Séminaire **GAUSS**

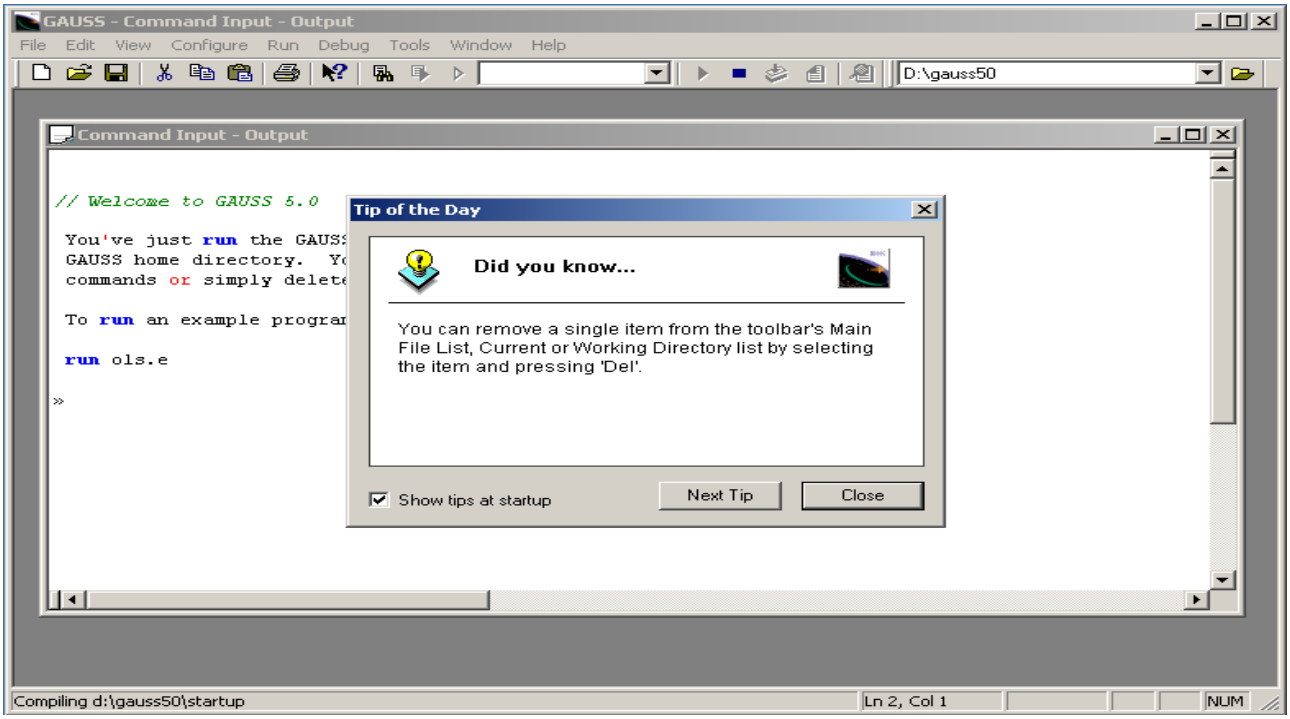
Thierry Roncalli
Groupe de Recherche Opérationnelle du Crédit Lyonnais
Bercy-Expo — Immeuble Bercy SUD — 4^e étage
90, Quai de Bercy — 75613 Paris Cedex 12

Octobre 2002

Table des matières

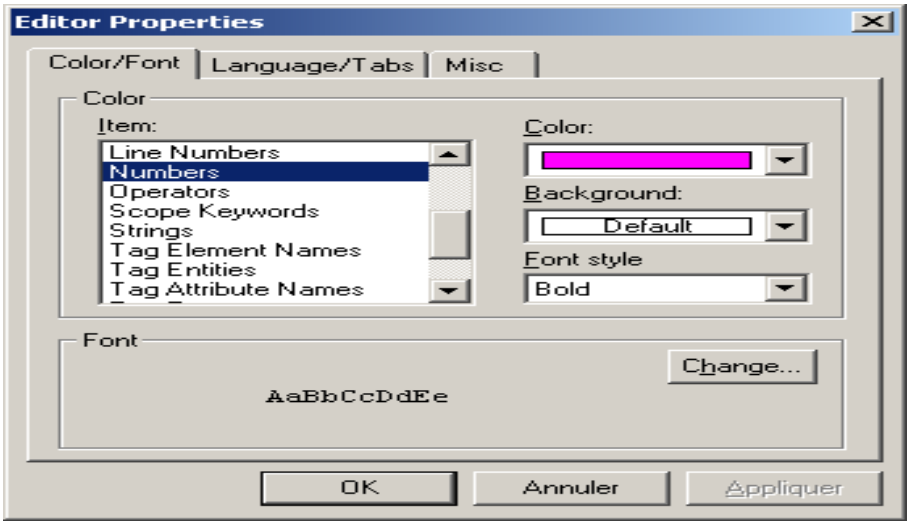
Première partie

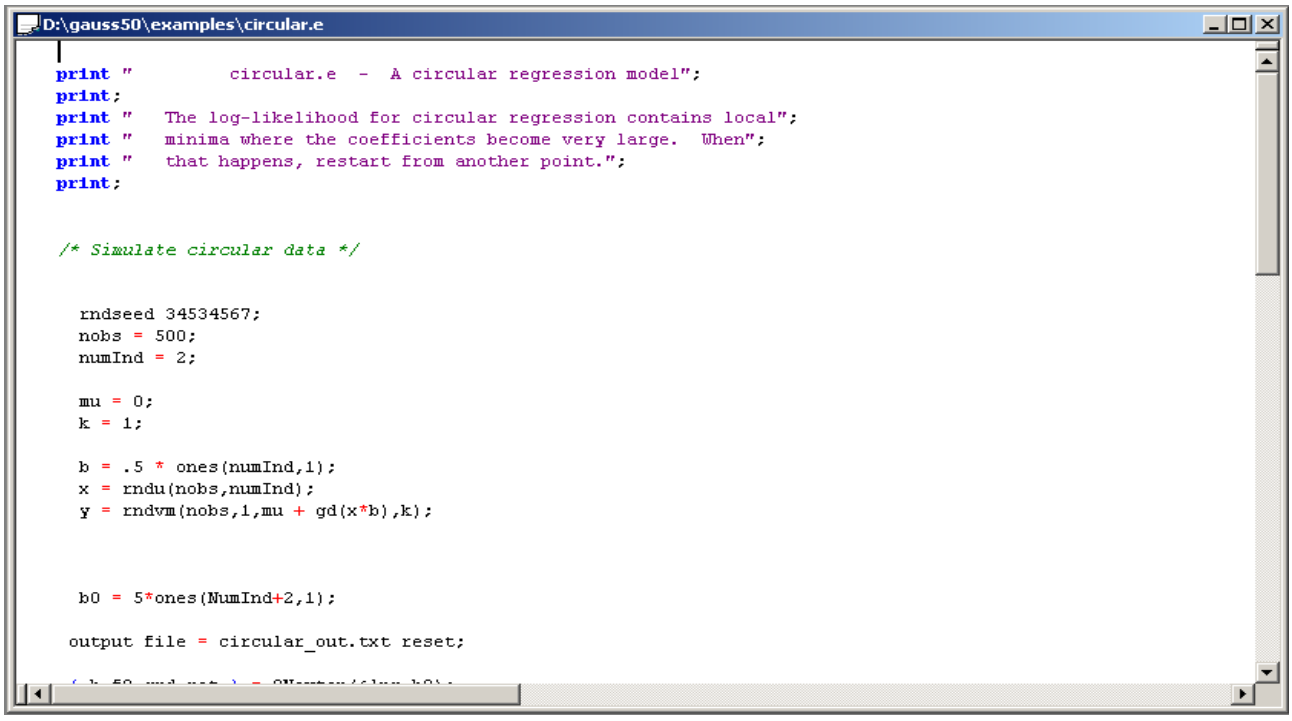
Présentation de l'environnement de développement



