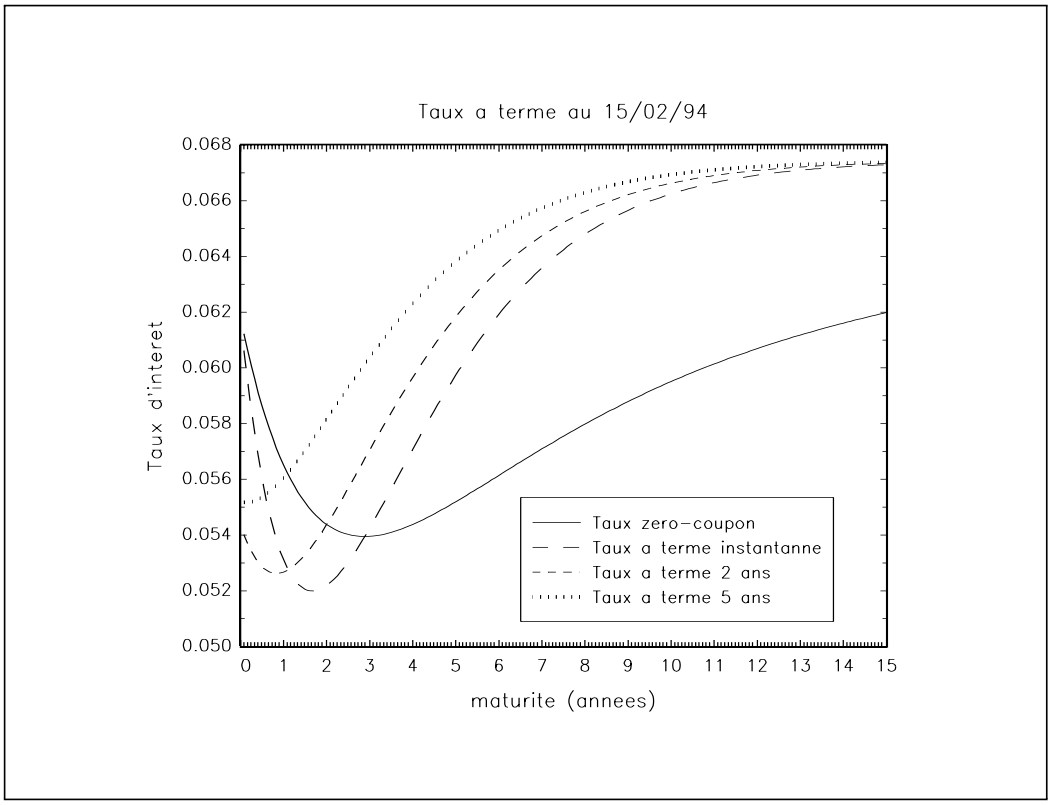
La construction de la courbe des taux est très interessante, car elle est supposée contenir de l'information sur l'évolution future des taux d'intérêt et de l'inflation. A partir de la courbe des taux du 15 Février 1994, nous



Graphique 38

pouvons calculer les taux à terme implicites. Sur le graphique (39), nous remarquons en particulier que le taux à terme instantanné implicite baisse régulièrement pour les maturités de 0 à 18 mois. Une interprétation possible est que les agents du marché obligataire anticipaient au 15 Février 1994 une baisse des taux court pendant 1 an et demi.

```
new;
library pgraph,tsm,optmum,finance;

load data[4,16] = CZ940215.dat;

coefficients = NelsonSiegel(data,1,0,0);

x = seqa(1/12,1/12,180);

R = NelsonSiegel_CourbeDesTaux(coefficients,x);
F = NelsonSiegel_TauxATerme(coefficients,x,0);

F1 = NelsonSiegel_TauxATerme(coefficients,x,2);
F2 = NelsonSiegel_TauxATerme(coefficients,x,5);

graphset;
  _pdate = ""; _pnum = 2;
  title("Taux a terme au 15/02/94");
  _pltype = 6|1|3|2;
  _plwidth = 0|0|0|10;
  xlabel("maturite (annees)"); ylabel("Taux d'interet");
  xtics(0,15,1,12);
  _plegstr = "Taux zero-coupon"\
             "\000Taux a terme instantanne"\
             "\000Taux a terme 2 ans"\
             "\000Taux a terme 5 ans";
```



Graphique 39

```
_plegctl = {2 5 4.45 1.1};
graphprt("-c=1 -cf=nelson2.eps -w=5");
xy(x,R~F~F1~F2);
```

7.3 Le prise en compte de poids endogènes

Dans la procédure `NelsonSiegel`, le vecteur des poids est exogène. Soit N le nombre d'obligations d'état de la base de données. Nous posons $w_n = \frac{1}{N}$. Dans ce cas, l'estimation du vecteur θ peut être biaisée. Par exemple, si la base de données contient principalement des obligations courtes, nous pouvons penser que la courbe des taux sera mieux estimée sur le segment court terme que sur les segments moyen et long termes. Il peut donc être intéressant d'endogénéiser le vecteur des poids. Ainsi, la Banque de France utilise le modèle de Svensson [1994] avec des poids égaux au carré de l'inverse de la sensibilité du prix de l'obligation au taux d'intérêt. La fonction à minimiser devient $\sum_{n=1}^{N} w_n(\theta)\left(P_t^{(n)}(\theta) - P_t^{(n)}\right)^2$ avec $w_n(\theta)$ le poids endogène de l'obligation n. Pour éviter des problèmes de convergence, il est nécessaire d'utiliser un algorithme récursif :

1. Choisir une valeur θ_0 de θ.
2. Calculer le vecteur des poids $w_n = w_n(\theta_i)$
3. Chercher la valeur θ_{i+1} telle que

$$\theta_{i+1} = \arg\min \sum_{n=1}^{N} w_n \left(P_t^{(n)}(\theta) - P_t^{(n)}\right)^2$$

