GAUSS spécial Finance II

Thierry Roncalli

Université Montesquieu-Bordeaux IV, Avenue Léon Duguit, 33608 Pessac Cedex

e-mail : roncalli@montesquieu.u-bordeaux.fr
web : http://www.montesquieu.u-bordeaux.fr/u/roncalli

Table des matières

1 Présentation de GAUSS 3.2.32

1.1 Introduction

GAUSS est un langage de programmation numérique et graphique. Ses points forts sont les suivants :

1. GAUSS est un langage matriciel vectorisé.
2. GAUSS possède de nombreuses fonctions mathématiques.
3. Les variables sont déclarées globales par défaut.
4. La gestion des fichiers sources est largement facilitée par l'utilisation de bibliothèques de procédures.
5. GAUSS possède de nombreuses bibliothèques et modules spécialisés en Econométrie et en Finance.

Voici quelques programmes qui illustrent les spécificités de GAUSS.

1.1.1 Programme n°1

Considérons une procédure de dénombrement de chemins binomiaux d'ordre N. Par exemple, pour N égal à 2, nous avons 4 chemins possibles

$$(1,1)$$
$$(1,0)$$
$$(0,1)$$
$$(0,0)$$

Le cas général n'est pas très facile à programmer. Néanmoins, l'utilisation du produit de Kronecker `.*.` et d'une procédure de réduction de matrice `trimr` permet d'obtenir un code assez court. Il suffit de remarquer que nous pouvons employer la séquence suivante

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{matrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{matrix}$$

La procédure

```
proc (1) = _denombrementCheminsBinomiaux(Narbitrage);
  local x,cn,xi,i;

  x = zeros(2^Narbitrage,Narbitrage);
  cn = 2^(Narbitrage-1);
```

```
  xi = ones(cn,1)|zeros(cn,1);
  x[.,1] = xi;
  i = 2;
  do until i > Narbitrage;
    cn = cn/2;
    xi = trimr(xi,cn,cn);
    x[.,i] = ones(2^(i-1),1).*.xi;
    i = i + 1;
  endo;

  retp(x);
endp;
```

L'exemple

```
new;

#include option1.src;

x = _denombrementCheminsBinomiaux(4);

output file = option1.out reset;

call printfmt(x,1);

output off;
```

```
1    1    1    1
1    1    1    0
1    1    0    1
1    1    0    0
1    0    1    1
1    0    1    0
1    0    0    1
1    0    0    0
0    1    1    1
0    1    1    0
0    1    0    1
0    1    0    0
0    0    1    1
0    0    1    0
0    0    0    1
0    0    0    0
```

1.1.2 Programme n°2

Nous pouvons utiliser la procédure précédente pour construire les chemins binomiaux du modèle de valorisation des actifs contingents de type Cox, Ross et Rubinstein. Il est intéressant de noter qu'une seule boucle suffit pour construire les valeurs du sous-jacent selon l'état de la nature.

La procédure

```
proc (3) = _constructionCheminsBinomiaux(S0,sigma,tau,r,delta,Narbitrage);
  local u,d,pi_,fa;
  local x,S,Si,i,xi,cn,j,prob;

  u = exp(sigma*sqrt(tau/Narbitrage));    /* coefficient de hausse   */
  d = 1/u;                                /* coefficient de baisse   */
  pi_ = (exp((r-delta)*tau/Narbitrage)-d)
        /(u-d);                           /* probabilite binomiale   */
  fa = exp(-r*tau);                       /* facteur d'actualisation */

  x = _denombrementCheminsBinomiaux(Narbitrage);
  S = zeros(2^Narbitrage,Narbitrage+1);
  S[.,1] = S0*ones(2^Narbitrage,1);
  Si = S0;
  i = 1;
  do until i > Narbitrage;
    xi = x[.,i];
    cn = xi .== 1;
    Si = Si .* (u.*cn + d.*(1-cn));
    S[.,i+1] = Si;
    i = i + 1;
  endo;

  j = sumc(x');
  prob = (pi_^j).*((1-pi_)^(Narbitrage-j));

  retp(S,prob,fa);
endp;
```

L'exemple

```
new;

#include option1.src;
#include option2.src;

{S,prob,fa} = _constructionCheminsBinomiaux(100,0.20,90/365,0.08,0,3);

output file = option2.out reset;

print "                                    Chemins binomiaux"\
      "                   |   probabilite";
print;

call printfmt(S~prob,1);

output off;
```

	Chemins binomiaux			probabilite
100	105.90139	112.15105	118.76952	0.160249
100	105.90139	112.15105	105.90139	0.13477922
100	105.90139	100	105.90139	0.13477922
100	105.90139	100	94.427464	0.11335757
100	94.427464	100	105.90139	0.13477922
100	94.427464	100	94.427464	0.11335757
100	94.427464	89.16546	94.427464	0.11335757
100	94.427464	89.16546	84.196684	0.095340648

1.1.3 Programme n°3

Nous pouvons utiliser la procédure `_constructionCheminsBinomiaux` pour valoriser les options exotiques, par exemple les options asiatiques ou lookback. Les trois procédures suivantes sont construites sur le même modèle :

1. Construction de l'ensemble des chemins binomiaux ;
2. Détermination de la fonction de payoff pour tous les états de la nature ;
3. Calcul du prix de l'option en utilisant la mesure de probabilité risque neutre.

L'emploi des commandes matricielles et des opérateurs $E \times E$ permet de rendre les procédures très lisibles.

Quelques commentaires sur le code :

- `S` est une matrice de dimension $2^N \times (N+1)$ avec N le nombre d'arbitrages du modèle CRR. Chaque ligne de la matrice correspond à une trajectoire du sous-jacent sous la mesure de probabilité neutre au risque.
- `ST = S[.,Narbitrage+1];` permet de sélectionner la $N+1$-ième colonne de la matrice `S`. `ST` est donc un vecteur de dimension 2^N comprenant les valeurs du sous-jacent à l'échéance de l'option pour l'ensemble des états de la nature.
- `Sm = minc(S');` correspond aux valeurs prises par la fonction $\min_{t_0 \leq t \leq T} S(t)$.
- Nous obtenons la fonction de payoff $G(T) = \max\left(0, S(T) - \min_{t_0 \leq t \leq T} S(t)\right)$ en employant les opérateurs élément par élément `.*` et `.>`.

La procédure

```
proc (2) = FloatingStrikeOption(S0,sigma,tau,r,delta,Narbitrage);
  local S,prob,fa;
  local ST,Sm,Gcall,Gput;
  local floatCall,floatPut;

  {S,prob,fa} = _constructionCheminsBinomiaux(S0,sigma,tau,r,delta,Narbitrage);
```

```
  ST = S[.,Narbitrage+1];
  Sm = meanc(S');

  Gcall = ST - Sm;
  Gcall = Gcall .* (Gcall .> 0);

  Gput = Sm - ST;
  Gput = Gput .* (Gput .> 0);

  floatCall = fa*(prob'Gcall);
  floatPut = fa*(prob'Gput);

  retp(floatCall,floatPut);
endp;


proc (2) = FixedStrikeOption(S0,K,sigma,tau,r,delta,Narbitrage);
  local S,prob,fa;
  local Sm,Gcall,Gput;
  local fixedCall,fixedPut;

  {S,prob,fa} = _constructionCheminsBinomiaux(S0,sigma,tau,r,delta,Narbitrage);

  Sm = meanc(S');

  Gcall = Sm - K;
  Gcall = Gcall .* (Gcall .> 0);

  Gput = K - Sm;
  Gput = Gput .* (Gput .> 0);

  fixedCall = fa*(prob'Gcall);
  fixedput = fa*(prob'Gput);

  retp(fixedCall,fixedPut);
endp;


proc (2) = LookBackOption(S0,sigma,tau,r,delta,Narbitrage);
  local S,prob,fa;
  local ST,Smin,Smax,Gcall,Gput;
  local lbCall,lbPut;

  {S,prob,fa} = _constructionCheminsBinomiaux(S0,sigma,tau,r,delta,Narbitrage);

  S = S';

  ST = S[Narbitrage+1,.];
  Smin = minc(S);
  Smax = maxc(S);
```

```
  Gcall = ST - Smin;
  Gcall = Gcall .* (Gcall .> 0);

  Gput = Smax - ST;
  Gput = Gput .* (Gput .> 0);

  lbCall = fa*(prob'Gcall);
  lbPut = fa*(prob'Gput);

  retp(lbCall,lbPut);
endp;
```

L'exemple

```
new;

#include option1.src;
#include option2.src;
#include option3.src;

{floatCall,floatPut} = floatingStrikeOption(100,0.20,90/365,0.08,0,3);

output file = option3.out reset;

print ftos(floatCall,"Valeur du call : %lf",10,5);

output off;


Valeur du call :    2.66160
```

1.1.4 Programme n°4

La *vectorisation* est l'une des notions les plus importantes du langage GAUSS. Considérons la valorisation d'options BS et CRR. La formule de Black-Scholes est

$$C = S_0\Phi(d_1) - Ke^{-r\tau}\Phi(d_2)$$

avec

$$\begin{aligned} d_1 &= \frac{\ln\frac{S_0}{K} + r\tau}{\sigma\sqrt{\tau}} + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= d_1 - \sigma\sqrt{\tau} \end{aligned}$$

La procédure suivante `callBS` permet de valoriser une option particulière. Nous pouvons aussi utiliser cette procédure pour valoriser m options. Dans ce cas, les arguments de la procédure sont des vecteurs de dimension m. Par exemple, avec

$$S_0 = \begin{bmatrix} 100 \\ 1065 \end{bmatrix} \quad K = \begin{bmatrix} 98 \\ 1070 \end{bmatrix} \quad \sigma = \begin{bmatrix} 0.25 \\ 0.86 \end{bmatrix} \quad \tau = \begin{bmatrix} \frac{90}{365} \\ \frac{60}{365} \end{bmatrix} \quad r = \begin{bmatrix} 0.10 \\ 0.10 \end{bmatrix}$$

nous valorisons deux options. La première porte sur un sous-jacent dont le prix est égal à 100 francs. C'est une option à 90 jours et de prix d'exercice égal à 98 francs. La seconde option est plus courte (60 jours) et porte sur un actif dont le prix est égal à 1065 francs. L'utilisation des opérateurs $E \times E$ permet cependant d'utiliser d'autres formats de matrice. Dans l'exemple, avec les matrices suivantes

$$S_0 = \begin{bmatrix} 100 \\ 98 \end{bmatrix} \quad K = \begin{bmatrix} 98 & 99 & 100 & 101 \end{bmatrix} \quad \sigma = 0.20 \quad \tau = \frac{90}{365} \quad r = 0.08$$

nous calculons 8 options d'achat

$$\begin{matrix} C_1 & C_2 & C_3 & C_4 \\ C_5 & C_6 & C_7 & C_8 \end{matrix}$$

Pour chacune de ces huit options, nous utilisons les mêmes volatilité, maturité et taux d'intérêt ; ce qui diffère sont, d'une part la valeur du sous-jacent (celle-ci change selon la ligne de la matrice) et, d'autre part la valeur du prix d'exercice (qui diffère selon la colonne de la matrice). Par exemple, C_7 correspond à une option dont le prix du sous-jacent est 98 francs et dont le prix d'exercice est 100 francs. Cette méthode peut aussi être employée pour les modèles binomiaux de type CRR.