1 L'éditeur

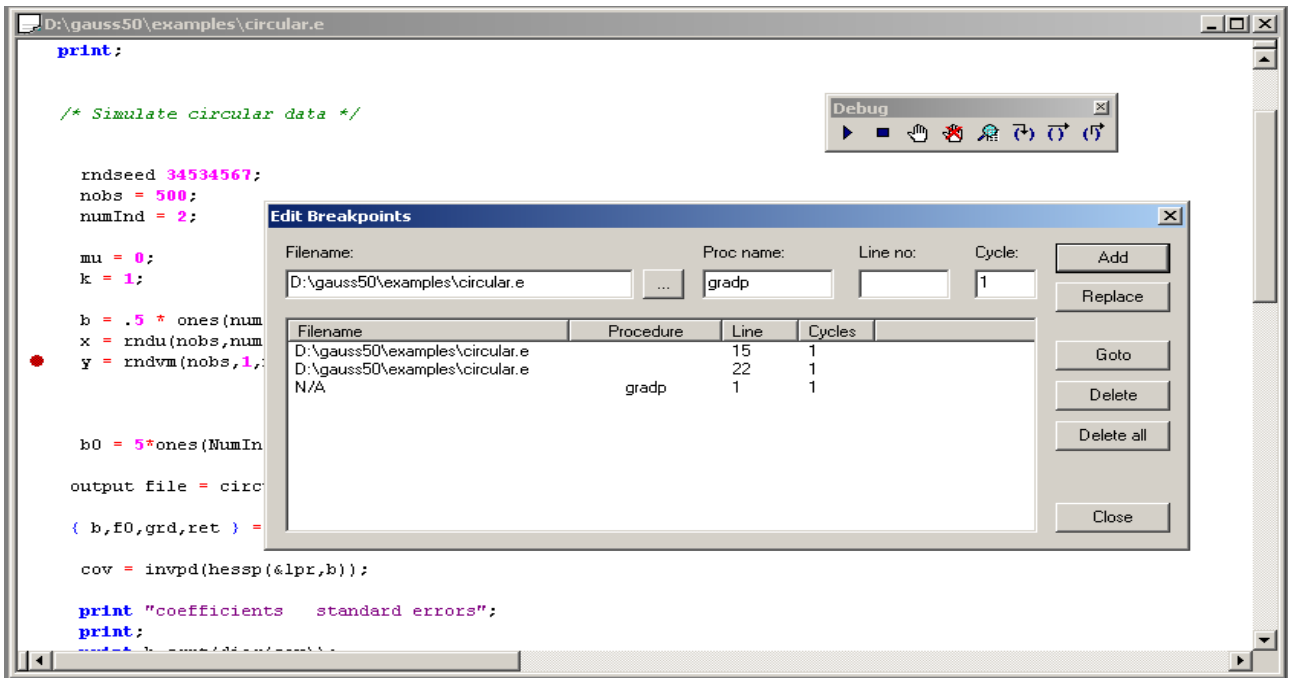
Par rapport aux versions 3.2 et 3.5, l'éditeur a été fortement modifié. Par exemple, celui-ci supporte la syntaxe surlignée, qui peut être configurée (**C**onfigure /**E**ditor properties).