4. Recommencer les étapes 2 et 3 jusqu'à ce que nous vérifions que $|\theta_{i+1} - \theta_i| \leq \varepsilon$.

Le programme **GAUSS** se présente de la façon suivante :

```
do until maxc(abs(theta-theta0)) <= eps;
```

```
  theta = theta0;
  w = fonction(theta);
  theta = NelsonSiegel(data,w,0,0);
endo;
```

8 Estimation des paramètres d'un processus de diffusion

8.1 Estimation des paramètres d'un mouvement brownien géométrique

Considérons le processus suivant

$$\left\{ \begin{array}{rcl} dX(t) & = & \mu X(t)\, dt + \sigma X(t)\, dW(t) \\ X(t_0) & = & x_0 \end{array} \right.$$

Nous rappelons que la solution de cette équation différentielle stochastique est

$$X(t) = x_0 \exp\left[\left(\mu - \frac{1}{2}\sigma^2\right)(t - t_0) + \sigma\left(W(t) - W(t_0)\right)\right]$$

Nous pouvons montrer que pour $t \geq t'$

$$\ln X(t) - \ln X(t') \sim \mathcal{N}\left(\left(\mu - \frac{1}{2}\sigma^2\right)(t - t'), \sigma^2(t - t')\right)$$

Nous pouvons alors estimer les paramètres $\theta = \left[\begin{array}{c} \mu \\ \sigma \end{array} \right]$ du modèle par maximum de vraisemblance ou par la méthode des moments généralisés.

8.1.1 Estimation ML

Considérons une série temporelle $\{y_t, t = 1, \ldots, T\}$ dont les observations sont mesurée à intervalle de temps constant h. La log-vraisemblance pour l'observation t est alors défini par

$$\ell_t = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln \sigma^2 h - \frac{1}{2}\frac{\left[\ln \frac{y_t}{y_{t-1}} - \left(\mu - \frac{1}{2}\sigma^2\right)h\right]^2}{\sigma^2 h}$$

```
new;
library tsm,optmum;
TSMset;

load x[500,1] = mbg.dat;

h = 0.1;

proc ml(theta);
  local e,mu,sigma,logl;

  mu = theta[1]; sigma = theta[2];
  e = ln(x[2:500]./x[1:499]) - (mu-0.5*sigma^2)*h;
  logl = -0.5*ln(2*pi) - 0.5*ln(sigma^2*h) - 0.5*e^2/(sigma^2*h);

  retp(logl);
endp;

output file = mbg1.out reset;

sv = 0|0.25;
_tsm_parnm = "mu"|"sigma";
{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;
```

```
Total observations:                                    499
Usable observations:                                   499
Number of parameters to be estimated:                    2
Degrees of freedom:                                    497
Value of the maximized log-likelihood function:  222.65429


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
mu              0.084258        0.069830         1.206610        0.228156
sigma           0.489756        0.015504        31.589353        0.000000

Covariance matrix: inverse of the negative Hessian.
```

8.1.2 Estimation GMM

Nous avons

$$\begin{cases} E_{t-1}\left[\varepsilon_t\right] = 0 \\ E_{t-1}\left[\varepsilon_t^2 - \sigma^2 h\right] = 0 \end{cases}$$

avec

$$\varepsilon_t = \ln \frac{y_t}{y_{t-1}} - \left(\mu - \frac{1}{2}\sigma^2\right) h$$

```
new;
library tsm,optmum;
TSMset;

load x[500,1] = mbg.dat;

Nobs = rows(x);
h = 0.1;

proc Moments(theta);
  local mu,sigma,e,M1,M2,M;

  mu = theta[1]; sigma = theta[2];
  e = ln(x[2:Nobs]./x[1:Nobs-1]) - (mu-0.5*sigma^2)*h;
  M1 = e;
  M2 = e^2-(sigma^2*h);
  M = M1~M2;

  retp(M);
endp;

output file = mbg2.out reset;

_tsm_parnm = "mu"|"sigma";
{theta,stderr,Mcov,Qmin} = gmm(&Moments,0|0.25);

output off;

Total observations:                                    499
Usable observations:                                   499
Number of parameters to be estimated:                    2
Degrees of freedom:                                    497
Number of moment conditions:                             2
Value of the criterion function:                   0.00000


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
```

```
mu              0.084260        0.070235         1.199698        0.230828
sigma           0.489757        0.015938        30.727964        0.000000
```

8.2 Estimation des paramètres d'un processus d'Ornstein-Uhlenbeck

Considérons le processus suivant

$$\left\{ \begin{array}{rcl} dX(t) & = & a(b - X(t))\, dt + \sigma\, dW(t) \\ X(t_0) & = & x_0 \end{array} \right.$$

Nous rappelons que la solution de cette équation différentielle stochastique est

$$X(t) = x_0 e^{-a(t-t_0)} + b\left(1 - e^{-a(t-t_0)}\right) + \sigma \int_{t_0}^{t} e^{a(\theta - t)}\, dW(\theta)$$

Nous pouvons montrer que pour $t \geq t'$

$$X(t) - e^{-a\left(t-t'\right)} X(t') \sim \mathcal{N}\left(b\left(1 - e^{-a\left(t-t'\right)}\right), \sigma^2 \frac{1 - e^{-2a\left(t-t'\right)}}{2a} \right)$$

Nous pouvons alors estimer les paramètres $\theta = \left[\begin{array}{c} a \\ b \\ \sigma \end{array} \right]$ du modèle par maximum de vraisemblance ou par la méthode des moments généralisés.