La procédure

```
proc (1) = callBS(S0,K,sigma,tau,r);
  local w_,d1,d2,Prime;

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

  Prime = S0.*cdfn(d1) - K.*exp(-r.*tau).*cdfn(d2);

  retp(Prime);
endp;

proc (1) = callCRR(S0,K,sigma,tau,r,N);
  local u,d,fa,pi_,q;
  local j,B,Prime,G;

  u = exp(sigma.*sqrt(tau./N));
  d = 1./u;
  fa = exp(-r.*tau);

  pi_ = (exp(r.*tau./N)-d)./(u-d);
  q = 1 - pi_;

  j = seqa(0,1,N+1);
  B = (N!)./((j!).*((N-j)!));

  Prime = 0;

  j = 0;
  do until j > N;
     G = S0.*(u^j).*(d^(N-j)) - K;
```

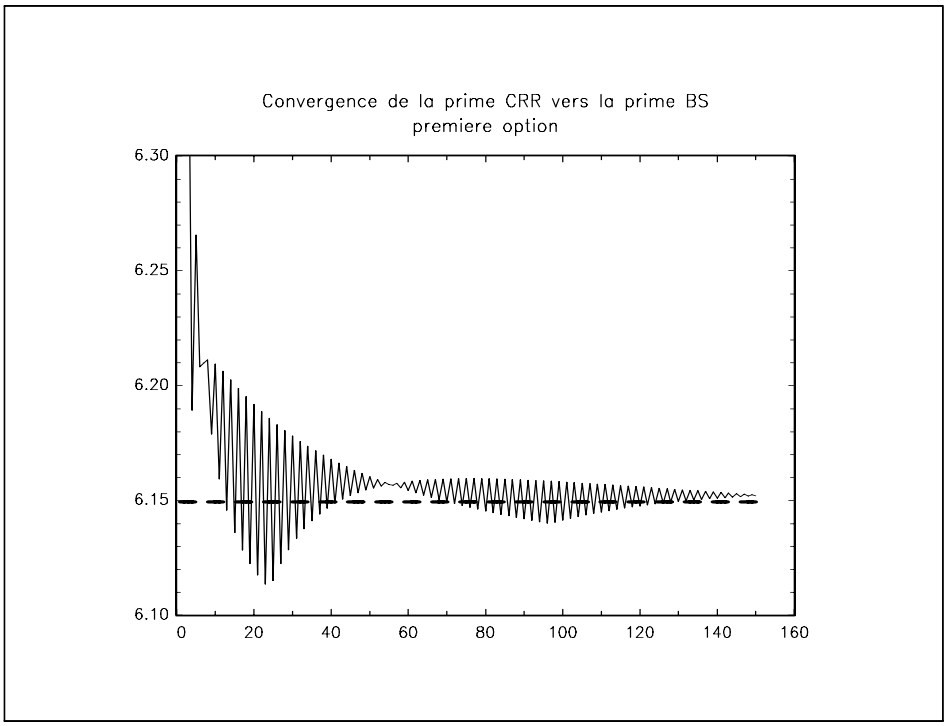
```
    G = G .* ( G .> 0);
    Prime = Prime + B[j+1]*(pi_^j).*(q^(N-j)).*G;
    j = j + 1;
  endo;

  Prime = fa.*Prime;

  retp(Prime);
endp;
```

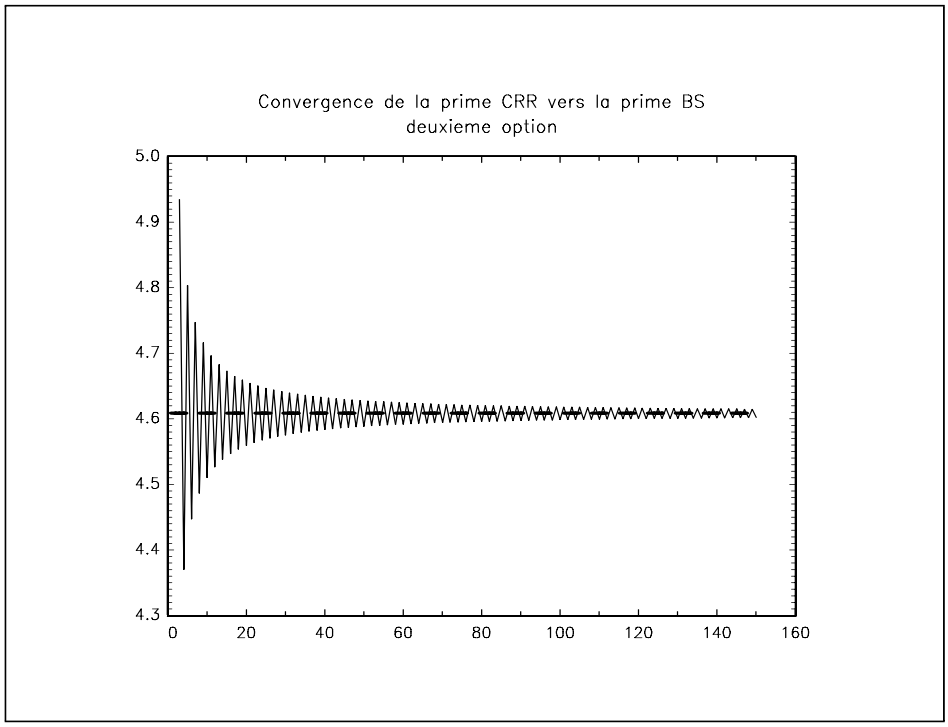
L'exemple

```
new;
library option,pgraph;

S0 = 100|98; K = 98~99~100~101;

PrimeBS = callBS(S0,K,0.20,90/365,0.08);
PrimeCRR = callCRR(S0,K,0.20,90/365,0.08,3);

output file = option4.out reset;

print "Prime Black-Scholes :"; print PrimeBS;
print;
print "Prime Cox-Ross-Rubinstein :"; print PrimeCRR;

output off;

PrimeBS = callBS(100|98,98,0.20|0.25,90/365|60/365,0.08);

N = 150;
PrimeCRR = miss(zeros(2,N),0);

i = 3;
do until i > N;
  PrimeCRR[.,i] = callCRR(100|98,98,0.20|0.25,90/365|60/365,0.08,i);
  i = i + 1;
endo;

titre = "Convergence de la prime CRR vers la prime BS";

graphset;
  _pdate = ""; _pnum = 2; _plwidth = 0|10;
  title(titre$+"\Lpremiere option");
  ytics(6.1,6.3,0.05,5);
  graphprt("-c=1 -cf=option4a.eps");
  xy(seqa(1,1,N),PrimeCRR[1,.]'~PrimeBS[1]*ones(N,1));

  graphprt("-c=1 -cf=option4b.eps");
  title(titre$+"\Ldeuxieme option");
```

```
  ytics(4.3,4.9,0.1,10);
  xy(seqa(1,1,N),PrimeCRR[2,.]'~PrimeBS[2]*ones(N,1));
```

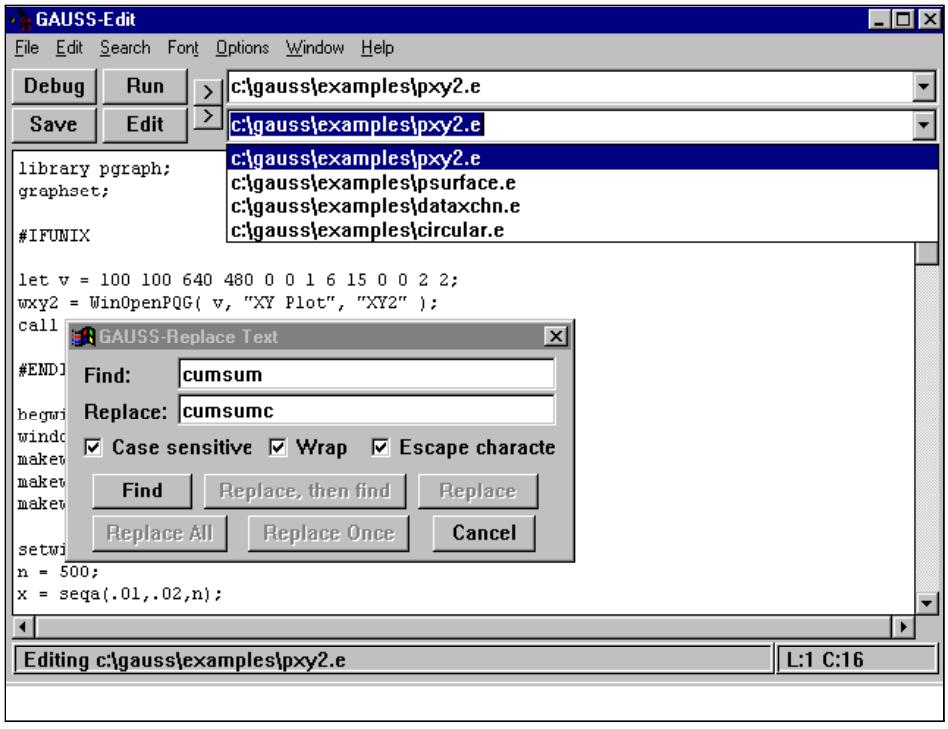
```
Prime Black-Scholes :

       6.1493286         5.5430891         4.9753745         4.4465780
       4.8758670         4.3474683         3.8587757         3.4093785

Prime Cox-Ross-Rubinstein :

       6.3957115         5.8421528         5.2885941         4.7350353
       5.1828222         4.6292634         4.0757047         3.5221460
```





1.2 L'environnement Windows NT/95

La version GAUSS pour Windows bénéficie de l'environnement Windows NT/95. Elle se présente de la façon suivante :



Quelques points intéressants :

- L'historique des programmes édités et exécutés est gardé en mémoire. Nous pouvons ainsi *basculer* d'un programme à un autre très facilement.
- La possibilité d'exécuter des programmes en mode batch ou multitache sans perdre le contrôle du système d'exploitation.

- Un nouvel utilitaire d'aide beaucoup plus complet (version 3.2.33+).
- L'accès à win 32 API.

Remarque : En mode batch, il est nécessaire de spécifier les répertoires si nous voulons que le programme s'exécute correctement. Considérons par exemple le fichier *batch.prg* du répertoire *c:\gauss\conf2*. Pour lancer ce programme, nous exécutons la ligne de commande suivante dans une fenêtre MS-DOS :

start /wait c:\gauss -b c:\gauss\conf2\batch.prg

Cependant, le répertoire courant de GAUSS est *c:\gauss*. C'est pourquoi il est nécessaire d'utiliser la commande `ChangeDir` ou de spécificier les chemins absolus.

```
new;

output file = c:\gauss\conf2\batch.out reset;

err = ChangeDir("c:\\gauss\\conf2");

if err $== "";

  print /flush "Une erreur s'est produite.";

else;

  load y;
  y = y + 1;
  save c:\gauss\conf2\y.fmt = y;
  print /flush "Le programme s'est bien execute.";

endif;

output off;
```

1.3 Optimisation de la version Windows

La version winNT/95 est en cours d'optimisation. Celle-ci a commencé avec la version 3.2.29.

Temps de calcul pour la version 3.2.26

```
300x300 normal distributed random matrix^1000____ :   8.630  sec.
Eigenval. of a normal distr. 200x200 randommatrix :   9.830  sec.
Inverse of a 500x500 uniform distr. random matrix :  31.040  sec.
500000 values sorted ascending___________________ :   8.240  sec.
800x800 Toeplitzmatrix___________________________ :   8.180  sec.
Cholesky decomposition of a 500x500-matrix_______ :   3.620  sec.
600x600 correlation matrix_______________________ : 111.780  sec.
500x500 cross-product matrix_____________________ :  61.020  sec.
FFT over 100000 values___________________________ :   4.840  sec.
Gaussian error function over a 500x500 matrix____ :   3.850  sec.
```

```
Gamma function over a 500x500 matrix_____________ :   1.930  sec.
Linear regression over a 500x500 matrix__________ :  22.250  sec.

Overall Performance______________________________ : 275.210  sec.
```

Temps de calcul pour la version 3.2.32

```
300x300 normal distributed random matrix^1000____ :   8.620  sec.
Eigenval. of a normal distr. 200x200 randommatrix :  23.290  sec.
Inverse of a 500x500 uniform distr. random matrix :  28.780  sec.
500000 values sorted ascending___________________ :   7.470  sec.
800x800 Toeplitzmatrix___________________________ :   9.940  sec.
Cholesky decomposition of a 500x500-matrix_______ :   3.790  sec.
600x600 correlation matrix_______________________ :  41.470  sec.
500x500 cross-product matrix_____________________ :  18.730  sec.
FFT over 100000 values___________________________ :   4.840  sec.
Gaussian error function over a 500x500 matrix____ :   3.850  sec.
Gamma function over a 500x500 matrix_____________ :   1.870  sec.
Linear regression over a 500x500 matrix__________ :  22.250  sec.

Overall Performance______________________________ : 174.900  sec.
```

1.4 Les nouvelles commandes

1.4.1 Les commandes de gestion de matrices creuses

GAUSS comprend désormais des commandes pour les matrices creuses : création d'une matrice creuse à partir d'une matrice dense, concaténation horizontale/verticale, multiplication vectorielle d'une matrice creuse par une matrice dense, solution d'un système linéaire creux. Ces commandes sont intéressantes lorsque nous manipulons des systèmes d'ordre élevé. Par exemple, l'utilisation de l'algorithme tridiagonal n'est pas toujours possible pour la résolution de certaines EDP financières. C'est notamment le cas pour la valorisation de certaines options du fait de conditions supplémentaires aux bornes (non Feynman-Kac). L'utilisation d'un schéma explicite avec des différences finies conduit alors à résoudre des système de la forme $Ax = b$ avec A une matrice presque tridiagonale. Voici quelques exemples de matrices creuses :

$$A = \begin{bmatrix} \star & \star & \star & \star & \star & \star & \star & \star & \star \\ \star & \star & \star & & & & & & \\ & \star & \star & \star & & & & & \\ & & \star & \star & \star & & & & \\ & & & \star & \star & \star & & & \\ & & & & \star & \star & \star & & \\ & & & & & \star & \star & \star & \\ & & & & & & \star & \star & \star \\ & & & & & & & \star & \star \end{bmatrix}$$

$$A = \begin{bmatrix} \star & \star & \star & & & & & & \\ \star & \star & \star & & & & & & \\ & \star & \star & \star & & & & & \\ & & \star & \star & \star & & & & \\ & & & \star & \star & \star & & & \\ \star & \star & \star & \star & \star & \star & \star & \star & \star \\ & & & & & \star & \star & \star & \\ & & & & & & \star & \star & \star \\ & & & & & & & \star & \star \end{bmatrix}$$