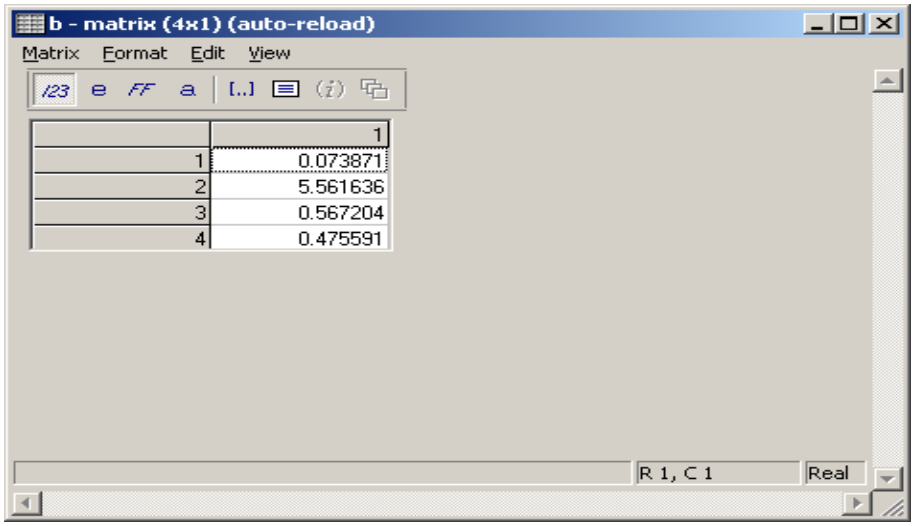


2 Le debogueur

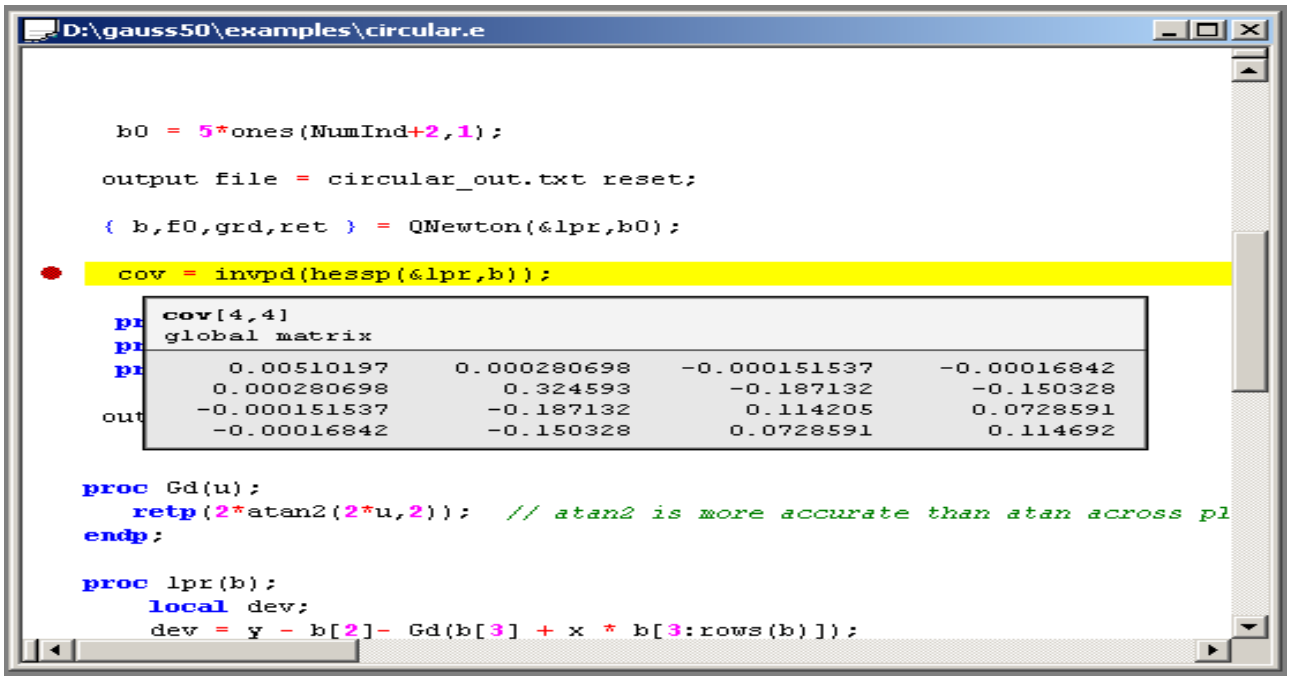
Le nouveau debogueur est très proche de celui fourni avec les versions 3.5 et 3.6 de **GAUSS**. Deux types de points d'arrêt sont disponibles : **Procedure breakpoints**, **Line number breakpoints**.



Dans le cas d'un point d'arrêt de type procédure, le debogueur s'arrête lorsque la procédure est exécutée. De même, trois types d'exécution sont possibles : Step Into, Step Over, Step Out. On peut par exemple choisir de rentrer ou non dans une procédure. Toutes les variables (globales et locales) sont éditables avec l'éditeur matriciel.

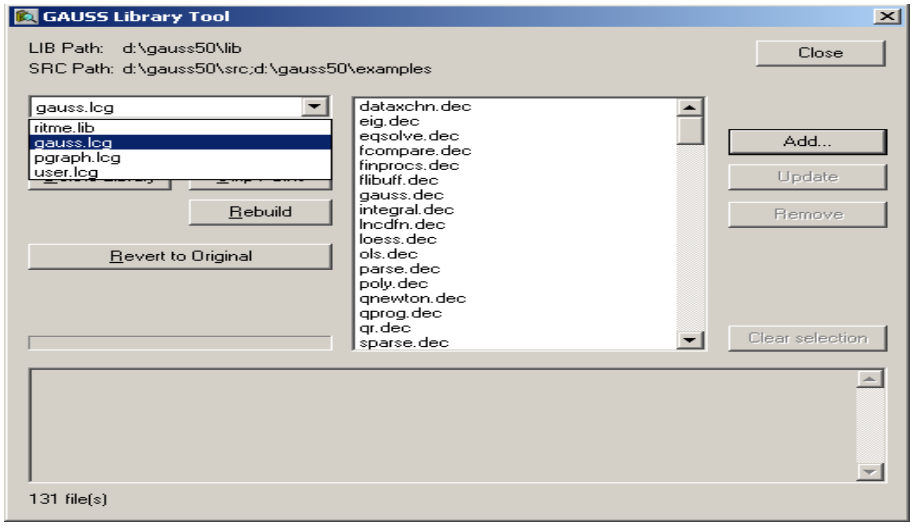


Nous pouvons visualiser rapidement une variable en positionnant la souris sur celle-ci.



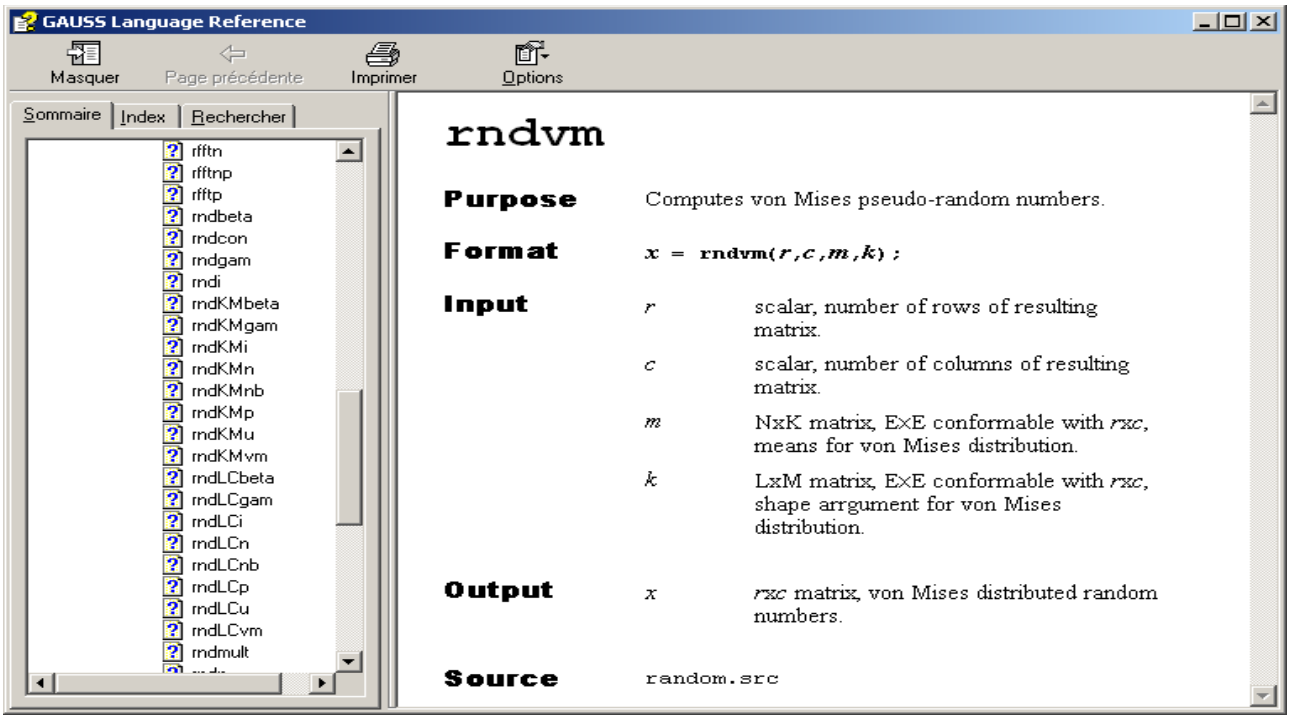
3 L'outil **Lib Tool** de gestion des bibliothèques