8.2.1 Estimation ML

Considérons une série temporelle $\{y_t, t = 1, \ldots, T\}$ dont les observations sont mesurée à intervalle de temps constant h. La log-vraisemblance pour l'observation t est alors défini par

$$\ell_t = -\frac{1}{2} \ln 2\pi - \frac{1}{2} \ln \sigma^2 \frac{1 - e^{-2ah}}{2a} - \frac{1}{2} \frac{\left[y_t - e^{-ah} y_{t-1} - b\left(1 - e^{-ah}\right)\right]^2}{\sigma^2 \frac{1 - e^{-2ah}}{2a}}$$

```
new;
library tsm,optmum;
TSMset;

load x[1000,1] = ou.dat;

h = 0.1;

proc ml(theta);
  local a,b,sigma,k1,k2,e,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];
  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);
  e = x[2:1000]-k1.*x[1:999]-b*(1-k1);
  logl = -0.5*ln(2*pi) - 0.5*ln(k2) - 0.5*e^2/k2;

  retp(logl);
endp;

output file = ou1.out reset;

_tsm_parnm = "a"|"b"|"sigma";
{theta,stderr,Mcov,Logl} = TD_ml(&ml,0.5|0.5|-0.5);

output off;

Total observations:                                   999
Usable observations:                                  999
```

```
Number of parameters to be estimated:                          3
Degrees of freedom:                                          996
Value of the maximized log-likelihood function:       2569.86206


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------
a               0.800286        0.007852       101.915066        0.000000
b               0.090854        0.007695        11.806302        0.000000
sigma           0.060774        0.001360        44.692558        0.000000

Covariance matrix: inverse of the negative Hessian.
```

8.2.2 Estimation GMM

Nous avons

$$\begin{cases} E_{t-1}\left[\varepsilon_t\right] = 0 \\ E_{t-1}\left[\varepsilon_t^2 - \sigma^2 \frac{1-e^{-2ah}}{2a}\right] = 0 \\ E_{t-1}\left[\varepsilon_t y_{t-1}\right] = 0 \end{cases}$$

avec

$$\varepsilon_t = y_t - e^{-ah} y_{t-1} - b\left(1 - e^{-ah}\right)$$

```
new;
library tsm,optmum;
TSMset;

load x[1000,1] = ou.dat;

h = 0.1;

proc Moments(theta);
  local a,b,sigma,k1,k2,e,M1,M2,M3,M;

  a = theta[1]; b = theta[2]; sigma = theta[3];
  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);
  e = x[2:1000]-k1.*x[1:999]-b*(1-k1);

  M1 = e;
  M2 = e^2-k2;
  M3 = e.*x[1:999];
  M = M1~M2~M3;

  retp(M);
endp;


output file = ou2.out reset;

_tsm_parnm = "a"|"b"|"sigma";
sv = rndu(3,1);
{theta,stderr,Mcov,Qmin} = gmm(&Moments,sv);

output off;

Total observations:                                          999
Usable observations:                                         999
Number of parameters to be estimated:                          3
Degrees of freedom:                                          996
Number of moment conditions:                                   3
```

```
Value of the criterion function:                    0.00000


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------
a               0.800286        0.009605        83.322344        0.000000
b               0.090854        0.007697        11.803929        0.000000
sigma           0.060774        0.001348        45.074976        0.000000
```

8.3 Estimation des paramètres d'une équation différentielle stochastique

Considérons l'équation différentielle stochastique suivante

$$\left\{ \begin{array}{rcl} dX(t) & = & \mu(t, X(t))\, dt + \sigma(t, X(t))\, dW(t) \\ X(t_0) & = & x_0 \end{array} \right. \tag{28}$$

La version discrétisée de Taylor à l'ordre 0.5 de cette EDS est

$$\left\{ \begin{array}{rcl} X_{t+1} & = & X_t + \mu(t, X_t)\, h + \varepsilon_{t+1} \\ \varepsilon_{t+1} & \sim & \mathcal{N}\left(0, \sigma^2(t, X_t)\, h\right) \end{array} \right.$$

avec h le pas de discrétisation.

8.3.1 Estimation ML naive

Références

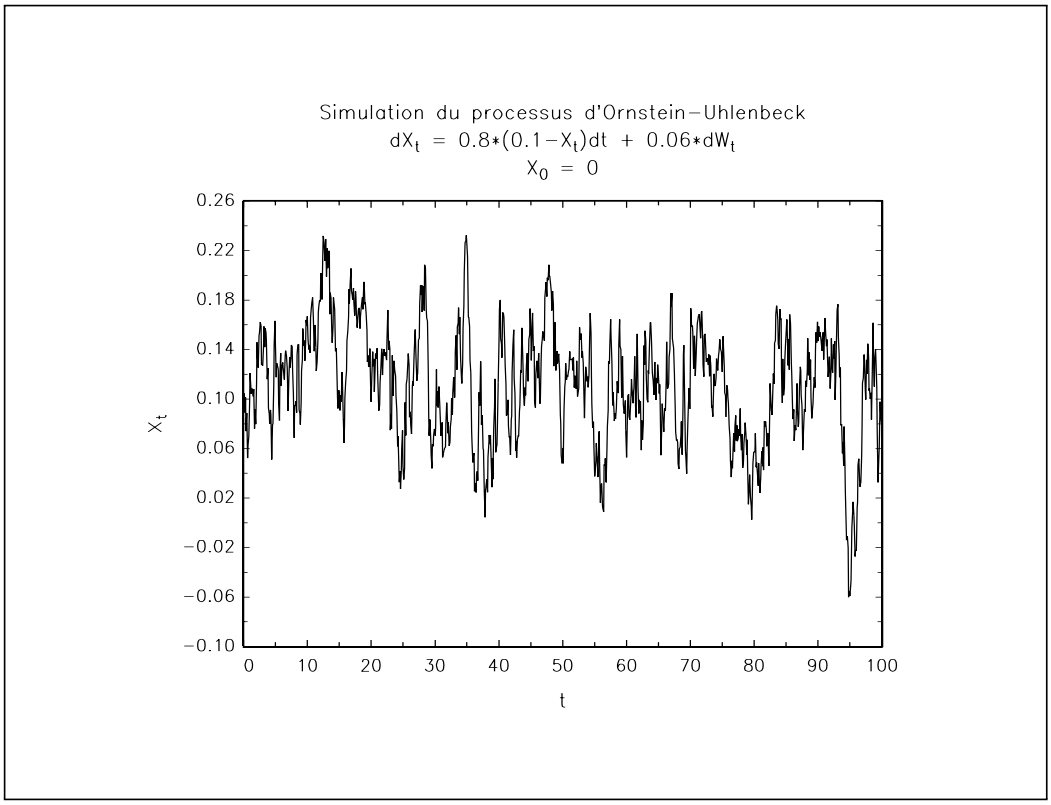
Broze, L, O. Scaillet et J-M. Zakoïan [1993], Testing for continuous-time models of the short-term interest rate, Colloque organisé par le Groupe Caisse Des Dépôts et le G.R.E.Q.E. les 2,3 et 4 septembre 1993 à Paris : *Dynamiques des marchés financiers et prévisions*