Nous pouvons aussi avoir

$$A = \begin{bmatrix} \star & \star & & & & & \star & \star & \star \\ \star & \star & \star & & & & & \star & \star \\ & \star & \star & \star & & & & & \star \\ & & \star & \star & \star & & & & \\ & & & \star & \star & \star & & & \\ & & & & \star & \star & \star & & \\ \star & & & & & \star & \star & \star & \\ \star & \star & & & & & \star & \star & \star \\ \star & \star & \star & & & & & \star & \star \end{bmatrix}$$

Dans ce cas, l'utilisation d'algorithmes pour les matrices denses est prohibitive en terme de temps de calcul et de gestion mémoire. Dans l'exemple suivant, nous reprenons la forme précédente de la matrice A. Nous fixons sa dimension à 10000. La création de cette matrice nécessite alors 800 Mo ($10000 \times 10000 \times 8$ octets). Or, 400 Ko suffisent seulement pour la stocker sous forme d'une matrice creuse, ce qui nous permet de manipuler cette matrice. Avec 32 Mo de mémoire vive et 1000 Mo de mémoire virtuelle, un programme équivalent qui n'utiliserait pas les commandes creuses demanderait plusieurs jours de calcul.

```
new;

N = 10000;    /* dimension de la matrice creuse */
s = seqa(1,1,N);

d0 = s[2:N]~s[1:N-1];
d1 = s~s;
d2 = s[1:N-1]~s[2:N];

d3 = 1 ~ N-2  |  1 ~ N-1  |  1 ~ N  |
                 2 ~ N-1  |  2 ~ N  |
                             3 ~ N  ;

d4 = N-2 ~ 1  |
     N-1 ~ 1  |  N-1 ~ 2  |
     N   ~ 1  |  N   ~ 2  |  N  ~3  ;

d = d0|d1|d2|d3|d4;

A = sparseFP(d~rndu(rows(d),1)*10,N,N);   /* Matrice creuse A */

x = rndn(N,1);
```

```
b = sparseTD(A,x);   /* b = A*x */

output file = sparse.out reset;

show A;

output off;


A
  400136 bytes at [01330eb8]    50017,1      MATRIX

65536 bytes program space, 1% used
31391728 bytes workspace, 29779264 bytes free
62 global symbols, 1500 maximum, 1 shown
```

1.4.2 Les procédures d'optimisation

Dans la version de base, GAUSS dispose désormais de procédures d'optimisation :

1. `eqSolve` : Résolution d'un système non linéaire
2. `Qnewton` : Algorithme de quasi-Newton
3. `Qprog` : Programmation quadratique
4. `sqpSolve` : Programmation non linéaire

La procédure `Qprog` s'avère très rapide dans la pratique. Quelques utilisations possibles : méthode DEA pour l'efficience allocative — *http://www.montesquieu.u-bordeaux.fr/u/roncalli/dea.html*, détermination de portefeuilles efficaces — *http://www.montesquieu.u-bordeaux.fr/u/roncalli/gauss&finan* etc.

La procédure `qpMCO` détermine la solution des moindres carrés ordinaires lorsque nous imposons des contraintes :

$$\min_{\beta} (Y - X\beta)^{\top} (Y - X\beta)$$

$$\text{sc} \quad \begin{cases} A\beta = B \\ C\beta \geq D \\ \underline{\beta} \leq \beta \leq \overline{\beta} \end{cases}$$

```
new;

rndseed 123;

proc (2) = qpMCO(y,x,A,B,C,D,bnds);
  local Q,r;
  local beta,u1,u2,u3,u4,retcode;

  Q = 2*(x'x);
  r = 2*x'y;
```

```
  {beta,u1,u2,u3,u4,retcode} = QProg(y/x,Q,r,A,B,C,D,bnds);

  retp(beta,retcode);
endp;


Nobs = 1000;
k = 25;
sigma = 1;

coeff = rndu(k,1);
x = 50*rndu(Nobs,k);
y = x*coeff + rndn(Nobs,1)*sigma;

/* Estimation non contrainte */

{beta1,retcode} = qpMCO(y,x,0,0,0,0,0);

/* beta[1] = 2 */

A = 1~zeros(1,k-1); B = 2;
{beta2,retcode} = qpMCO(y,x,A,B,0,0,0);

/*
   beta[1] + ... + beta[k] = 0
   beta[1] + 2 *beta[2] > 5
   beta[3] - beta[4]  > 2
   -0.5 < beta[i] < 4
*/

A = ones(1,k); B = 0;
C = 1~2~zeros(1,k-2)      |
    0~0~1~-1~zeros(1,k-4) ;
D = 5|2;
bnds = { -0.5 4};
{beta3,retcode} = qpMCO(y,x,A,B,C,D,bnds);

output file = mco.out reset;

print beta1[1:10]~beta2[1:10]~beta3[1:10];

output off;
```

```
      0.75285545        2.0000000        1.4368509
      0.32006351       0.28792053        1.7815745
      0.17993927       0.11660904        1.5000000
      0.90738662       0.81564121      -0.50000000
      0.35868723       0.32530055      0.038244367
```

```
    0.22178544       0.15325542      -0.50000000
    0.78582469       0.74051385      -0.50000000
    0.39147069       0.34259510      -0.50000000
    0.12533869       0.11298412      -0.50000000
    0.18427291       0.11056641       0.17615002
```

1.4.3 La gestion des fichiers

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. Voir les exemples de "GAUSS et la Finance" (*http://www.montesquieu.u-bordeaux.fr/u/roncalli/gauss&finance.html*). La procédure de conversion `csv2spt` (de la bibliothèque **SPT)** de fichiers DATASTREAM au format CSV utilise ces commandes. Voici le code de cette procédure :

```
proc (0) = csv2spt(fichierCSV,l);
  local finput,foutput,text,data;
  local j,m,a,a2000,p,d,str,fw,i;

  finput = fichierCSV $+ ".csv";
  foutput = fichierCSV $+ ".prx";

  let string text[5,2] =   ";"      " "
                           "#n/a"   "."
                           ","      "."
                           "\34"    " "
                           "/"      " ";

  call _csvtospt3(finput,foutput,text,l);
  load data[] = ^foutput;
  data = reshape(data,rows(data)/4,4);
  j = data[.,1];
  m = data[.,2];
  a = data[.,3];
  p = data[.,4];

  a2000 = a .<= 10;
  a = (a2000.*2000 + (1-a2000).*1900) + a;
  d = a*10000 + m*100 + j;

  fw = fopen(foutput,"w");

  i = 1;
  do until i > rows(d);
    str = ftos(d[i],"%lf",10,0) $+ ftos(p[i],"   %lf",20,10);
    call fputst(fw,str);
    i = i + 1;
  endo;

  fw = close(fw);
```

```
  retp;
endp;


proc (1) = _csv2spt1(str,substr1,substr2);
  local newstr,l,l1,pos;

  newstr = str;

  l =  strlen(str);
  l1 =  strlen(substr1);

  pos = 1;

  do while pos == 1;
    {newstr,pos} = _csv2spt2(newstr,substr1,substr2,l,l1);
  endo;

  retp(newstr);
endp;


proc (2) = _csv2spt2(str,substr1,substr2,l,l1);
  local pos,str1,str2,newstr;

  pos = strindx(str,substr1,1);

  if pos == 0;
    retp(str,0);
  endif;

  str1 = strsect(str,1,pos-1);
  str2 = strsect(str,pos+l1,l);

  newstr = str1 $+ substr2 $+ str2;

  retp(newstr,1);
endp;


proc (0) = _csvtospt3(finput,foutput,text,l);
  local N,fr,fw,str,i,j;

  N = rows(text);

  fr = fopen(finput,"r");
  fw = fopen(foutput,"w");

  j = 0;
```

```
  do until eof(fr);
    str = fgetsa(fr,1);

    j = j + 1;
    if j <= l;
      continue;
    endif;

    i = 1;
    do until i > N;
      str = _csv2spt1(str,text[i,1],text[i,2]);
      i = i + 1;
    endo;

    call fputs(fw,str);
  endo;

  fr = close(fr);
  fw = close(fw);

  retp;
endp;
```

L'utilisation de ces nouvelles commandes permet par exemple de créer très facilement une procédure de conversion de bases de données HTML au format GAUSS.

La procédure `filesa` fonctionne comme la procédure `files`. Pour obtenir des informations supplémentaires sur les fichiers, nous utilisons la procédure `fileinfo`.

Exemple d'utilisation de la commande `filesa`

```
new;

output file = file1.out reset;

f = files("c:\\gauss\\*.exe",0);
print $f;

f = filesa("*.out");
print f;

output off;


          gauss             .exe
           ATOG             .EXE
        unstall             .exe
        CHKARGS             .EXE

    OPTION1.OUT
```

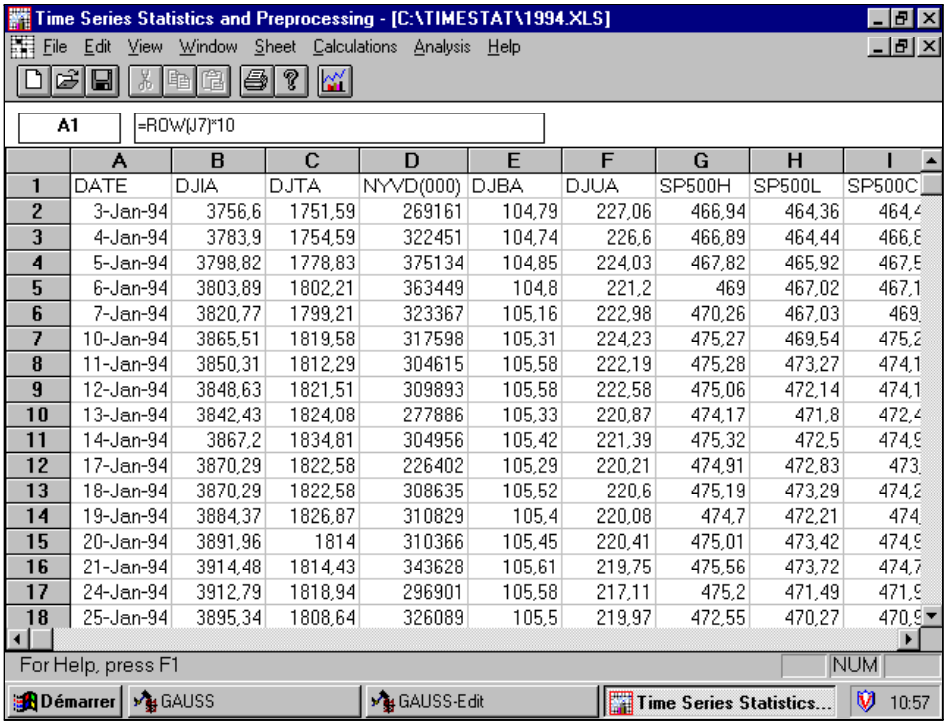
```
     OPTION2.OUT
     OPTION3.OUT
     OPTION4.OUT
       BATCH.OUT
      SPARSE.OUT
         MC0.OUT
       FILE1.OUT
          BM.OUT
       FILE2.OUT
```

Exemple d'utilisation de la commande `fileinfo`

```
new;

{fnames,finfo} = fileinfo("*.out");

output file = file2.out reset;

print "Nom des fichiers" fnames; print;
print "Dimension (en Ko)"; call printfmt(finfo[.,8],1);

output off;


Nom des fichiers
     OPTION1.OUT
     OPTION2.OUT
     OPTION3.OUT
     OPTION4.OUT
       BATCH.OUT
      SPARSE.OUT
         MC0.OUT
       FILE1.OUT
          BM.OUT
       FILE2.OUT

Dimension (en Ko)
            1120
             802
              29
             338
              34
             216
             532
             338
            1319
             419
```

Quatre nouvelles commandes sont disponibles dans la version 3.3.32 pour exporter/importer des bases de données au format Lotus, Excel, Quatro, Symphony, dBase, Paradox, etc. Dans l'exemple suivant, nous importons des données au format Excel 4.0. Il est intéressant de noter le codage

adopté par Excel pour les dates. La procédure `DateExcel` permet de retrouver la date à partir du code Excel.