Dans les versions antérieures, la gestion des bibliothèques se faisait exclusivement avec la commande `lib`. Désormais, **GAUSS** est livré avec un utilitaire Lib Tool qui permet de gérer les bibliothèques de façon plus conviviale. Les options de la commande `lib` (`-update`, `-build`, `-delete`, `-list`, `-addpath`, `-gausspatch`, `-leavepath`, `-nopath`) sont disponibles dans Lib Tool.

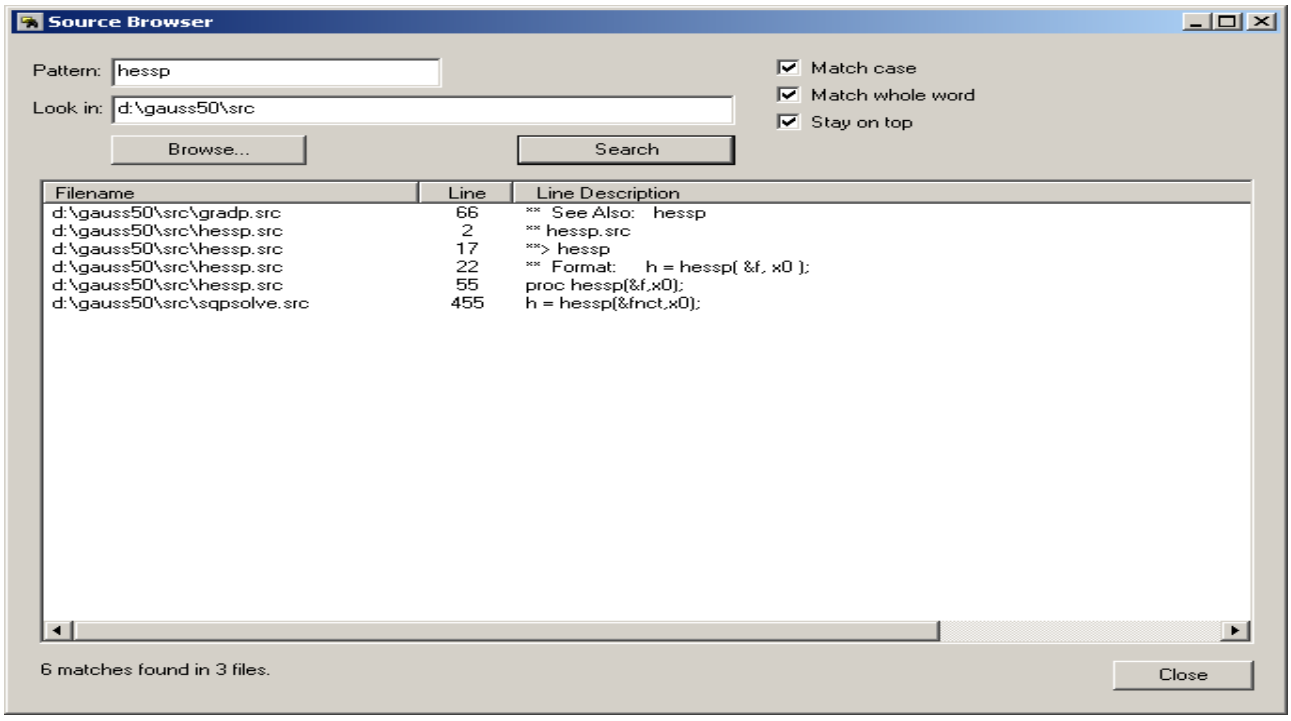


4 L'aide

Il existe deux types d'aide dans **GAUSS**. Le premier type est au format HTML. On y accède par le menu Help (**H**elp /**R**eference) ou en utilisant la touche **F1**.



Pour obtenir l'aide en ligne, nous utilisons le browser ou la touche **CTRL-F1**.



Deuxième partie

Le langage GAUSS

5 Les matrices

GAUSS est un langage compilé matriciel typé complexe. Il n'est donc pas nécessaire de déclarer les variables.

Voyons un exemple. Nous cherchons à calculer

$$\mathcal{E} = \mathbb{E}\left[X_1 X_2^2 + X_3^2 \mid (X_3, X_4) \notin \mathcal{D}\left((0,0), 1\right)\right]$$

avec (X_1, X_2, X_3, X_4) un vecteur aléatoire gaussien standard de matrice de corrélation ρ et $\mathcal{D}\left((0,0), 1\right)$ le disque centré en $(0,0)$ et de rayon 1.

Une solution possible est la suivante :

```
new;

proc (1) = rndmn_cr(rho,ns);
  retp( rndn(ns,rows(rho))*chol(rho) );
endp;

ns = 50000;

let rho[4,4] = 1.0 0.5 0.5 0.5
               0.5 1.0 0.5 0.5
```