Broze, Scaillet et Zakoian [1993] proposent d'estimer le vecteur θ des paramètres de l'équation (28) par maximum de vraisemblance. Soit la série temporelle $\{x_t, t = 1, \ldots, T\}$. La log-vraisemblance pour l'observation t est alors défini par

$$\ell_t = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln \sigma^2(t, x_t)\, h - \frac{1}{2}\frac{[x_{t*1} - x_t - \mu(t, x_t)\, h]^2}{\sigma^2(t, x_t)\, h}$$

Considérons une estimation ML naive des coefficients d'un processus d'Ornstein-Ulhenbeck. Le programme suivant **eds.sim** permet de simuler une réalisation avec $a = 0.8$, $b = 0.1$ et $\sigma = 0.06$ (voir le graphique 40). La simulation est obtenue à partir d'une discrétisation exacte du processus d'Ornstein-Uhlenbeck.

```
new;
library pgraph;

rndseed 123;

x0 = 0.1;
a = 0.8; b = 0.1; sigma = 0.06;
h = 0.1;
Nobs = 1000;

k1 = exp(-a*h);
k2 = (1-exp(-2*a*h))/(2*a);

x = recserar(b*(1-k1) + (sigma*sqrt(k2))*rndn(Nobs,1),x0,k1);

save eds = x;

graphset;
  _pdate = ""; _pnum = 2;
  title("Simulation du processus d'Ornstein-Uhlenbeck"\
        "\LdX]t[ = 0.8*(0.1-X]t[)dt + 0.06*dW]t["\
```



Graphique 40

```
      "\LX]0[ = 0");
  xlabel("t");
  ylabel("X]t[");
  graphprt("-c=1 -cf=eds.eps -w=5");
  xy(seqa(0,h,Nobs),x);
```

Dans le programme suivant, nous estimons le vecteur θ par les méthodes ML exacte et ML naive. Les deux méthodes donnent des résultats très proches.

```
new;
library tsm,optmum;
TSMset;

load x = eds;

Nobs = rows(x);
h = 0.1;

proc ml1(theta);
  local a,b,sigma,k1,k2,e,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];
  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);
  e = x[2:1000]-k1.*x[1:999]-b*(1-k1);
  logl = -0.5*ln(2*pi) - 0.5*ln(k2) - 0.5*e^2/k2;

  retp(logl);
endp;

proc ml2(theta);
```

```
  local a,b,sigma,et,vt,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];

  et = x[2:Nobs] - x[1:Nobs-1] - a*(b-x[1:Nobs-1])*h;
  vt = (sigma^2)*h;

  logl = -0.5*ln(2*pi) - 0.5*ln(vt) - 0.5*et^2./vt;

  retp(logl);
endp;


output file = eds1.out reset;

_tsm_parnm = "a"|"b"|"sigma";

sv = 0.5|0.5|0.5;
{theta,stderr,Mcov,Logl} = TD_ml(&ml1,sv);

print; print;
{theta,stderr,Mcov,Logl} = TD_ml(&ml2,theta);

output off;


Total observations:                                   999
Usable observations:                                  999
Number of parameters to be estimated:                   3
Degrees of freedom:                                   996
Value of the maximized log-likelihood function:  2553.48649


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
a               0.884219        0.139306         6.347321        0.000000
b               0.111384        0.007019        15.868924        0.000000
sigma           0.062031        0.001450        42.789616        0.000000

Covariance matrix: inverse of the negative Hessian.



Total observations:                                   999
Usable observations:                                  999
Number of parameters to be estimated:                   3
Degrees of freedom:                                   996
Value of the maximized log-likelihood function:  2553.48649


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
a               0.846238        0.127794         6.621914        0.000000
b               0.111385        0.007023        15.859257        0.000000
sigma           0.059386        0.001329        44.700154        0.000000

Covariance matrix: inverse of the negative Hessian.
```