	A	B	C	D	E	F	G	H	I
1	DATE	DJIA	DJTA	NYVD(000)	DJBA	DJUA	SP500H	SP500L	SP500C
2	3-Jan-94	3756,6	1751,59	269161	104,79	227,06	466,94	464,36	464,4
3	4-Jan-94	3783,9	1754,59	322451	104,74	226,6	466,89	464,44	466,8
4	5-Jan-94	3798,82	1778,83	375134	104,85	224,03	467,82	465,92	467,5
5	6-Jan-94	3803,89	1802,21	363449	104,8	221,2	469	467,02	467,1
6	7-Jan-94	3820,77	1799,21	323367	105,16	222,98	470,26	467,03	469
7	10-Jan-94	3865,51	1819,58	317598	105,31	224,23	475,27	469,54	475,2
8	11-Jan-94	3850,31	1812,29	304615	105,58	222,19	475,28	473,27	474,1
9	12-Jan-94	3848,63	1821,51	309893	105,58	222,58	475,06	472,14	474,1
10	13-Jan-94	3842,43	1824,08	277886	105,33	220,87	474,17	471,8	472,4
11	14-Jan-94	3867,2	1834,81	304956	105,42	221,39	475,32	472,5	474,9
12	17-Jan-94	3870,29	1822,58	226402	105,29	220,21	474,91	472,83	473
13	18-Jan-94	3870,29	1822,58	308635	105,52	220,6	475,19	473,29	474,2
14	19-Jan-94	3884,37	1826,87	310829	105,4	220,08	474,7	472,21	474
15	20-Jan-94	3891,96	1814	310366	105,45	220,41	475,01	473,42	474,9
16	21-Jan-94	3914,48	1814,43	343628	105,61	219,75	475,56	473,72	474,7
17	24-Jan-94	3912,79	1818,94	296901	105,58	217,11	475,2	471,49	471,9
18	25-Jan-94	3895,34	1808,64	326089	105,5	219,97	472,55	470,27	470,9

Exemple d'utilisation de la commande `import`

```
new;

#include datexcel.src;

{x,nm} = import("c:\\timestat\\1994.xls",0,0);

nm = nm';

output file = import.out reset;

print $nm;
print x[1:10,.];

let date1 = 1900 01 01;
let date2 = 1994 01 03;

/* Codage Excel pour les dates : 1900/01/01 = 1, 1900/01/02 = 2, etc */

nj = 1 + etdays(date1,date2);
nj = nj + 1;                    /* correction pour l'annee 1900 */
print; print nj; print;

i = 1;
do until i > 100;
  d = DateExcel(x[i,1]);
```

```
  print i~d';
  i = i + 10;
endo;

output off;
```

DATE	DJIA	DJTA	NYVD(000	DJBA
34337.000	3756.6000	1751.5900	269161.00	104.79000
34338.000	3783.9000	1754.5900	322451.00	104.74000
34339.000	3798.8200	1778.8300	375134.00	104.85000
34340.000	3803.8900	1802.2100	363449.00	104.80000
34341.000	3820.7700	1799.2100	323367.00	105.16000
34344.000	3865.5100	1819.5800	317598.00	105.31000
34345.000	3850.3100	1812.2900	304615.00	105.58000
34346.000	3848.6300	1821.5100	309893.00	105.58000
34347.000	3842.4300	1824.0800	277886.00	105.33000
34348.000	3867.2000	1834.8100	304956.00	105.42000

34337.000				
1.0000000	1994.0000	1.0000000	3.0000000	
11.000000	1994.0000	1.0000000	17.000000	
21.000000	1994.0000	1.0000000	31.000000	
31.000000	1994.0000	2.0000000	14.000000	
41.000000	1994.0000	3.0000000	1.0000000	
51.000000	1994.0000	3.0000000	15.000000	
61.000000	1994.0000	3.0000000	29.000000	
71.000000	1994.0000	4.0000000	13.000000	
81.000000	1994.0000	4.0000000	28.000000	
91.000000	1994.0000	5.0000000	12.000000	

Procédure DateExcel

```
/*
** Procedure pour retrouver la date
** a partir du codage Excel
*/

proc DateExcel(de);
  local cn1,cn2,mo,y,m,d,dg;

  de = de - 1;  /* correction pour l'annee 1900 */

  /* Determination de l'annee */

  y = 1900 + trunc(de/365);

  cn1 = etdays(1900|01|01,y|01|01);
  cn2 = etdays(1900|01|01,y|12|31);
```

```
  if cn1 > de;
    y = y - 1;
  endif;

  if cn2 < de;
    y = y + 1;
  endif;

  if _isleap(y);
    let mo = 31 29 31 30 31 30 31 31 30 31 30 31;
  else;
    let mo = 31 28 31 30 31 30 31 31 30 31 30 31;
  endif;
  mo = cumsumc(mo);

  /* Determination du mois */

  de = de - etdays(1900|01|01,y|01|01);
  m = 1 + sumc(de .> mo);

  /* Determination du jour */

  mo = 0|mo;
  d = de - mo[m];

  dg = y|m|d;

  retp(dg);
endp;
```

1.5 L'accès à Win 32 API

Avec l'apparition de la version Windows, Aptech a modifié l'interface avec les langages externes (FLI — Foreign Language Interface). Celle-ci se fait désormais par le biais de DLL compatibles GAUSS. Celles-ci sont créées par exemple à partir de langages C/C++ (VC++, Symantec C++, Watcom C/C++, etc[1]), Fortran (Fortran Power Station, Digital Visual Fortran, etc) ou Visual Basic. Nous avons donc maintenant accès à WIN 32 API sans aucune difficulté. A partir de GAUSS, nous pouvons par exemple créer des fenêtres, les fermer, les ouvrir, lire un CD-ROM, exécuter d'autres programmes.

$\Longrightarrow$Lecture de base de données CD-ROM.

1.5.1 Exemple n°1

Cet exemple est issu de Mercury. La procédure `play_sound` est construite à partir de la routine `PlaySound` de *winmm.lib* de **MFC**.

```
new;
dlibrary vblink;
```

[1]Pour l'instant, il semble qu'il y ait des difficultés pour utiliser la version NT/95 de g++.

```
cls;

print "GAUSS peut jouer the Microsoft Sound.";
print;

FichierSon = "C:\\WINDOWS\\MEDIA\\The Microsoft Sound.wav";
retour = 0;

i = 1;
do until i > 3;
  dllcall -r play_sound(retour,FichierSon);
  i = i + 1;
endo;
```

1.5.2 Exemple n°2

C'est un autre exemple de Mercury. La procédure `show_window` est basée sur les routines `FindWindow` et `ShowWindow` (que nous trouvons aussi dans **VB**).

fichier win32api.src

```
dlibrary vblink;

proc (0) = ActiverFenetreGauss;
  local nomFenetre,retour,action;

  nomFenetre  = "GAUSS";
  retour = 0;
  action  = 3;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;

proc (0) = DesActiverFenetreGauss;
  local nomFenetre,retour,action;

  nomFenetre  = "GAUSS";
  retour = 0;
  action  = 7;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;

proc (0) = ActiverFenetreGaussEdit;
  local nomFenetre,retour,action;

  nomFenetre  = "GAUSS-Edit";
```

```
  retour = 0;
  action  = 3;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;

proc (0) = DesActiverFenetreGaussEdit;
  local nomFenetre,retour,action;

  nomFenetre  = "GAUSS-Edit";
  retour = 0;
  action  = 7;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;

proc (0) = ActiverFenetreTimeStat;
  local nomFenetre,retour,action;

  nomFenetre  = "Time Series Statistics and Preprocessing";
  retour = 0;
  action  = 3;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;

proc (0) = DesActiverFenetreTimeStat;
  local nomFenetre,retour,action;

  nomFenetre  = "Time Series Statistics and Preprocessing";
  retour = 0;
  action  = 7;

  dllcall show_window(retour,nomfenetre,action);
  retp;
endp;
```

Le programme

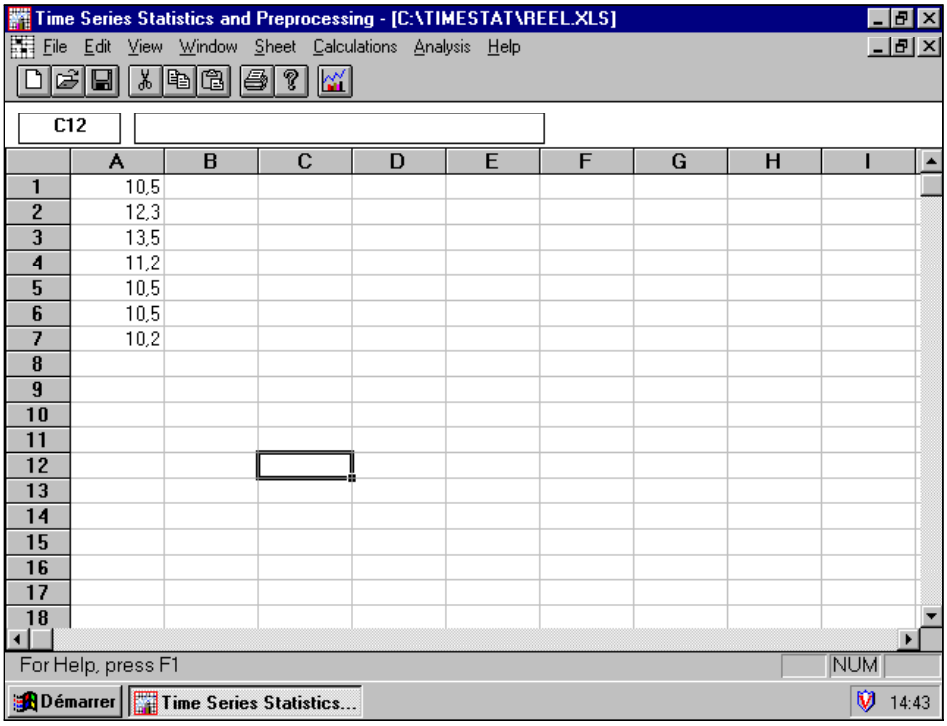
```
new;
#include win32api.src;

sec = 1;

i = 1;
do until i > 3;
```

```
  call ActiverFenetreGauss; call pause(sec);
  call ActiverFenetreGaussEdit; call pause(sec);
  call ActiverFenetreTimeStat; call pause(sec);

  call DesActiverFenetreTimeStat; call pause(sec);
  call DesActiverFenetreGaussEdit; call pause(sec);
  call DesActiverFenetreGauss; call pause(sec);

  i = i + 1;
endo;
```

1.6 Un exemple de traitement statistique en temps réel

Les nouvelles fonctionnalités et commandes de la version NT/95 permettent d'élaborer des traitements statistiques en temps réel. Nous supposons que les données arrivent de façon continue sur le disque dur. Pour simuler ce processus, nous utilisons le logiciel TimeStat, qui permet de sauvegarder les données au format Excel. Ce logiciel peut être téléchargé à l'adresse suivante :

http://sunsite.unc.edu/pub/archives/misc.invest/programs/tstat120.zip



Pour savoir si des données nouvelles sont arrivées, nous procédons très simplement. Nous testons si le fichier a été modifié, par exemple en regardant si la date d'enregistrement est la même. Si la réponse est positive, nous chargeons les nouvelles données et calculons les statistiques correspondantes.