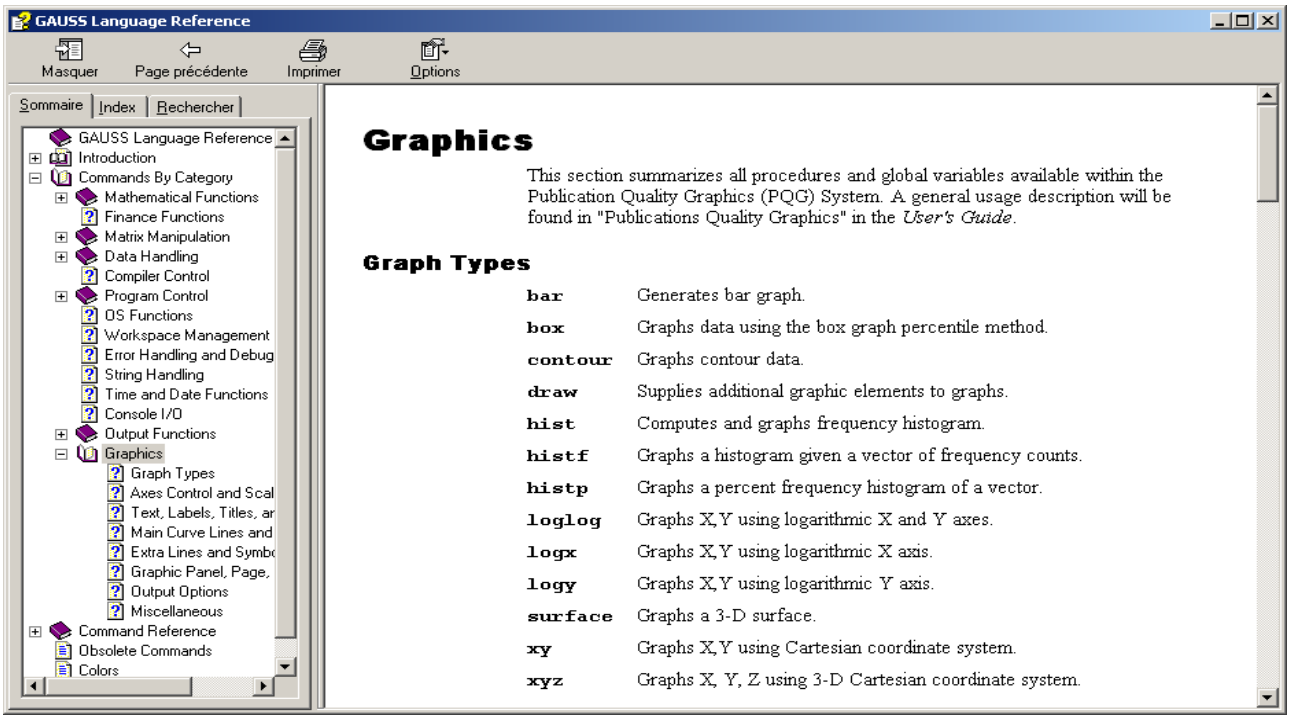
```
              0.5 0.5 1.0 0.5
              0.5 0.5 0.5 1.0;

x = rndmn_cr(rho,10000);

cnd = x[.,3]^2 + x[.,4]^2 .> 1^2;
E = sumc( cnd .* (x[.,1] .* x[.,2]^2 + x[.,3]^2) ) / sumc(cnd);
print E;
```

6 Un ensemble complet de procédures

GAUSS est livré avec une bibliothèque de fonctions. A titre d'information, la version de base comprend plus de 800 commandes et procédures.

```
cls;
print;
print "This Example creates a random 600x600 positive definite matrix";
print "inverts it, and prints how long it took to complete these steps.";
print;

rndseed 34546;

output file = time_results.txt on;

t0 = date;
x = rndu(700,600);
print "seconds to generate random matrix - " ethsec(t0,date)/100;

t0 = date;
y = moment(x,0);
print "seconds to compute cross-product - " ethsec(t0,date)/100;

t0 = date;
z = invpd(y);
print "seconds to compute inverse - " ethsec(t0,date)/100;

output off;

print;
print "Results appended to the time_results.txt file";
print "your working directory";
```

7 La bibliothèque graphique



8 Les structures

Pour déclarer une structure, nous utilisons la commande (ou le mot-clé) `struct`. Cinq types de variables peuvent être définis (comme pour la commande `declare`) :

1. **array**
2. **scalar**
3. **matrix**
4. **string**
5. **string array**

Voici un exemple de déclaration d'une structure à quatre membres :

```
struct EconometricModel {
matrix y;                   // Endogeneous variable
matrix x;                   // Exogeneous variables
string model;               // Model type = ols, nls
matrix f;                   // Pointer (function y = f(x;beta) + u)
};

fn fun(x,beta) = x[.,1]*beta[1] + x[.,2]^beta[2];

struct EconometricModel m1;
m1.f = &fun;
m1.model = ''nls'';
```

Une structure peut être un membre d'une autre structure :

```
struct person
{
string name;
string phone;
};

struct company
{
scalar id;
string name;
struct person boss;
};

struct company FT;

FT.name = ''France Telecom'';
FT.boss.name = ''Bon'';
```

Pour passer une structure dans une procédure, il faut utiliser le mot-clé `struct` dans les arguments. Une structure passée comme argument d'entrée dans une procédure est de type local. Si celle-ci est modifiée dans la procédure, alors **GAUSS** procède à une copie de la structure (ce qui est inefficient d'un point de vue programmation). Par exemple, il y a copie de la structure `cercle` dans la procédure `perimeter`, mais pas dans la procédure `area`.

```
new;

struct cercle
{
scalar x;
scalar y;
scalar r;
scalar area;
scalar perimeter;
};


proc (1) = area(struct cercle c);
  local r;
  r = c.r;
  retp(pi*r^2);
endp;

proc (1) = perimeter(struct cercle c);
  c.perimeter = 2*pi*c.r;
  retp(c.perimeter);
endp;

struct cercle rond;

rond.r = 1;
a = area(rond);

print a;
print rond.area;
```

```
p = perimeter(rond);
print p;
print rond.perimeter;

       3.1415927
      0.00000000
       6.2831853
      0.00000000
```

Une procédure peut retourner des structures. Il faut néanmoins que celles-ci soient déclarées.

```
new;

struct cercle
{
scalar x;
scalar y;
scalar r;
scalar area;
scalar perimeter;
};

proc (1) = area_perimeter(struct cercle c1);
  local r;
  struct cercle c2;

  r = c1.r;

  c2.x = c1.x;
  c2.y = c1.y;
  c2.r = r;

  c2.area = pi*r^2;
  c2.perimeter = 2*pi*r;

  retp(c2);
endp;

struct cercle rond;
rond.r = 1;

struct cercle c;

c = area_perimeter(rond);

print c.area;
print c.perimeter;


       3.1415927
       6.2831853
```

Remarque 1 *Nous pouvons utiliser les structures dans une bibliothèque. Pour cela, il est nécessaire de les*

dsCreate	Creates an instance of a structure of type DS set to default values.
pvCreate	Returns an initialized instance of a PV structure.
pvGetIndex	Gets row indices of a matrix in a parameter vector.
pvGetParNames	Generates names for parameter vector stored in PV structure.
pvGetParVector	Retrieves parameter vector from PV structure.
pvLength	Returns length of vector p.
pvList	Retrieves names of packed matrices in PV structure.
pvPack	Packs general matrix into a PV structure with matrix name.
pvPackm	Packs general matrix into a PV structure with a mask and matrix name.
pvPacks	Packs symmetric matrix into a PV structure.
pvPacksm	Packs symmetric matrix into a PV structure with a mask.
pvPutParVector	Inserts parameter vector into PV structure.
pvTest	Tests an instance of a PV structure to determine if it is a proper PV structure.
pvUnpack	Unpacks matrices stored in a PV structure.