Cependant, l'estimation ML naive peut conduire à des résultats surprenants, s'il existe des biais de discrétisation trop important (dans l'exemple suivant, h est égal à 2).

```
new;
library tsm,optmum;
TSMset;

rndseed 123;

x0 = 0.1;
a = 0.8; b = 0.1; sigma = 0.06;
h = 2;
Nobs = 1000;

k1 = exp(-a*h);
k2 = (1-exp(-2*a*h))/(2*a);

x = recserar(b*(1-k1) + (sigma*sqrt(k2))*rndn(Nobs,1),x0,k1);

proc ml1(theta);
  local a,b,sigma,k1,k2,e,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];
  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);
  e = x[2:1000]-k1.*x[1:999]-b*(1-k1);
  logl = -0.5*ln(2*pi) - 0.5*ln(k2) - 0.5*e^2/k2;

  retp(logl);
endp;

proc ml2(theta);
  local a,b,sigma,et,vt,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];

  et = x[2:Nobs] - x[1:Nobs-1] - a*(b-x[1:Nobs-1])*h;
  vt = (sigma^2)*h;

  logl = -0.5*ln(2*pi) - 0.5*ln(vt) - 0.5*et^2./vt;

  retp(logl);
endp;


output file = eds2.out reset;

_tsm_parnm = "a"|"b"|"sigma";

sv = 0.5|0.5|0.5;
{theta,stderr,Mcov,Logl} = TD_ml(&ml1,sv);

print; print;
sv = 0.5|0.5|0.5;
{theta,stderr,Mcov,Logl} = TD_ml(&ml2,sv);

output off;
```

```
Total observations:                                   999
Usable observations:                                  999
Number of parameters to be estimated:                   3
Degrees of freedom:                                   996
Value of the maximized log-likelihood function:  1619.36158
```

```
Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
a               0.771432        0.072321        10.666739        0.000000
b               0.102787        0.001925        53.394194        0.000000
sigma           0.060827        0.002785        21.838342        0.000000

Covariance matrix: inverse of the negative Hessian.



Total observations:                                    999
Usable observations:                                   999
Number of parameters to be estimated:                    3
Degrees of freedom:                                    996
Value of the maximized log-likelihood function:  1619.36158


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
a               0.393116        0.015457        25.432821        0.000000
b               0.102787        0.001925        53.393613        0.000000
sigma           0.033827        0.000757        44.698560        0.000000

Covariance matrix: inverse of the negative Hessian.
```

8.3.2 Estimation GMM naïve

Références

Chan, K.C., G.A. Karolyi, F.A. Longstaff et A.B. Sanders [1992], An empirical comparison of alternative models of short-term interest rate, *Journal of Finance*, **47**, 1209-1227

Chan, Karolyi, Longstaff et Sanders [1992] proposent d'estimer le vecteur θ des paramètres de l'équation (28) par la méthode des moments généralisés. Nous avons

$$\begin{cases} E_t\left[\varepsilon_{t+1}\right] = 0 \\ E_t\left[\varepsilon_{t+1}^2 - \sigma^2\left(t, x_t\right) h\right] = 0 \\ E_t\left[\varepsilon_{t+1} x_t\right] = 0 \\ E_t\left[\left(\varepsilon_{t+1}^2 - \sigma^2\left(t, x_t\right) h\right) x_t\right] = 0 \\ \ldots \end{cases}$$

avec

$$\varepsilon_{t+1} = x_{t*1} - x_t - \mu\left(t, x_t\right) h$$

```
new;
library tsm,optmum;
TSMset;

load x = eds;

Nobs = rows(x);
h = 0.1;

proc CKLS(theta);
  local a,b,sigma,et,vt,M1,M2,M3,M4,M;

  a = theta[1]; b = theta[2]; sigma = theta[3];

  et = x[2:Nobs] - x[1:Nobs-1] - a*(b-x[1:Nobs-1])*h;
  vt = (sigma^2)*h;
```

```
  M1 = et;
  M2 = et.*x[1:Nobs-1];
  M3 = et^2 - vt;
  M4 = (et^2 - vt).*x[1:Nobs-1];

  M = M1~M2~M3~M4;

  retp(M);
endp;

output file = eds3.out reset;

_tsm_parnm = "a"|"b"|"sigma";
sv = 0.5|0.5|-0.5;
{theta,stderr,Mcov,Qmin} = gmm(&CKLS,sv);

output off;
```

```
Total observations:                                   999
Usable observations:                                  999
Number of parameters to be estimated:                   3
Degrees of freedom:                                   996
Number of moment conditions:                            4
Value of the criterion function:                  0.00066


Hansen's test of overidentifying restrictions:    0.65521
p-value:                                          0.41826
```

Parameters	estimates	std.err.	t-statistic	p-value
a	0.851535	0.122006	6.979431	0.000000
b	0.110835	0.006950	15.948042	0.000000
sigma	0.059324	0.001312	45.226786	0.000000

8.3.3 Estimation par Inférence Indirecte

Références

Gouriéroux, C., A. Monfort et E. Renault [1993], Indirect inference, *Journal of Applied Econometrics*, **8**, S85-S118

Gouriéroux, Monfort et Renault [1993] suggèrent d'utiliser l'inférence indirecte pour estimer les paramètres d'une équation différentielle stochastique. L'idée principale est de remplacer le modèle initial $\mathcal{M}$ par un autre modèle $\mathcal{M}^\star$ plus simple à estimer par maximum de vraisemblance. Mais dans ce cas, l'estimateur n'est pas en général consistant. Soient θ le vecteur des paramètres du modèle $\mathcal{M}$ et $\theta^\star$ le vecteur des paramètres du modèle $\mathcal{M}^\star$. Le processus issu de $\mathcal{M}$ pour une valeur de θ égale à θ_0 ne possède pas les mêmes propriétés que celui issu de $\mathcal{M}^\star$ pour une valeur de $\theta^\star$ égale à θ_0. Mais on peut supposer qu'il existe une valeur θ_1 prise par θ qui permet d'obtenir un processus équivalent pour $\theta^\star$ égale à θ_0. Les auteurs proposent l'algorithme suivant pour obtenir l'estimateur indirect (EI) :

1. Soit y l'échantillon issu du modèle $\mathcal{M}$. On calcule l'estimateur du maximum de vraisemblance à partir du modèle $\mathcal{M}^\star$
$$\hat{\theta}_{\mathrm{ML}} = \arg\max \sum_{t=1}^{T} \ell^\star\left(y_t; \theta\right)$$
2. On simule S trajectoires du modèle $\mathcal{M}$, que nous notons $y^{(s)}$ avec $s = 1, \ldots, S$.