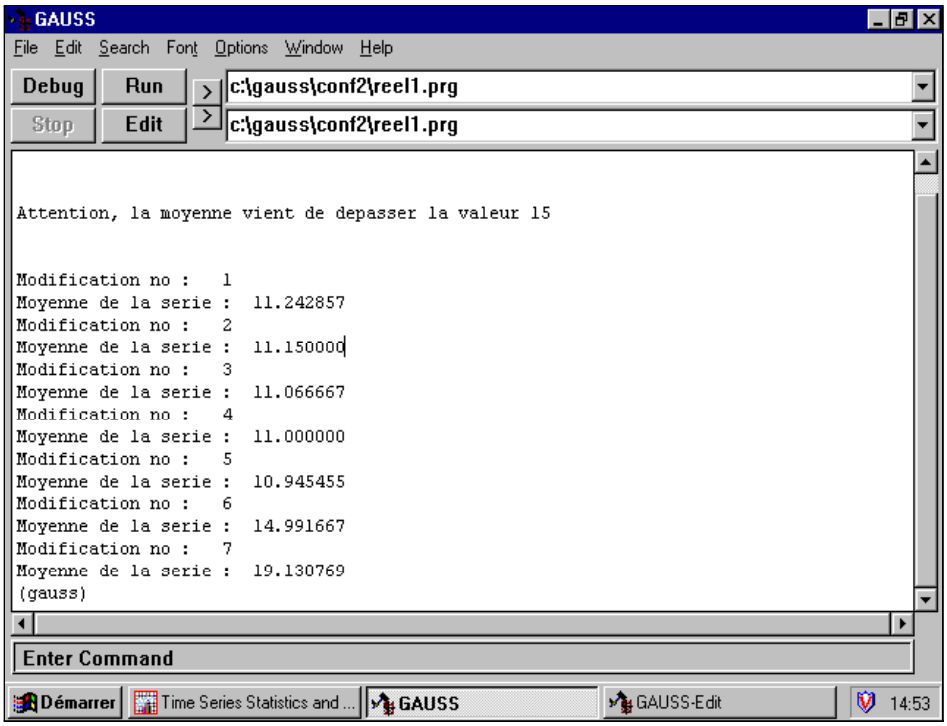
```
new;

cls;
_dxprint = 0;
```

```
#include win32api.src;
#include files2.src;

call DesActiverFenetreGaussEdit;
call DesActiverFenetreGauss;
call ActiverFenetreTimeStat;

fichier = "c:\\timestat\\reel.xls";

if not FileExist(fichier);
  call ActiverFenetreGauss;
  cls;
  strng = "Le fichier " $+ fichier $+ " n'existe pas.";
  print /flush strng;
  end;
endif;

{cn,dchngt} = FileChange(fichier,0);
m = 0;

i = 1;

do while 1;

  {cn,dchngt} = FileChange(fichier,dchngt);

  if cn == 1;
    {x,nom} = import(fichier,0,0);
    m = meanc(x);
    locate 5+2*i,1;
    print /flush ftos(i,"Modification no : %lf",3,0);
    print /flush ftos(m,"Moyenne de la serie : %lf",10,6);
    i = i + 1;
  endif;

  locate 2,2; print " ";

  if m > 15;
    call DesActiverFenetreTimeStat;
    call ActiverFenetreGauss;
    print "Attention, la moyenne vient de depasser la valeur 15";
    end;
  endif;

endo;
```

Les procédures

```
proc (1) = FileExist(fname);
```

```
  local fnames,finfo;

  {fnames,finfo} = fileinfo(fname);

  if finfo == 0;
    retp(0);
  else;
    retp(1);
  endif;

endp;


proc (2) = FileChange(fname,d);
  local fnames,finfo;

  {fnames,finfo} = fileinfo(fname);

  if d == 0;
    retp(1,finfo[10]);
  endif;

  if d == finfo[10];
    retp(0,finfo[10]);
  else;
    retp(1,finfo[10]);
  endif;

endp;
```



2 Mercury : Outils de communication

Mercury est un ensemble d'outils de communication pour **GAUSS**. Il est écrit par John Breslaw, l'auteur de **GaussX**. La version actuelle est 1.0.2. La bibliothèque **vblink** comprend un ensemble de procédures qui permettent de communiquer avec l'application externe. De même, les procédures **VB** permettent de communiquer avec GAUSS. Nous avons alors le schéma suivant :

$$\text{GAUSS} \quad \overset{\text{vblink}}{\underset{\text{VB}}{\rightleftarrows}} \quad \text{Application Externe}$$

Par exemple, `call senddata(x);` (commande exécutée dans GAUSS) retourne la matrice `x` dans l'application externe. Cette matrice peut ensuite être lue dans l'application externe avec la commande `GET_DATA`, si celle-ci supporte les commandes **VB**.

- Remarque : le code source est fourni avec la disquette pour les développeurs.

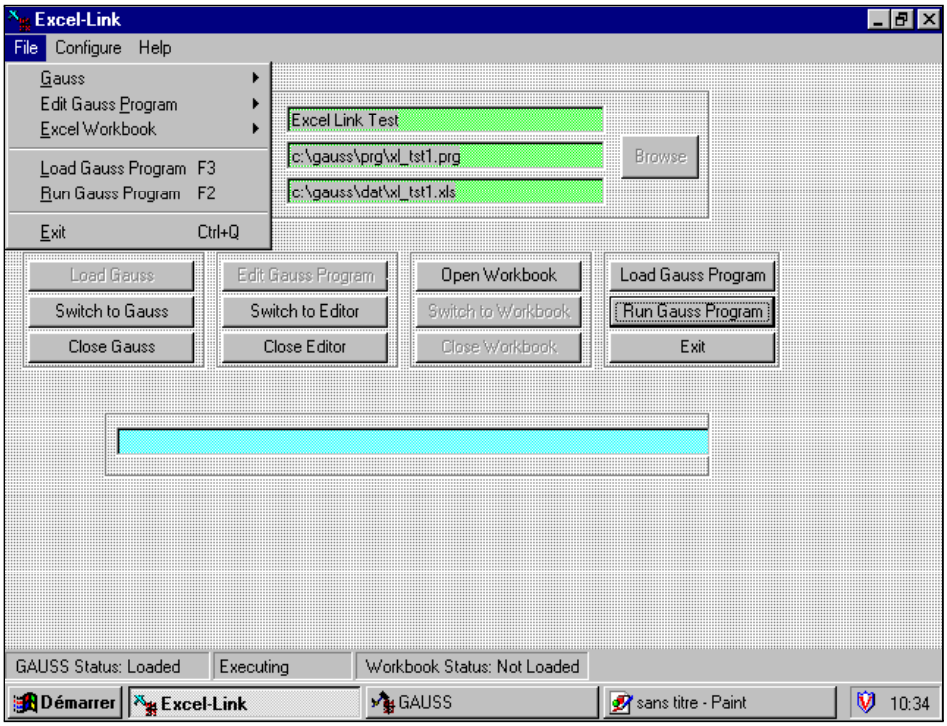
2.1 Un exemple d'utilisation du presse-papiers

Nous pouvons récupérer le contenu du presse-papiers avec la commande `paste`. La commande `copy` permet de copier une matrice ou une chaîne de caractères dans le presse-papiers. Ces deux commandes sont construites très simplement à partir de fichiers DLL. Par exemple, le code source de `copy` est

```
/*=============================================================================
**
**   COPY
**      call copy(x) copies x to the clipboard.
**
=============================================================================*/
 proc copy(x);
   local tx,rx, cx,  ret, s;
   tx = type(x);
   ret = 1;
   if tx == 13;                @ string@
     s = x;
     dllcall copy_string_to_clipboard(ret,s);
   elseif tx == 6;             @ matrix @
     if (rows(x) == 0 and cols(x) == 0);
       dllcall clear_clipboard(ret);
       retp("");
     endif;
     rx = rows(x); cx = cols(x);
     dllcall copy_matrix_to_clipboard(ret,x,rx,cx);
   else;
     call errmsg(ret,"Invalid data type for 'Copy'");
   endif;

   call errmsg(ret,"Error copying to clipboard");
   retp("");
 endp;
```

2.2 Liaison avec Excel

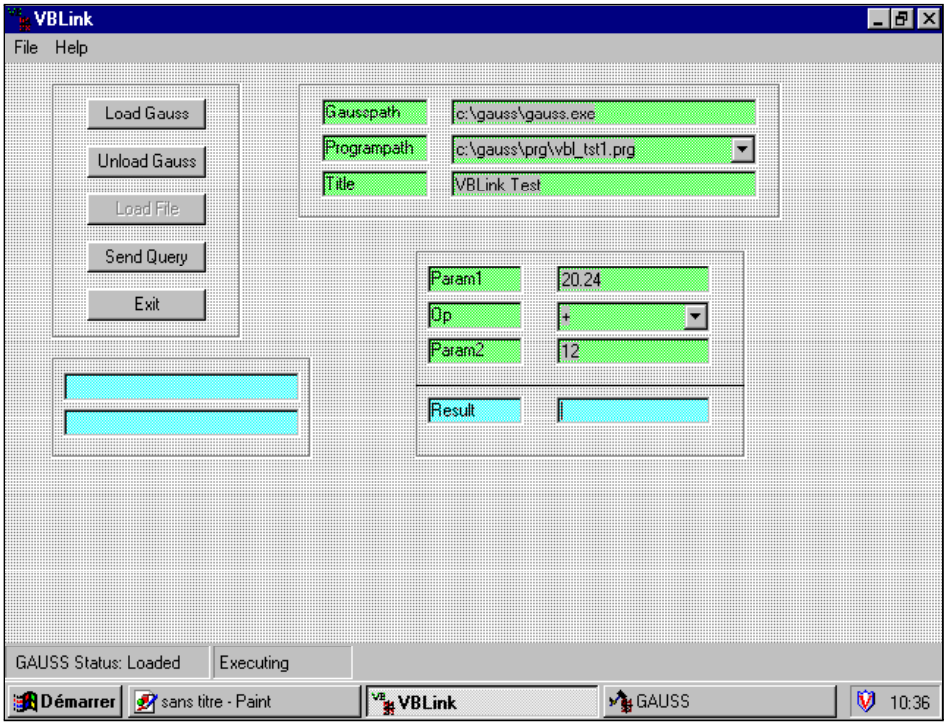
Excel_Link est une interface écrite entièrement en **Visual Basic**. Le code source Visual Basic est fourni avec Mercury. John Breslaw a écrit cette application pour montrer les possibilités DDE (dynamic data exchange) de GAUSS. Une nouvelle fois, la liaison GAUSS — Excel se fait par l'intermédiaire de fichiers DLL.



2.3 Création d'applications GUI

L'exemple suivant illustre l'utilisation de GAUSS dans une application externe. Celle-ci a été construite à partir de **VB**. Cette application externe est une interface entre l'utilisateur et GAUSS. Elle récolte les informations que l'utilisateur lui fournit dans un environnement GUI, les envoie dans GAUSS qui les traite et récupère les résultats. Dans ce cas, l'utilisateur n'a pas besoin de connaître le langage GAUSS pour utiliser cette application externe.

- Illustration : un pricer d'option



2.4 Création de DLL compatibles avec GAUSS

voir les exemples précédents.

3 FANPAC

FANPAC est un module d'estimation et de gestion du risque. Les modèles implémentés sont les suivants :

1. ARCH(q) — Normal distribution ARCH
2. TARCH(q) — t-distribution ARCH
3. GARCH(p,q) — Normal distribution GARCH
4. TGARCH(p,q) — t-distribution GARCH
5. GARCHM(p,q) — GARCH-in-mean
6. EGARCH(p,q) — exponential GARCH
7. IGARCH(p,q) — Normal distribution integrated GARCH
8. ITGARCH(p,q) — t-distribution integrated GARCH
9. FIGARCH(p,q) — Normal fractionally integrated GARCH
10. FITGARCH(p,q) — t-distribution fractionally integrated GARCH
11. BGARCH(p,q) — Multivariate Normal BEKK GARCH
12. TBGARCH(p,q) — Multivariate t-dist. BEKK GARCH
13. MGARCH(p,q) — Mult. Normal "diagonal vec" GARCH

14. TMGARCH(p,q) — Mult. t-dist. "diagonal vec" GARCH
15. ARIMA(p,d,q) — Normal ARIMA
16. TARIMA(p,d,q) — t-distribution ARIMA
17. OLS — Normal ordinary least squares
18. TOLS — t-distribution ordinary least squares

Une des originalités de FANPAC est l'utilisation des contraintes souples de Nelson et Cao pour imposer la stationnarité du modèle et la positivité des variances conditionnelles. Considérons par exemple le modèle GARCH(1,1) suivant :

$$\begin{cases} y_t = x_t\beta + \varepsilon_t \\ \varepsilon_t \sim N\left(0, h_t^2\right) \\ h_t^2 = \omega + \alpha_1 h_{t-1}^2 + \alpha_2 \varepsilon_{t-1}^2 \end{cases}$$

Pour imposer les contraintes précédentes, la plupart des logiciels utilise en outre la reparamétrisation $\omega = \dot{\omega}^2$, $\alpha_1 = \dot{\alpha}_1^2$ et $\alpha_2 = \dot{\alpha}_2^2$. Ce n'est pas le cas de FANPAC qui utilise la programmation non linéaire (SQP).

3.1 La syntaxe FANPAC

Il existe deux modes syntaxiques dans FANPAC. Une autre originalité de FANPAC est l'utilisation de mots-clés, la syntaxe est alors plus proche des logiciels statistiques de type SAS ou TSP que des langages de programmation.