TAB. 1 – Liste des commandes gérant les structures

déclarer. C'est pourquoi nous devons inclure leur définition dans le programme principal avec une directive de compilation : `# include ***.sdf`. *A titre d'exemple, voici le fichier ds.sdf de définition de la structure* `DS` *:*

```
struct DS {
           scalar type;
           matrix dataMatrix;
           array dataArray;
           string dname;
           string array vnames;
          };
```

Si le programme principal fait appel à une procédure avec des arguments de type `DS`*, nous devons rajouter la ligne de commande* `# include ds.sdf`.

9 Les tableaux multidimensionnels

Voyons quelques exemples d'utilisation des tableaux multidimensionnels (qui sont en fait des tenseurs).

```
new;

cls;

let array x[2,2,3] = 1 2 3 4 5 6 7 8 9 10 11 12;
print "x = " x;

y = exp(x) .* x .* cos(x);
print "y = " y;

z = y[1,.,.] + y[2,.,.];
print "z = " z;

w = aconcat(y[1,.,.],z,3);
print "w = " w;


x =
```

aconcat	Concatenates conformable matrices and arrays in a user-specified dimension.
amean	Computes the mean across one dimension of an N-dimensional array.
areshape	Reshapes a scalar, matrix, or array into an array of user-specified size.
arrayalloc	Creates an N-dimensional array with unspecified contents.
arrayinit	Creates an N-dimensional array with a specified fill value.
arraytomat	Changes an array to type matrix.
asum	Computes the sum across one dimension of an N-dimensional array.
atranspose	Transposes an N-dimensional array.
getarray	Gets a contiguous subarray from an N-dimensional array.
getdims	Gets the number of dimensions in an array.
getmatrix	Gets a contiguous matrix from an N-dimensional array.
getmatrix4D	Gets a contiguous matrix from a 4-dimensional array.
getorders	Gets the vector of orders corresponding to an array.
getscalar3D	Gets a scalar from a 3-dimensional array.
getscalar4D	Gets a scalar form a 4-dimensional array.
loopnextindex	Increments an index vector to the next logical index and jumps to the specified label if the index did not
mattoarray	Changes a matrix to a type array.
nextindex	Returns the index of the next element or subarray in an array.
previousindex	Returns the index of the previous element or subarray in an array.
putarray	Puts a contiguous subarray into an N-dimensional array and returns the resulting array.
setarray	Sets a contiguous subarray of an N-dimensional array.
walkindex	Walks the index of an array forward or backward through a specified dimension.

TAB. 2 – Liste des functions concernant les tableaux multidimensionnels

```
Plane [1,.,.]

       1.0000000        2.0000000        3.0000000
       4.0000000        5.0000000        6.0000000

Plane [2,.,.]

       7.0000000        8.0000000        9.0000000
       10.000000        11.000000        12.000000
y =

Plane [1,.,.]

       1.4686939       -6.1498646       -59.653593
      -142.75093        210.49601        2324.1620

Plane [2,.,.]

       5787.2795       -3469.8359       -66446.685
      -184817.80        2914.8336        1648095.3
z =

Plane [1,.,.]

       5788.7482       -3475.9858       -66506.338
      -184960.55        3125.3296        1650419.5
```

```
w =

Plane [1,.,.]

       1.4686939       -6.1498646       -59.653593
      -142.75093        210.49601        2324.1620

Plane [2,.,.]

       5788.7482       -3475.9858       -66506.338
      -184960.55        3125.3296        1650419.5
```

Un exemple de simulation d'une variable aléatoire χ_2.

```
new;

cls;

/*
** Simulation d'une matrice 1000*3 de chi2(1)
*/

r = 1000;
c = 3;

x = rndn(r,c);
chi2 = x^2;

print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant une boucle
*/

nu = 5;
chi2 = zeros(r,c);

for i (1,c,1);
  chi2[.,i] = sumc(rndn(nu,r)^2);
endfor;

print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant la commande reshape
*/

rndseed 123;
chi2 = reshape(sumc(rndn(nu,r*c)^2),r,c);
```

```
print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant les tableaux multidimensionnels
*/

rndseed 123;
x = rndn(r*c*nu,1);
orders = nu|r|c;

y = areshape(x,orders);
y = y .* y;

chi2 = arraytomat(asum(y,3));
print meanc(chi2)~stdc(chi2);


       1.0904490        1.5662331
       1.1000481        1.5862624
       1.0854098        1.4747488

       5.0120703        3.1569619
       4.8923906        3.2383112
       5.0366850        3.1582330

       4.9158512        2.9633891
       5.0314151        3.1005409
       5.0650645        3.2240497

       4.9158512        2.9633891
       5.0314151        3.1005409
       5.0650645        3.2240497
```

Dans le programme suivant, nous considrons la simulation d'un processus GBM multidimensionnel. **Notons que la simulation de plusieurs trajectoires nécessitent l'utilisation d'une boucle.**

```
new;
library pgraph;

/*
** Modele GBM multidimensionnel
**
** dX_i(t) = \mu_i * X_i(t) * dt + \sigma_i * X_i(t) * dW_i(t), i = 1,...,n
**
** E[W_i(t)W_j(t)] = rho_{i,j}
**
**
*/

proc (1) = simulate_mGBM(x0,mu,sigma,rho,t);
```

```
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = zeros(Nt,Nx);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(1,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
    x[i,.] = x0;

    i = i + 1;
  endo;

  retp(x);
endp;

let x0 = 100 50 75;
let mu = 0.05 0.07 0.08;
let sigma = 0.15 0.25 0.05;

let rho = {1,
           0.5, 1,
           0.25, 0.15, 1};

rho = xpnd(rho);  // transforme le vecteur 'vech(rho)' en matrice de correlation

t = seqa(0,1/365,5*365+1);

x = simulate_mGBM(x0,mu,sigma,rho,t);

graphset;
  _pdate = ""; _pframe = 0; _pnum = 2;
  _pltype = 6|1|3; _plwidth = 5;
  title("\214GBM Simulation");
  xlabel("\214time");
```

```
  xtics(0,5,1,12);
  xy(t,x);


/*
** Si nous voulons obtenir non pas une seule trajectoire simulee du processus 3D
** mais plusieurs trajectoires, nous pouvons utiliser une boucle
**
*/

Ns = 1000;
Nt = rows(t);

x1 = zeros(Nt,Ns); x2 = x1; x3 = x1;

t0 = hsec;

for i (1,Ns,1);
  x = simulate_mGBM(x0,mu,sigma,rho,t);
  x1[.,i] = x[.,1];
  x2[.,i] = x[.,2];
  x3[.,i] = x[.,3];
endfor;

ct = (hsec-t0)/100;

print ftos(ct,"Computational time : %f seconds.",5,2);

graphset;
  _pltype = 3;
  xy(t,x1[.,1:100]);   // Graphe des 100 premieres trajectoires de X_1(t)


Computational time : 59.45 seconds.
```