3. Pour chaque trajectoire simulée, nous calculons

$$\hat{\theta}_{\mathrm{ML}}^{(s)} = \arg\max \sum_{t=1}^{T} \ell^{\star}\left(y_t^{(s)};\theta\right)$$

4. L'estimateur indirect est défini par

$$\hat{\theta}_{\mathrm{EI}} = \underset{\theta\in\Theta}{\arg\min} \left[\hat{\theta}_{\mathrm{ML}} - \frac{1}{S}\sum_{s=1}^{S}\hat{\theta}_{\mathrm{ML}}^{(s)}\right]^{\top} \Upsilon \left[\hat{\theta}_{\mathrm{ML}} - \frac{1}{S}\sum_{s=1}^{S}\hat{\theta}_{\mathrm{ML}}^{(s)}\right]$$

où Υ est une matrice p.d.s..

Nous avons implémenté l'algorithme décrit ci-dessus dans la procédure `Inference_Indirecte`. Le modèle $\mathcal{M}$ correspond à l'équation différentielle stochastique

$$\left\{\begin{array}{rcl} dX(t) & = & \mu(\theta, X(t))\,dt + \sigma(\theta, X(t))\,dW(t) \\ X(t_0) & = & x_0 \end{array}\right.$$

et le modèle $\mathcal{M}^{\star}$ correspond à la version discrétisée de Taylor à l'ordre 0.5 de cette EDS

$$\left\{\begin{array}{rcl} X_{t+1} & = & X_t + \mu(\theta, X_t)\,h + \varepsilon_{t+1} \\ \varepsilon_{t+1} & \sim & \mathcal{N}\left(0, \sigma^2(\theta, X_t)\,h\right) \end{array}\right.$$

Pour estimer le vecteur des paramètres θ, l'utilisateur de la procédure doit définir les fonctions $\mu(\theta, x)$ et $\sigma(\theta, x)$. Il doit aussi préciser le pas de discrétisation h, ainsi que la matrice Υ.

Inference_Indirecte

■ **Objectif**
Estimation des paramètres d'une équation différentielle stochastique par inférence indirecte.

■ **Format**
theta_EI = Inference_Indirecte(&mu,&sigma,h,S,&simul,seed,sv,theta_ML,Upsilon);

■ **Entrées**

&mu	pointeur d'une procédure qui calcule la fonction $\mu(\theta, x)$.
&sigma	pointeur d'une procédure qui calcule la fonction $\sigma(\theta, x)$.
h	scalaire, pas de discrétisation h.
&simul	pointeur d'une procédure qui simule le processus $y^{(s)}$.
seed	scalaire, compteur initial pour la génération des nombres aléatoires.
sv	vecteur $k \times 1$, valeurs de départ pour l'algorithme d'optimisation.
theta_ML	vecteur $k \times 1$, estimation $\hat{\theta}_{\mathrm{ML}}$ du maximum de vraisemblance.
Upsilon	matrice $k \times k$, matrice des poids Υ.

■ **Sortie**

theta_EI	vecteur $k \times 1$, estimateur indirect $\hat{\theta}_{\mathrm{EI}}$.

ei.dec

```
declare matrix _EI_data;
declare matrix _EI_mu;
declare matrix _EI_sigma;
declare matrix _EI_h;
declare matrix _EI_seed;
declare matrix _EI_S;
declare matrix _EI_S;
declare matrix _EI_theta;
declare matrix _EI_Upsilon;
declare matrix _EI_simul;
```

ei.ext

```
external matrix _EI_data;
external matrix _EI_mu;
external matrix _EI_sigma;
external matrix _EI_h;
external matrix _EI_seed;
external matrix _EI_S;
external matrix _EI_S;
external matrix _EI_theta;
external matrix _EI_Upsilon;
external matrix _EI_simul;
```

ei.src

```
proc (1) = Inference_Indirecte(mu,sigma,h,S,simul,seed,sv,theta,Upsilon);
  local simul:proc,mu:proc,sigma:proc;
  local theta_EI,fmin,grd,retcode;

  _EI_theta = theta;
  _EI_Upsilon = Upsilon;
  _EI_S = S;
  _EI_simul = &simul;
  _EI_seed = seed;
  _EI_h = h;
  _EI_mu = &mu;
  _EI_sigma = &sigma;

  {theta_EI,fmin,grd,retcode} = optmum(&_EI_function,sv);

  retp(theta_EI);
endp;

proc _EI_ML(theta);
  local h,xt,Nobs,mu,sigma,et,vt,logl;

  h = _EI_h;

  xt = _EI_data;
  Nobs = rows(xt);

  mu = _EI_mu;
  sigma = _EI_sigma;
  local mu:proc,sigma:proc;

  et = xt[2:Nobs] - xt[1:Nobs-1] - mu(theta,xt[1:Nobs-1])*h;
  vt = sigma(theta,xt[1:Nobs-1])^2*h;

  logl = -0.5*ln(2*pi) - 0.5*ln(vt) - 0.5*et^2/vt;

  retp(-sumc(logl));
endp;

proc _EI_function(theta_EI);
  local theta,Upsilon,seed,S,simul;
  local ys,i,theta_s,fmin,grd,retcode,coeff,dev;

  theta = _EI_theta;
  Upsilon = _EI_Upsilon;

  seed = _EI_seed;
  rndseed seed;
```

```
  S = _EI_S;

  simul = _EI_simul;
  local simul:proc;

  /* Etape 2 de l'algorithme */

  ys = {};
  i = 1;
  do until i > S;
    ys = ys~simul(theta_EI);
    i = i + 1;
  endo;

  /* Etape 3 de l'algorithme */

  __output = 0;

  coeff = {};

  i = 1;
  do until i > S;
    _EI_data = ys[.,i];
    {theta_s,fmin,grd,retcode} = optmum(&_EI_ML,theta);
    coeff = coeff|theta_s';
    i = i + 1;
  endo;

  __output = 2;

  dev = theta - meanc(coeff);

  retp(dev'Upsilon*dev);
endp;
```