3.1.1 La syntaxe programmation

Considérons un modèle GARCH(1,2). Nous obtenons

$$\begin{cases} y_t = 0.993 + \varepsilon_t \\ \varepsilon_t \sim N\left(0, h_t^2\right) \\ h_t^2 = 0.931 + 0.841 h_{t-1}^2 + 0.010 \varepsilon_{t-1}^2 + 0.116 \varepsilon_{t-2}^2 \end{cases}$$

```
new;
library fanpac;
fanset;

/* Modele Garch(1,2) */

_fan_p = 1;
_fan_q = 2;

/* Chargement de la serie */

load y[320,7] = wilshire.asc;
_fan_series = y[.,4];
Nobs = rows(_fan_series);

/* Contrainte de stationnarite du GARCH */
```

```
_nlp_IneqProc = &garch_ineq;

/* bornes sur les coefficients GARCH */

_nlp_Bounds = { .001   1e250,  /* omega > 0    */
                 0     1,      /* garch_1 >= 0 */
                -1e250 1e250,
                -1e250 1e250,
                -1e250 1e250 };

/* utilisation du gradient analytique */

_nlp_GradProc = &garch_n_grd;

_nlp_IterInfo = 1;
sv = {.2, .3, .2, .1, 1};
{coefs,f,g,retcode} = nlp(&garch_n,sv);

output file = fanpac1.out reset;

print coefs;

output off;


      0.93066861
      0.84099192
     0.010302515
      0.11565207
      0.99287855
```

3.1.2 La syntaxe mots-clés

Le programme suivant correspond à l'exemple précédent. L'emploi de mots-clés (`keyword`) rend l'utilisation de FANPAC très facile.

```
new;
library fanpac;

session Exemple1 'wilshire';

setVarNames date cwprice cwdiv cwret ewprice ewdiv ewret;
setDataSet wilshire.asc;
setSeries cwret;

setInferenceType InvWald;
estimate modele1 garch(1,2);

output file = fanpac2.out reset;

showResults;
```

```
output off;


================================================================================
                                Session: Exemple1
--------------------------------------------------------------------------------
                                   wilshire
--------------------------------------------------------------------------------
 FANPAC Version 1.0.0          Data Set:  wilshire         2/16/1997 10:38:55
================================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                  Run: modele1
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

return code =    0
normal convergence


Model:  GARCH(1,2)

Number of Observations     : 320
Observations in likelihood : 318
Degrees of Freedom         : 313

AIC         1852.20
BIC         1871.01
LRS         1842.20

          roots
     ---------------

      -8.3913293
       1.0304391
       1.1890800

       Abs(roots)
     ---------------

       8.3913293
       1.0304391
       1.1890800


  unconditional variance
----------------------------
       28.154742
```

```
Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from inversion of Wald statistic

Series: cwret

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
-------------------------------------------------------------
omega          0.931      0.692      0.009      2.292
Garch1         0.841      0.048      0.747      0.862
Arch1          0.010      0.030      0.000      0.043
Arch2          0.116      0.063      0.012     -0.009
Const          0.993      0.230      0.541      1.445


Correlation Matrix of Parameters

omega         1.000   -0.587    0.131   -0.214    0.079
Garch1       -0.587    1.000    0.005   -0.534    0.142
Arch1         0.131    0.005    1.000   -0.586    0.180
Arch2        -0.214   -0.534   -0.586    1.000   -0.274
Const         0.079    0.142    0.180   -0.274    1.000
```

3.2 Quelques exemples

3.2.1 Exemple n°1

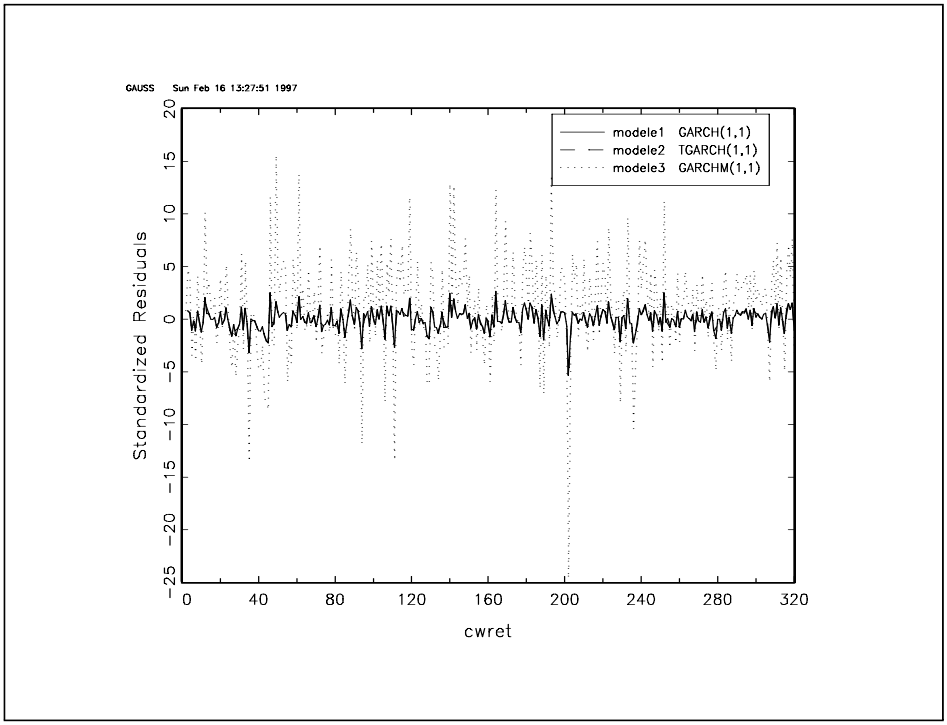
```
new;
library fanpac;

session Exemple 'wilshire';

setVarNames date cwprice cwdiv cwret ewprice ewdiv ewret;
setDataSet wilshire.asc;
setSeries cwret;

setInferenceType InvWald;

estimate modele1 arima(1,0,1);
estimate modele2 garch(1,1);

output file = fanpac3.out reset;

forecast 5;

output off;
```

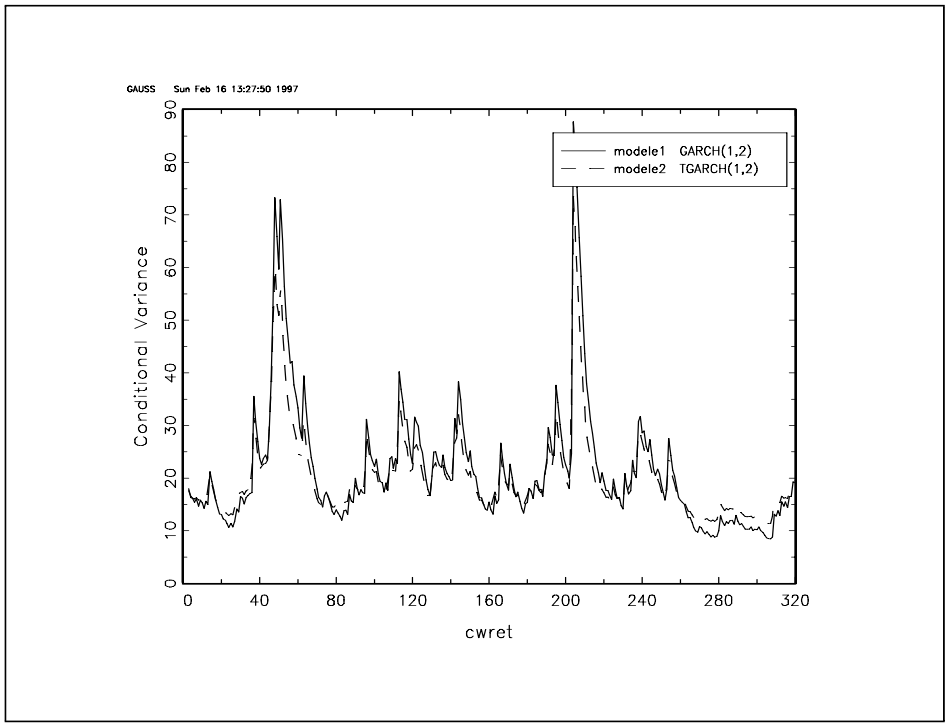
```
=====================================================
                      FORECAST
-----------------------------------------------------

-----------------------------------------------------
               modele1: ARIMA(1,0,1)
-----------------------------------------------------

          lower        time series          upper
          limit         forecast            limit

        -7.98346         0.81066         9.60477
        -7.41451         1.39905        10.21261
        -7.90156         0.92462         9.75080
        -7.52722         1.30716        10.14154
        -7.84098         0.99872         9.83842
-----------------------------------------------------

-----------------------------------------------------
               modele2: GARCH(1,1)
-----------------------------------------------------

          lower        time series          upper
          limit         forecast            limit

              .         1.10889               .
              .         1.10889               .
              .         1.10889               .
              .         1.10889               .
              .         1.10889               .
-----------------------------------------------------


------------------------------


       forecast of
   conditional variance

         20.17184
         20.24600
         20.31610
         20.38235
         20.44496

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

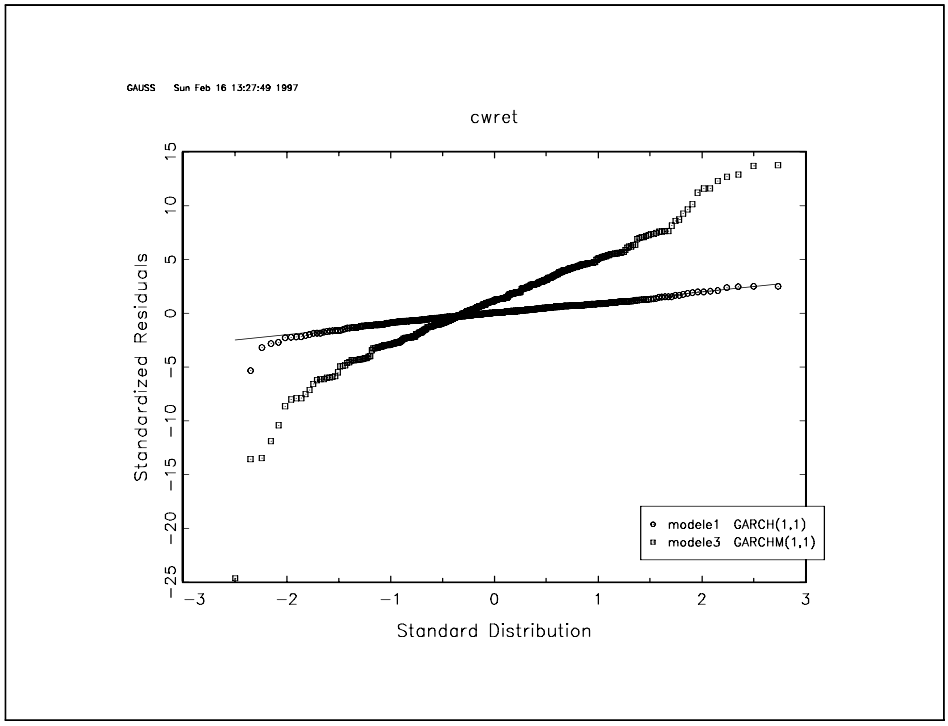
3.2.2 Exemple n°2

```
new;
```

```
library fanpac,pgraph;

session Exemple 'wilshire';

setVarNames date cwprice cwdiv cwret ewprice ewdiv ewret;
setDataSet wilshire.asc;
setSeries cwret;

plotSeries;
plotSeriesACF;
plotSeriesPACF;

setInferenceType InvWald;

estimate modele1 garch(1,2);
estimate modele2 tgarch(1,2);
estimate modele3 garchm(1,1);