Avec les tenseurs, il n'est plus nécessaire d'employer une boucle (et les temps de calcul sont fortement réduits).

```
new;
library pgraph;

cls;

/*
** Modele GBM multidimensionnel
**
** dX_i(t) = \mu_i * X_i(t) * dt + \sigma_i * X_i(t) * dW_i(t), i = 1,...,n
**
** E[W_i(t)W_j(t)] = rho_{i,j}
**
**
*/
```

```
proc (1) = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = zeros(Nt,Nx);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(1,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
    x[i,.] = x0;

    i = i + 1;
  endo;

  retp(x);
endp;

proc (1) = array_simulate_mGBM(x0,mu,sigma,rho,t,Ns);
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = arrayinit(Nt|Ns|Nx,0);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
```

```
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(ns,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
    x[i,.,.] = x0;

    i = i + 1;
  endo;

  retp(x);
endp;

let x0 = 100 50 75;
let mu = 0.05 0.07 0.08;
let sigma = 0.15 0.25 0.05;

let rho = {1,
           0.5, 1,
           0.25, 0.15, 1};

rho = xpnd(rho);  // transforme le vecteur 'vech(rho)' en matrice de correlation

dt = 365;
t = seqa(0,1/dt,5*dt+1);

/*
** Verifions tout d'abord que les procedures array_simulate_GBM
** et matrix_simulate_GBM donnent les memes resultats lorsque
** le nombre de trajectoires est fixe a 1.
*/

rndseed 123;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1);
x = getmatrix(atranspose(x,2|1|3),1);

rndseed 123;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);

err = maxc(maxc(abs(x-y)));
print ftos(err,"Maximal absolute error = %lE",10,5);


/*
** Comparons maintenant les temps de calcul
*/

print;
print "Computational time for matrix_simulate_mGBM";
```

```
t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for one simulation = %lf seconds",5,2);

t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for two simulations = %lf seconds",5,2);

t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for three simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,10,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
print ftos((hsec-t0)/100,"for 10 simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,100,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
print ftos((hsec-t0)/100,"for 100 simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,1000,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
print ftos((hsec-t0)/100,"for 1000 simulations = %lf seconds",5,2);

print;
print "Computational time for array_simulate_mGBM";

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1);
print ftos((hsec-t0)/100,"for one simulation = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,2);
print ftos((hsec-t0)/100,"for two simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,3);
print ftos((hsec-t0)/100,"for three simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,10);
print ftos((hsec-t0)/100,"for 10 simulations = %lf seconds",5,2);
```

```
t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,100);
print ftos((hsec-t0)/100,"for 100 simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1000);
print ftos((hsec-t0)/100,"for 1000 simulations = %lf seconds",5,2);


Ns = 100;

x = array_simulate_mGBM(x0,mu,sigma,rho,t,Ns);

x = atranspose(x,3|1|2);
x1 = getmatrix(x,1);              /* Matrice Nt * 100 : la colonne i correspond a la i-ieme trajectoire
x2 = getmatrix(x,2);
x3 = getmatrix(x,3);


Computational time for matrix_simulate_mGBM
for one simulation =  0.06 seconds
for two simulations =  0.11 seconds
for three simulations =  0.17 seconds
for 10 simulations =  0.59 seconds
for 100 simulations =  5.78 seconds
for 1000 simulations = 57.86 seconds

Computational time for array_simulate_mGBM
for one simulation =  0.08 seconds
for two simulations =  0.06 seconds
for three simulations =  0.08 seconds
for 10 simulations =  0.11 seconds
for 100 simulations =  0.44 seconds
for 1000 simulations =  3.80 seconds
```

10 Les fonctions Threadsafe

```
new;

#include optim.sdf;                    //  #include pv.sdf
                                       //  #include ds.sdf

Nobs = 1000;
Np = 500;

param = rndu(Np,1);
data = rndn(Nobs,Np);

fn fct1(p) = p[1]*exp(-data*p);
t0 = hsec;
```

gradMT	Computes numerical gradient.
gradMTm	Computes numerical gradient with mask.
hessMT	Computes numerical Hessian.
hessMTg	Computes numerical Hessian using gradient procedure.
hessMTgw	Computes numerical Hessian using gradient procedure with weights.
hessMTm	Computes numerical Hessian with mask.
hessMTmw	Computes numerical Hessian with mask and weights.
hessMTw	Computes numerical Hessian with weights.
sqpSolveMT	Solve the nonlinear programming problem.
sqpSolveMTcontrolCreate	Creates an instance of a structure of type sqpSolveMTcontrol set to default values.
sqpSolveMTlagrangeCreate	Creates an instance of a structure of type sqpSolveMTlagrange set to default values.
sqpSolveMToutCreate	Creates an instance of a structure of type sqpSolveMTout set to default values.

TAB. 3 – Liste des functions Multi Thread

```
g1 = gradp(&fct1,param);
print /flush ftos((hsec-t0)/100,"Computational time : %lf sec.",5,3);

struct PV param_MT;
param_MT = pvCreate;
param_MT = pvPack(param_MT,param,"param");

struct DS data_MT;
data_MT = dsCreate;
data_MT.dataMatrix = data;
data_MT.dataMatrix = 0;

proc fct2(struct PV p, struct DS d);
    local param,y;
    param = pvUnpack(p,"param");
    y = param[1]*exp(-data*param);  // y = param[1]*exp(-d.dataMatrix*param);
    retp(y);
endp;

t0 = hsec;
g2 = gradMT(&fct2,param_MT,data_MT);
print /flush ftos((hsec-t0)/100,"Computational time : %lf sec.",5,3);
```

11 Les bibliothèques externes

C'est un aspect relativement peu exploité par les programmeurs. Pourtant, il permet de réaliser un certain nombre de taches qui

1. prennent trop de temps en GAUSS (par exemple, les procédures qui font appel à de nombreuses boucles) ;
2. ou qui ne sont pas accessibles à partir des commandes de GAUSS (par exemple, l'accès à *user.dll*, *kernel.dll* ou *mm.dll*).