Considérons un exemple d'utilisation de la procédure `Inference_Indirecte`. Dans le programme suivant, nous estimons les paramètres d'un processus d'Ornstein-Uhlenbeck. Les données sont observées avec $h = 1$.

```
new;
library tsm,finance,optmum;
TSMset;

rndseed 123;

/* Simulation d'un processus */

x0 = 0.1; a = 0.8; b = 0.1; sigma = 0.06;
h = 0.1;
Nobs = 10000;

k1 = exp(-a*h); k2 = (1-exp(-2*a*h))/(2*a);
data = recserar(b*(1-k1) + (sigma*sqrt(k2))*rndn(Nobs,1),x0,k1);

/* Observations discretes avec h = 1 */

data = data[seqa(1,10,1000)];
h = 1;

/* Estimation Naive */

Nobs = 1000;
```

```
proc ml(theta);
  local a,b,sigma,et,vt,logl;

  a = theta[1]; b = theta[2]; sigma = theta[3];

  et = data[2:Nobs] - data[1:Nobs-1] - a*(b-data[1:Nobs-1])*h;
  vt = (sigma^2)*h;

  logl = -0.5*ln(2*pi) - 0.5*ln(vt) - 0.5*et^2./vt;

  retp(logl);
endp;

{theta_ML,stderr,Mcov,Logl} = TD_ml(&ml,0.5|0.5|0.5);


/* Estimation Indirecte */

proc OU_mu(theta,x);
  local a,b;
  a = theta[1];
  b = theta[2];
  retp(a*(b-x));
endp;

proc OU_sigma(theta,x);
  local sigma;
  sigma = theta[3];
  retp(sigma);
endp;

proc (1) = rnd(theta);
  local x0,a,b,sigma,Nobs,h,k1,k2,x;

  x0 = data[1];
  a = theta[1];
  b = theta[2];
  sigma = theta[3];
  Nobs = 10000;
  h = 0.1;

  k1 = exp(-a*h); k2 = (1-exp(-2*a*h))/(2*a);
  x = recserar(b*(1-k1) + (sigma*sqrt(k2))*rndn(Nobs,1),x0,k1);

  x = x[seqa(1,10,1000)];

  retp(x);
endp;

sv = 0.8|0.1|0.06;
theta_EI =
   Inference_Indirecte(&OU_mu,&OU_sigma,1,3,&rnd,123,sv,theta_ML,eye(3));

output file = eds4.out reset;

_tsm_parnm = "a"|"b"|"sigma";

print; print;
print "              Vraies valeurs    Est. Naive   Inference Indirecte";
```

```
print;
call printfmt(_tsm_parnm~sv~theta_ML~theta_EI,0~1~1~1);

output off;
```

	Vraies valeurs	Est. Naive	Inference Indirecte
a	0.8	0.54192082	0.75532948
b	0.1	0.10011594	0.10033718
sigma	0.06	0.041092438	0.059381831

L'inférence indirecte, dans sa version originale, est cependant difficile à mettre en œuvre. En effet, considérons par exemple l'équation différentielle stochastique

$$\begin{cases} dX_t &= a\left(b - X_t\right)\, dt + \sigma X_t^{\frac{1}{5}}\, dW_t \\ X\left(t_0\right) &= x_0 \end{cases}$$

En utilisant le théorème de Yamada et Watanabe, on montre qu'il existe une solution unique. Mais nous ne connaissons pas la forme symbolique f du processus de diffusion $X_t = f\left(t\right)$. Dans ce cas, il est impossible d'exécuter l'étape 2 de l'algorithme. C'est pourquoi nous devons modifier la procédure d'estimation et consider un modèle $\mathcal{M}^\bullet$ approximant le vrai modèle $\mathcal{M}$:

1. Soit y l'échantillon issu du modèle $\mathcal{M}$. On calcule l'estimateur du maximum de vraisemblance à partir du modèle $\mathcal{M}^\star$
$$\hat{\theta}_{\mathrm{ML}} = \arg\max \sum_{t=1}^{T} \ell^\star\left(y_t; \theta\right)$$
2. On simule S trajectoires du modèle $\mathcal{M}^\bullet$, que nous notons $y^{(s)}$ avec $s = 1, \ldots, S$.
3. Pour chaque trajectoire simulée, nous calculons
$$\hat{\theta}_{\mathrm{ML}}^{(s)} = \arg\max \sum_{t=1}^{T} \ell^\star\left(y_t^{(s)}; \theta\right)$$
4. L'estimateur indirect est défini par
$$\hat{\theta}_{\mathrm{EI}} = \underset{\theta \in \Theta}{\arg\min} \left[\hat{\theta}_{\mathrm{ML}} - \frac{1}{S}\sum_{s=1}^{S} \hat{\theta}_{\mathrm{ML}}^{(s)}\right]^\top \Upsilon \left[\hat{\theta}_{\mathrm{ML}} - \frac{1}{S}\sum_{s=1}^{S} \hat{\theta}_{\mathrm{ML}}^{(s)}\right]$$
où Υ est une matrice p.d.s..

Il n'est pas nécessaire de modifier la procédure `Inference_Indirecte` pour implémenter cet algorithme. Il suffit seulement de définir différemment la procédure de simulation, qui correspond maintenant au modèle $\mathcal{M}^\bullet$.