plotQQ modele1 modele3;
plotCV modele1 modele2;
plotSR;
```







4 SPT

SPT est une bibliothèque GAUSS pour la modélisation de la structure par terme. Celle-ci peut être téléchargée à l'URL suivante :

http://www.montesquieu.u-bordeaux.fr/u/roncalli/spt.html

Elle comprend plus de 70 procédures et 200 exemples. La liste des procédures est la suivante :

- **affichageArbre** : Procédure d'affichage d'un arbre binomial.
- **ArbreHoLee** : Calcul des valeurs des coupons zéro $P_i^{(n)}(\tau)$ dans le modèle de Ho et Lee [1985].
- **BDFS** : Modèle de Balduzzi, Das, Foresi et Sundaram [1996].
- **BDT** : Modèle de Black, Derman et Toy [1990].
- **BlackKarasinski** : Modèle de Black et Karasinski [1991].
- **calendrier_31jpm** : Construction d'une série temporelle multidimensionnelle en tenant compte de 31 jours par mois.
- **callBlackScholes** : Calcul de la prime d'achat d'une option européenne dans le modèle de Black et Scholes [1973] lorsque le taux d'intérêt $r(t)$ est stochastique.
- **CIR** : Modèle de Cox, Ingersoll et Ross [1985b].
- **CouponZero_to_Obligation** : Création d'une base de données d'obligations (format **SPT**) à partir d'obligations à coupon zéro.
- **creationObligation** : Création d'une base de données d'obligations (format **SPT**) à partir des fichiers *.inf* et *.prx*.
- **csv2spt** : Conversion d'une base de données DATASTREAM en fichier *.prx*.
- **DebandtLesne** : Estimation des coefficients du modèle de DeBandt et Lesne [1991].
- **DebandtLesne_CouponZero** : Calcul des prix des coupons zéro $P_t^c(\tau)$ (modèle de DeBandt et Lesne [1991]).
- **DebandtLesne_TauxATerme** : Calcul des taux à terme $F_t(\tau, m)$ (modèle de DeBandt et Lesne [1991]).
- **DebandtLesne_TauxZero** : Calcul des taux zéro $R_t(\tau)$ (modèle de DeBandt et Lesne [1991]).
- **DebandtLesne2** : Estimation des coefficients du modèle généralisé de DeBandt et Lesne [1991].
- **DebandtLesne2_CouponZero** : Calcul des prix des coupons zéro $P_t^c(\tau)$ (modèle généralisé de DeBandt et Lesne [1991]).
- **DebandtLesne2_TauxATerme** : Calcul des taux à terme $F_t(\tau, m)$ (modèle généralisé de DeBandt et Lesne [1991]).
- **DebandtLesne2_TauxZero** : Calcul des taux zéro $R_t(\tau)$ (modèle généralisé de DeBandt et Lesne [1991]).
- **Decodage_Obligation** : Extraction des informations (prix, valeur et maturité des coupons) de la n-ième obligation d'un base de données d'obligations (format **SPT**).
- **ei_EDS** : Estimation des paramètres d'une équation différentielle stochastique par la méthode de l'Inférence Indirecte.

- **gmm_EDS** : Estimation des paramètres d'une équation différentielle stochastique (méthode des moments généralisés).
- **gmm_MBG** : Estimation des paramètres d'un mouvement brownien géométrique (méthode des moments généralisés).
- **gmm_OU** : Estimation des paramètres d'un processus d'Ornstein-Uhlenbeck (méthode des moments généralisés).
- **gVasicek** : Calcul de la fonction $b_t(\tau)$ du modèle généralisé de Vasicek (HULL et WHITE [1990]).
- **HeathJarrowMorton** : Simulation du taux d'intérêt $r(t)$ dans le modèle de HEATH, JARROW et MORTON [1992] sous la mesure de probabilité neutre au risque.
- **HoLeeCAP** : Calcul des valeurs du payoff d'un cap dans le modèle de HO et LEE [1985]
- **HoLeeFLOOR** : Calcul des valeurs du payoff d'un floor dans le modèle de HO et LEE [1985]
- **HoLeeTaux** : Calcul des taux d'intérêt $r_i^{(n)}(\tau)$ dans le modèle de HO et LEE [1985].
- **HoLeeValorisation** : Valorisation d'un actif contingent dans le modèle de HO et LEE [1985] par la technique de remontée de l'arbre.
- **HullWhite** : Modèle de HULL et WHITE [1993].
- **Interpolation_Donnees** : Interpolation spline cubique d'une série temporelle multidimensionnelle.
- **intsimpHJM** : Calcul de l'intégrale $\int_{t_0}^{t} \sigma_n(u,t) \left[\int_u^t \sigma_n(u,v)\, dv\right] du$ par l'algorithme de Simpson (modèle de HEATH, JARROW et MORTON [1992]).
- **Lire_EDP** : Procédure de lecture de la base de données contenant la solution de l'équation de valorisation du prix du coupon zéro.
- **LongstaffSchwartz** : Modèle de LONGSTAFF et SCHWARTZ [1992].
- **_maturite_entre_2dates** : Calcul de la maturité (en années) entre deux dates t_1 et t_2.
- **mc_SPT** : Résolution numérique d'un modèle de structure par terme à une variable d'état par la méthode de Monte Carlo (algorithme de Euler-Maruyama).
- **mc_SPTm** : Résolution numérique d'un modèle de structure par terme à plusieurs variables d'état par la méthode de Monte Carlo (schéma de Taylor de forme forte à l'ordre 0.5).
- **Merton** : Modèle de MERTON [1973].
- **ml_EDS** : Estimation des paramètres d'une équation différentielle stochastique (méthode du maximum de vraisemblance).
- **ml_MBG** : Estimation des paramètres d'un mouvement brownien géométrique (méthode du maximum de vraisemblance).

- **ml_OU** : Estimation des paramètres d'un processus d'Ornstein-Uhlenbeck (méthode du maximum de vraisemblance).
- **NelsonSiegel** : Estimation des coefficients du modèle de NELSON et SIEGEL [1987].
- **NelsonSiegel_** : Procédure **NelsonSiegel** qui ne nécessite pas la bibliothèque **TSM**.
- **NelsonSiegel_CouponZero** : Calcul des prix des coupons zéro $P_t^c(\tau)$ (modèle de NELSON et SIEGEL [1987]).
- **NelsonSiegel_Inversion** : Calcul de $\tau^{\#}(m)$ (modèle de NELSON et SIEGEL [1987]).
- **NelsonSiegel_Pente** : Calcul de la pente $p_t(\tau)$ de la courbe des taux (modèle de NELSON et SIEGEL [1987]).
- **NelsonSiegel_SPN** : Calcul du spread S_t, de la pente p_t et du niveau $\bar{R}_t(\tau_1, \tau_2)$ de la courbe des taux (modèle de NELSON et SIEGEL [1987]).
- **NelsonSiegel_TauxATerme** : Calcul des taux à terme $F_t(\tau, m)$ (modèle de NELSON et SIEGEL [1987]).
- **NelsonSiegel_TauxZero** : Calcul des taux zéro $R_t(\tau)$ (modèle de NELSON et SIEGEL [1987]).
- **optionCouponZero** : Prime d'une option européenne sur une obligation à coupon zéro (modèle de JAMSHIDIAN [1990]).
- **selectionArbre** : Sélection du vecteur des valeurs du j-ième arbitrage d'un arbre binomial.
- **selectionArbreHoLee** : Sélection de l'arbre binomial de maturité τ dans le modèle de HO et LEE [1985].
- **simulation_EDS** : Simulation d'une équation différentielle stochastique (algorithme de Euler-Maruyama).
- **simulation_EDSm** : Simulation d'une équation différentielle stochastique multidimensionnelle (schéma de Taylor de forme forte à l'ordre 0.5).
- **simulation_MBG** : Simulation d'un mouvement brownien géométrique (méthode exacte).
- **simulation_OU** : Simulation d'un processus d'Ornstein-Uhlenbeck (méthode exacte).
- **solution_EDP** : Résolution numérique d'un modèle de structure par terme à une variable d'état par les méthodes des différences finies et des θ-schémas.
- **SPT** : Initialisation d'un modèle de structure par terme à une variable d'état.
- **SPTm** : Initialisation d'un modèle de structure par terme à plusieurs variables d'état.
- **SPTset** : Initialisation des variables globales à leurs valeurs par défaut.
- **Svensson** : Estimation des coefficients du modèle de SVENSSON [1994].
- **Svensson_** : Procédure **Svensson** qui ne nécessite pas la bibliothèque **TSM**.
- **Svensson_CouponZero** : Calcul des prix des coupons zéro $P_t^c(\tau)$ (modèle de SVENSSON [1994]).

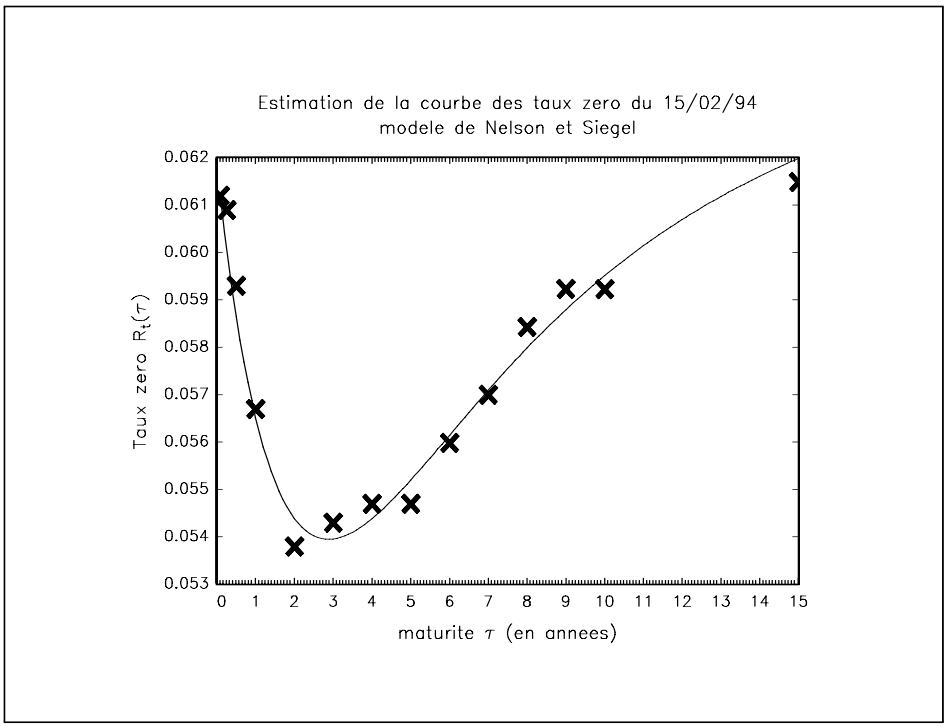
- **Svensson_TauxATerme** : Calcul des prix des taux à terme $F_t(\tau, m)$ (modèle de SVENSSON [1994]).
- **Svensson_TauxZero** : Calcul des taux zéro $R_t(\tau)$ (modèle de SVENSSON [1994]).
- **Vasicek** : Modèle de VASICEK [1977].

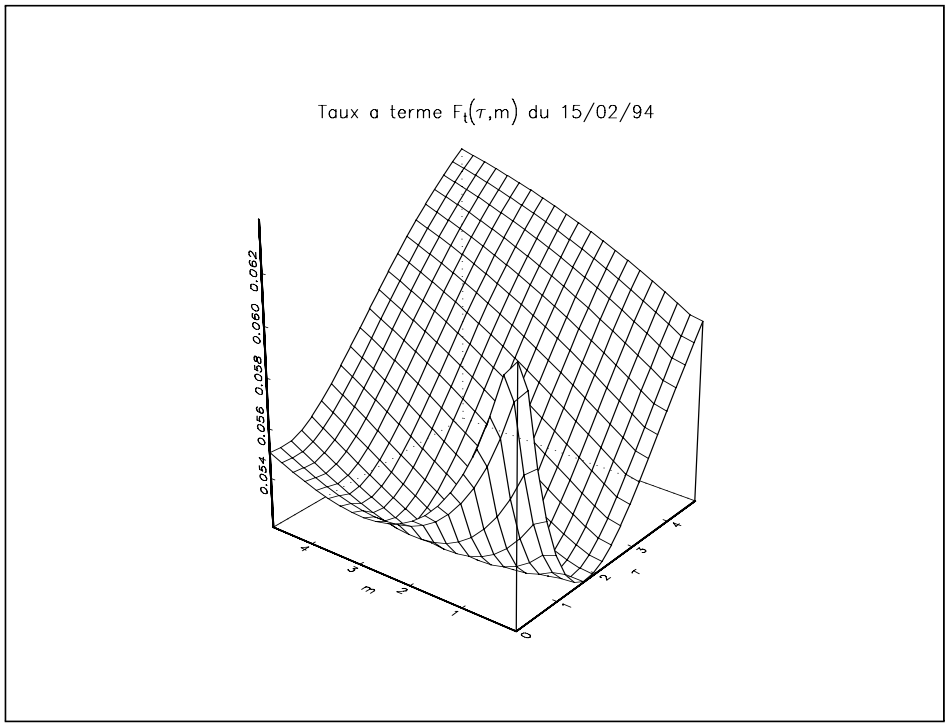
4.1 Modélisation de la structure par terme

Nous pouvons estimer la structure par terme à partir de différentes méthodes (Nelson-Siegel, Debandt-Lesne, Nelson-Siegel augmenté ou Svenssen, spline polynomiale). L'estimation peut être réalisée à partir d'obligations zéro coupon ou à partir d'un ensemble d'obligation (oat, btan, etc). Pour chacune des méthodes, nous pouvons calculer les taux zéro, les taux à terme et les coupons zéro.