Le lecteur peut consulter le séminaire de Londres pour un traitement plus complet de ce sujet :

http ://www.city.ac.uk/cubs/ferc/thierry/conf3pdf.zip

http ://www.city.ac.uk/cubs/ferc/thierry/gdll1.html

L'exemple qui suit utilise le compilateur gratuit LCC-Win32 disponible sur http ://www.cs.virginia.edu/~lcc-win32/.

Vous pouvez télécharger la bibliothèque `MVT` à l'adresse suivante :

`http ://gro.creditlyonnais.fr/content/wp/mvt.zip`

Cette bibliothèque permet de calculer les cdf de distributions t multivariées :

> **MVT** is a GAUSS library for computing multivariate t cdf. It is based on the Fortran packages mvt.f and mvtdstpack.f written by Alan Genz. The Fortran source code is available from his web page :
>
> `www.sci.wsu.edu/math/faculty/genz/homepage`
>
> The Fortran packages contain different subroutines to compute t-probabilities. The algorithms of these subroutines are described in the following articles :
>
> Genz, A. and F. Bretz [1999], Numerical computation of the multivariate t-probabilities with applications to power calculation of multiple contrasts, *Journal of Statistical Computation and Simulation*, **63(4)**, 361-378 (available from `www.sci.wsu.edu/math/faculty/genz/homepage`)
>
> Genz, A. and F. Bretz [2000], Comparison of methods for the computation of multivariate t-probabilities, submitted (available from `www.sci.wsu.edu/math/faculty/genz/homepage`)

La bibliothéque `MVT` a été construite de la faon suivante.

1. Le code fortran a été transformé en C avec le programme *f2c*.
2. Pour chaque fonction que nous voulons utiliser avec **GAUSS**, nous avons construit une fonction d'emballage (*wrapper function*) dont les paramètres sont de type `double *`.
3. Pour chaque fichier DLL, nous rajoutons un point d'entrée (*entry point*).
4. Enfin, nous créons une procédure GAUSS, plus conviviale à utiliser que la fonction DLL.

Prenons par exemple le cas de la distribution t bivariée. La fonction `C` correspondante est `mvbvt_`. La fonction d'emballage est `dllmvbvt`.

```
__declspec(dllexport) int dllmvbvt(double *_m, double *_nu, double *_lower_x, double *_upper_x,
                                   double *_lower_y, double *_upper_y, double *_infin_x,
                                   double *_infin_y, double *_correl, double *_value)
{
    int m, nu, *infin;
    double *lower, *upper, *correl, value;
    int j;

    /* cast */
    m        = (int) *_m;
    nu       = (int) *_nu;

    lower = (double*) malloc(2*sizeof(double));
    upper = (double*) malloc(2*sizeof(double));
    infin = (int*) malloc(2*sizeof(int));
    correl = (double*) malloc(sizeof(double));

    for (j=0;j<m;j++)
```

```
    {
        lower[0] = (double) _lower_x[j];
        lower[1] = (double) _lower_y[j];
        upper[0] = (double) _upper_x[j];
        upper[1] = (double) _upper_y[j];
        infin[0] = (int) _infin_x[j];
        infin[1] = (int) _infin_y[j];
        correl[0] = (double) _correl[j];

        value = mvbvt_(&nu, lower, upper, infin, _correl);

        _value[j] = value;

    }

    free(lower);
    free(upper);
    free(infin);
    free(correl);

    return(0);
}
```

La procédure **GAUSS** correspondante est `cdfbvt`.

```
proc (1) = cdfbvt(x,y,rho,nu);
  retp(_cdfbvt(__INFn,x,__INFn,y,rho,nu));
endp;

proc (1) = _cdfbvt(xMin,xMax,yMin,yMax,rho,nu);
  local N,M,NM;
  local e,infin_x,infin_y;
  local p;

  N = maxc(cols(xMin)|cols(xMax)|cols(yMin)|cols(yMax)|cols(rho));
  M = maxc(rows(xMin)|rows(xMax)|rows(yMin)|rows(yMax)|rows(rho));


  infin_x = (-1) .* (xMin .== __INFn .and xMax .== __INFp) +
             0  .* (xMin .== __INFn .and xMax ./= __INFp) +
             1  .* (xMin ./= __INFn .and xMax .== __INFp) +
             2  .* (xMin ./= __INFn .and xMax ./= __INFp);

  infin_y = (-1) .* (yMin .== __INFn .and yMax .== __INFp) +
             0  .* (yMin .== __INFn .and yMax ./= __INFp) +
             1  .* (yMin ./= __INFn .and yMax .== __INFp) +
             2  .* (yMin ./= __INFn .and yMax ./= __INFp);

  e = ones(M,N);
  xMin = vecr(xMin .* e);
  xMax = vecr(xMax .* e);
  yMin = vecr(yMin .* e);
  yMax = vecr(yMax .* e);
  infin_x = vecr(infin_x .* e);
```

```
  infin_y = vecr(infin_y .* e);
  rho = vecr(rho .* e);

  NM = N*M;
  p = zeros(M,N);

  dllcall _dllmvbvt(NM,nu,xMin,xMax,yMin,yMax,infin_x,infin_y,rho,p);

  retp(p);
endp;
```

Voici un exemple d'utilisation de la procédure `cdfbvt` :

```
new;
library mvt;

mvtset;

x = rndn(10,2);
let rho[2,2] = 1 0.25 0.25 1;
nu = 2;

cdf0 = cdfbvt(x[.,1],x[.,2],rho[1,2],nu);

nu = 100000; /* Asymtotic case --> BVN */

cdf1 = cdfbvt(x[.,1],x[.,2],rho[1,2],nu);
cdf2 = cdfbvn(x[.,1],x[.,2],rho[1,2]);

print "    var. 1    var. 2    cdfbvt    cdfbvn     diff.";
print "=================================================";
call printfm(x~cdf1~cdf2~(cdf1-cdf2),1,"%lf"~10~3);
```

Remarque 2 *Avec la nouvelle version de* ***GAUSS****, nous pouvons optimiser l'accès aux DLL en employant les commandes d'initialisation* `matalloc` *et* `matinit`*.*

Troisième partie

Présentation d'études réalisées au Groupe de recherche Opérationnelle du Crédit Lyonnais

Ces études issues de la veille technologique du GRO sont disponibles sur le site web :

`http ://gro.creditlyonnais.fr`

Le lien direct pour la page **GAUSS** du GRO est :

`http ://gro.creditlyonnais.fr/content/rd/home_gauss.htm`