9 Gestion de grosses bases de donnees

9.1 La construction d'une base de données GAUSS à partir d'un fichier ASCII : l'exemple de la base HFDF93 d'Olsen & Associates

HFDF93 de la société suisse Olsen & Associates est une base de données en temps continu sur les cours de change. Le fichier ASCII *dem&usd.dat* contient l'évolution du cours de change DEM/USD pour la période allant du 1er octobre 1992 au 30 septembre 1993. Ce fichier est cependant très difficile à exploiter, puisqu'il contient plus de 1 300 000 observations et représente 67.3 Mo d'information. Le fichier *dem&usd.dat* se présente de la façon suivante :

```
;(***************** P R O P R I E T A R Y   F I L E *********************)
;(* This file is part of the Olsen & Associates data distribution: HFDF93 *)
;(* The entire data distribution as well as this file are the property of *)
;(* Olsen & Associates and are proprietary and confidential. It is not    *)
;(* permitted to disclose, copy or distribute this file, neither wholly   *)
```

```
;(* nor in part, except by written permission of Olsen & Associates.      *)
;(* Olsen & Associates shall not be liable for any loss or damage         *)
;(* whatsoever caused arising directly or indirectly in connection with   *)
;(* the use of the data set.                                              *)
;(************************************************************************)
;
;                          Olsen & Associates
;              Research Institute for Applied Economics.
;           Seefeldstrasse 233, CH-8008 Zurich, Switzerland.
;                          E-mail: hfdf@olsen.ch
;                        Telephone: 41-1-386-48-48
;
; Tick extraction for DEM^USD
; ---------------------------
;
; Filtering: O&A standard filter for historical tests
;
;   date     time       price    country      filter
;                                    city     Good (1) or Bad (0)
;CCYY-MM-DD  (GMT)    bid    ask         bank
1992-10-01 00:00:14 1.4116 1.4121 392 01 0058 1
1992-10-01 00:00:54 1.4108 1.4118 036 02 0130 1
...
```

GAUSS possède de nouvelles commandes pour la gestion des fichiers I/O : `close`, `closeall`, `eof`, `fcheckerr`, `fclearerr`, `fflush`, `fgets`, `fgetsa`, `fgetsat`, `fgetst`, `fopen`, `fputs`, `fputst`, `fseek`, `fstrerror`, `ftell`. Ces nouvelles commandes permettent notamment de manipuler de gros fichiers ASCII. Le programme suivant construit à partir du fichier *dem&usd.dat* un fichier ASCII *dem&usd.asc* contenant les informations suivantes

Annee	Mois	Jour	Heure (en secondes)	cours BID

Chaque ligne d'observation du fichier *dem&usd.dat* est traitée comme une chaîne de caractère. Nous manipulons ensuite cette chaîne de caractère afin d'extraire et/ou construire les informations qui nous intéressent. Nous pouvons ensuite créer la nouvelle base de données (`call fputst(fichier2,str);`).

```
new;

fichier1 = fopen("dem&usd.dat","r");
fichier2 = fopen("dem&usd.asc","w");

i = 1;
do until i > 50;
  str = fgets(fichier1,100);
  i = i + 1;
endo;

blanc = "   ";

i = 1;
do until eof(fichier1);
  str = fgets(fichier1,60);

  {date_,str} = token(str);
  {heure,str} = token(str);
  {bid,str} = token(str);
  {ask,str} = token(str);
  {pays,str} = token(str);
  {ville,str} = token(str);
  {banque,str} = token(str);
  {filtre,str} = token(str);
```

```
  annee = strsect(date_,1,4);
  mois = strsect(date_,6,2);
  jour = strsect(date_,9,2);

  heures = strsect(heure,1,2);
  minutes = strsect(heure,4,2);
  secondes = strsect(heure,7,2);

  temps = 3600*stof(heures) + 60*stof(minutes) + stof(secondes);
  temps = ftos(temps,"%*.*lf",6,0);

  str = annee $+ blanc $+ mois $+ blanc $+ jour $+ blanc $+ temps;
  str = str $+ blanc $+ bid;

  call fputst(fichier2,str);

endo;

closeall;
```

Le fichier *dem&usd.asc* se présente de la façon suivante :

```
1992   10   01      300   1.4113
1992   10   01      314   1.4103
1992   10   01      326   1.4110
1992   10   01      340   1.4110
1992   10   01      348   1.4105
...
```

Nous pouvons aussi utiliser les nouvelles commandes de gestion des fichiers I/O pour créer une base de données **GAUSS**. L'avantage de manipuler une base de données au format **GAUSS** est la possibilité de lecture selective des observations et des variables. Nous pouvons par exemple sélectionner les cours du Lundi, construire les rendements horaires, minutes, 30 secondes ou 15 secondes, ne considérer que les cours cotés entre 11 heures et 13 heures pour le mois de janvier 1993, etc.

```
new;

fichier1 = fopen("dem&usd.dat","r");

let vname = jour heure bid ask filtre;
create fichier2 = forex with ^vname,5,4;

i = 1;
do until i > 50;
  str = fgets(fichier1,100);
  i = i + 1;
endo;

date_de_reference = 1992|01|01;

i = 1;
do until eof(fichier1);
  str = fgets(fichier1,60);

  {date_,str} = token(str);
  {heure,str} = token(str);
  {bid,str} = token(str);
  {ask,str} = token(str);
  {pays,str} = token(str);
  {ville,str} = token(str);
  {banque,str} = token(str);
```

```
  {filtre,str} = token(str);

  annee = strsect(date_,1,4);
  mois = strsect(date_,6,2);
  jour = strsect(date_,9,2);

  heures = strsect(heure,1,2);
  minutes = strsect(heure,4,2);
  secondes = strsect(heure,7,2);

  jour = etdays(date_de_reference,stof(annee)|stof(mois)|stof(jour));
  heure = 3600*stof(heures) + 60*stof(minutes) + stof(secondes);
  bid = stof(bid);
  ask = stof(ask);
  filtre = stof(filtre);

  x = jour~heure~bid~ask~filtre;
  call writer(fichier2,x);

endo;

closeall;
```

Il faut 10 minutes pour construire la base de données **GAUSS** précédente avec un Pentium 100. La lecture de la base de données est ensuite très rapide (environ 1 minute pour la base précédente, qui contient 1 300 000 observations).

9.2 La gestion d'une base de données : l'exemple de la base HMD de la Société de Bourse Française