```
new;
library spt,optmum,pgraph;
SPTset;

/* Chargement des donnees relatives aux ReuterBonds */

maturite = loadd(spt_data_path $+ "mat_frf");
y = loadd(spt_data_path $+ "frf");
Dates = y[.,1]; Pc = trimr(y',1,0)';

/* Selection d'une date */

Pc = selif(Pc, Dates .== 19940215); Pc = Pc';

/* Creation d'une base de donnees d'obligations */

o150294 = CouponZero_to_Obligation(Pc,maturite);

/* Estimation - modele de Nelson et Siegel */

c = NelsonSiegel_(o150294,1,0,0);

/* Courbe des taux zero */

tau = seqa(1/10,1/10,300);
TauxZero = NelsonSiegel_TauxZero(c,tau);

yield = -ln(Pc)./maturite;
Nobs = rows(yield);

graphset;
  _pdate = ""; _pnum = 2;
  fonts("simplex simgrma");
  title("Estimation de la courbe des taux zero du 15/02/94"\
        "\Lmodele de Nelson et Siegel");
  _psym = maturite~yield~ones(Nobs,1).*(14~8~2~1~1);
```

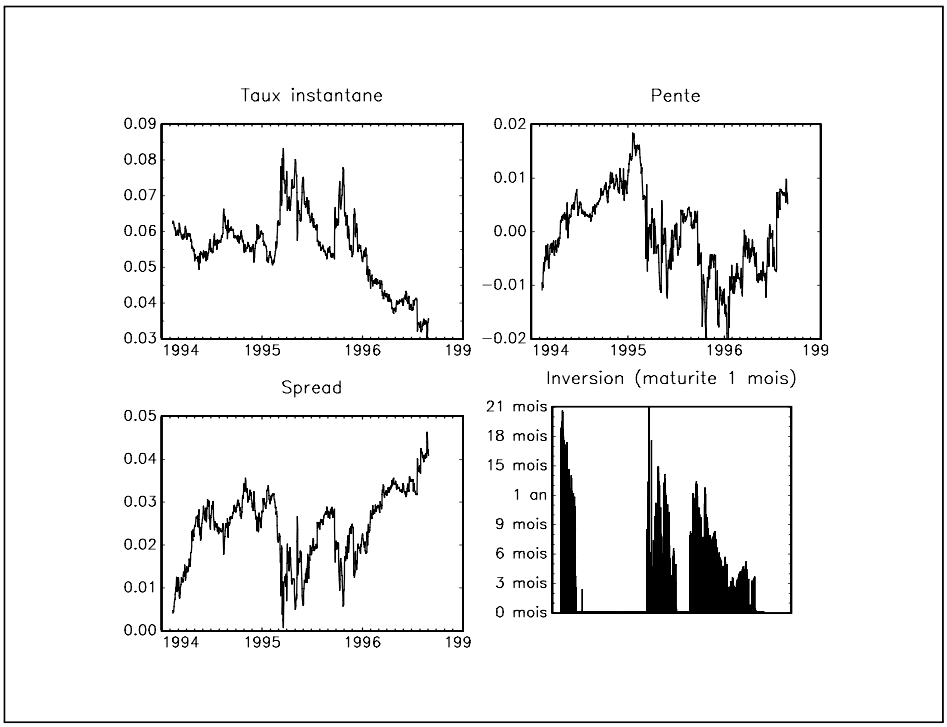
```
  xlabel("maturite \202t\201 (en annees)");
  ylabel("Taux zero R]t[(\202t\201)");
  xtics(0,15,1,12);
  ytics(0.053,0.062,0.001,0);
  graphprt("-c=1 -cf=spt1.eps -w=5");
  xy(tau,TauxZero);


/* Courbe des taux a terme */

tau = 0|seqa(1/12,3/12,4*5); m = tau;
TauxATerme = NelsonSiegel_TauxATerme(c,tau',m);

graphset;
  fonts("simplex simgrma");
  title("Taux a terme F]t[\202\34t\201,m\202\118\201"\
        " du 15/02/94");
  _pdate = ""; _pframe = 0;
  xlabel("\202t\201");
  ylabel("m");
  graphprt("-c=1 -cf=spt2.eps -w=5");
  surface(tau',m,TauxATerme);
```






```
new;
library spt,optmum,pgraph;
SPTset;

fichier = spt_data_path $+ "NS_frf"; load NS_frf = ^fichier;
Dates = NS_frf[.,1];

r0 = NelsonSiegel_TauxZero(NS_frf[.,2 3 4 5]',0.000001)';
{S,P,N} = NelsonSiegel_SPN(NS_frf,1/100,1);
tauD = NelsonSiegel_Inversion(NS_frf,1/12);

data = Dates~r0~P~S~tauD;
data = Interpolation_Donnees(calendrier_31jpm(Data,19940101,19961231));
Nobs = rows(data);

graphset;

  _pdate = ""; _pnum = 2; _pnumht = 0.25; _paxht = 0.25; _ptitlht = 0.30;
  let xlab = "1994" "1995" "1996" "1997";
  asclabel(xlab,0);
  xtics(1,Nobs,31*12,12);

  begwind;
  window(2,2,0);

  setwind(1);
    title("Taux instantane");
    xy(seqa(1,1,Nobs),data[.,2]);
```

```
setwind(2);
  title("Pente");
  ytics(-0.02,0.02,0.01,0);
  xy(seqa(1,1,Nobs),data[.,3]);

setwind(3);
  title("Spread");
  ytics(0.00,0.05,0.01,0);
  xy(seqa(1,1,Nobs),data[.,4]);

setwind(4);
  title("Inversion (maturite 1 mois)");
  let ylab = "0 mois" "3 mois" "6 mois" "9 mois" "1 an" "15 mois"
             "18 mois" "21 mois";
  let xlab = " " " " " " " " " ";
  asclabel(xlab,ylab);
  axmargin(2,1,1,1);
  ytics(0,21/12,3/12,3);
  bar(seqa(1,1,Nobs),data[.,5]);

  graphprt("-c=1 -cf=spt3.eps");
endwind;
```



- De nombreux modèles théoriques sont disponibles dans SPT.
- De nombreuses applications sont fournies avec SPT (prévision d'inflation, mesure de la crédibilité, etc).

4.2 Valorisation des actifs contingents au taux d'intéret

Les principaux modèles de **SPT** sont : Black-Derman-Toy, Black-Karasinski, Heath-Jarrow-Morton, Ho-Lee et Hull-White (modèle trinomial à un seul facteur). Pour le modèle continu HJM, la procédure `HeathJarrowMorton` simule les trajectoires du taux d'intérêt sous la mesure de probabilité neutre au risque. La valorisation des actifs se fait alors par la méthode de Monte Carlo. Pour les modèles binomiaux et trinomiaux, la valorisation se fait par la technique de remontée de l'arbre.

4.2.1 Un exemple BDT

```
new;
library spt;
SPTset;

/* Exemple de Black-Derman-Toy [1990] */

let TauxZero = 0.10 0.11 0.12 0.125 0.13;
let Volatilite =  0.19 0.18 0.17 0.16;

_print = 0;
R = BDT(TauxZero,Volatilite);

output file = spt3.out reset;

outwidth 256;
Msym " "; format 11,6;
call affichageArbre(R);
msym ".";

output off;
```

```
                                                  0.280077
                                        0.228404
                              0.196941                    0.203377
                    0.143180                    0.162571
   0.100000                   0.137401                    0.147682
               0.0979156                   0.115713
                          0.0958616                    0.107239
                                        0.0823614
                                                  0.0778717
```

4.2.2 Un exemple HW

```
new;
library spt;

tau = seqa(1,1,5);
TauxZero = 0.08-0.05*exp(-0.18*tau);
```

```
CouponZero = exp(-tau.*TauxZero);
a = 0.1;
sigma = 0.01;
deltaT = 1;

Msym " ";
outwidth 256;  format 11,6;

_print = 1;

output file = spt4.out reset;

arbreHW = HullWhite(CouponZero,a,sigma,deltaT);
call affichageHW(arbreHW);
Msym ".";

output off;
```

```
                          (2,2)       (3,2)       (4,2)
                (1,1)     (2,1)       (3,1)       (4,1)
      (0,0)     (1,0)     (2,0)       (3,0)       (4,0)
                (1,-1)    (2,-1)      (3,-1)      (4,-1)
                          (2,-2)      (3,-2)      (4,-2)
```

Noeud precedent i	Noeud precedent j	Noeud suivant i	Noeud suivant j	Probabilite p	Taux d'interet r
		0	0	1.000000	0.038236
0	0	1	1	0.166667	0.069366
0	0	1	0	0.666667	0.052046
0	0	1	-1	0.166667	0.034725
1	1	2	2	0.121667	0.097177
1	1	2	1	0.656667	0.079856
1	1	2	0	0.221667	0.062536
1	0	2	1	0.166667	0.079856
1	0	2	0	0.666667	0.062536
1	0	2	-1	0.166667	0.045215
1	-1	2	0	0.221667	0.062536
1	-1	2	-1	0.656667	0.045215
1	-1	2	-2	0.121667	0.027895
2	2	3	2	0.886667	0.105070
2	2	3	1	0.026667	0.087749
2	2	3	0	0.086667	0.070429
2	1	3	2	0.121667	0.105070
2	1	3	1	0.656667	0.087749
2	1	3	0	0.221667	0.070429

```
    2     0         3     1          0.166667      0.087749
    2     0         3     0          0.666667      0.070429
    2     0         3    -1          0.166667      0.053108
    2    -1         3     0          0.221667      0.070429
    2    -1         3    -1          0.656667      0.053108
    2    -1         3    -2          0.121667      0.035788
    2    -2         3     0          0.086667      0.070429
    2    -2         3    -1          0.026667      0.053108
    2    -2         3    -2          0.886667      0.035788
    3     2         4     2          0.886667      0.110940
    3     2         4     1          0.026667      0.093620
    3     2         4     0          0.086667      0.076299
    3     1         4     2          0.121667      0.110940
    3     1         4     1          0.656667      0.093620
    3     1         4     0          0.221667      0.076299
    3     0         4     1          0.166667      0.093620
    3     0         4     0          0.666667      0.076299
    3     0         4    -1          0.166667      0.058979
    3    -1         4     0          0.221667      0.076299
    3    -1         4    -1          0.656667      0.058979
    3    -1         4    -2          0.121667      0.041658
    3    -2         4     0          0.086667      0.076299
    3    -2         4    -1          0.026667      0.058979
    3    -2         4    -2          0.886667      0.041658

                          0.0971769    0.105070    0.110940
              0.0693664   0.0798564   0.0877495   0.0936199
  0.0382365   0.0520459   0.0625359   0.0704290   0.0762994
              0.0347254   0.0452154   0.0531085   0.0589788
                          0.0278949   0.0357880   0.0416583
```

4.2.3 Un exemple de valorisation

Pour chacun des modèles, il existe plusieurs procédures pour déterminer l'arbre d'évolution, pour valoriser un actif contingent de type européen et pour valoriser les caps et floors européens. Il est très facile de s'inspirer des procédures existantes pour valoriser d'autres actifs (swaption par exemple) et de nature différente (option américaine par exemple). Dans l'exemple suivant, nous valorisons l'option d'achat Matif-Pibor 3 mois avec le modèle de Hull et White[2].

```
new;
library spt,pgraph;

/* Caracteristiques de l'option d'achat du 02/01/1992 */

C = 0.15; F0 = 90.25; K = 90.3; tau = 0.20;  sigma = 0.01093;

/* Prime theorique de Black */

r = 0.0975;
```

[2]Nous ne tenons pas compte du caractère américain de l'option.

```
w = sigma.*sqrt(tau);
d1 = ln(F0./K)./w + 0.5*w;
d2 = d1 - w;
PrimeBlack = F0.*exp(-r.*tau).*cdfn(d1)-K.*exp(-r.*tau).*cdfn(d2);

/* Primes theoriques Hull-White */

a = 0.01|seqa(0.1,0.1,19);
deltaT = tau/5;
t = seqa(deltaT,deltaT,5);

TauxZero = 0.0975*ones(5,1);
CouponZero = exp(-t.*TauxZero);
C1 = zeros(rows(a),1);

i = 1;
do until i > rows(a);
  C1[i] = optionFuture(CouponZero,a[i],sigma,deltaT,K);
  i = i + 1;
endo;

TauxZero = 0.0975 | 0.0974 | 0.0973 | 0.0972 | 0.0971;
CouponZero = exp(-t.*TauxZero);
C2 = zeros(rows(a),1);

i = 1;
do until i > rows(a);
  C2[i] = optionFuture(CouponZero,a[i],sigma,deltaT,K);
  i = i + 1;
endo;

TauxZero = 0.0975 | 0.0976 | 0.0977 | 0.0978 | 0.0979;
CouponZero = exp(-t.*TauxZero);
C3 = zeros(rows(a),1);

i = 1;
do until i > rows(a);
  C3[i] = optionFuture(CouponZero,a[i],sigma,deltaT,K);
  i = i + 1;
endo;


graphset;
  _pdate = ""; _pnum = 2; _paxht = 0.20;
  title("Influence de la structure par terme sur la prime"\
        "\Ld'une option d'achat sur Matif-Pibor future 3 mois");
  _plctrl = -1; _psymsiz = 6;
  _plegstr = "spt plate\000spt descendante\000spt ascendante";
  _plegctl = {2 6 5.5 4.25};
  xlabel("a");
```

```
  ytics(0.10,0.20,0.02,0);
  _pline = 1~6~0~PrimeBlack~100~PrimeBlack~1~15~10;
  _pmsgstr = "Prime Theorique de Black";
  _pmsgctl = 0.3~0.152~0.2~0~1~7~6;
  graphprt("-c=1 -cf=spt5.eps");
  xy(a,C1~C2~C3);


proc optionFuture(CouponZero,a,sigma,deltaT,K);
  local arbreHW,G,payoff,C,arbreC;

  _print = 0;
  arbreHW = HullWhite(CouponZero,a,sigma,deltaT);

  G = (100*(1-arbreHW[.,6]) - K);
  G = G.*(G.>0);
  payoff = arbreHW[.,3 4]~G;

  {C,arbreC} = HWvalorisation(arbreHW,deltaT,payoff);
  C = C[rows(CouponZero)];

  retp(C);
endp;
```