Econométrie Financière

Exercices de Programmation

Thierry Roncalli

Février 2006

Table des matières

Première partie

Première Séance : Gestion de portefeuille, méthode généralisée des moments et analyse spectrale

1 Gestion de portefeuille

1.1 Allocation de portefeuille

1.1.1 Optimisation risque/rendement

```
new;

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

sharpe = mu ./ sigma;

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

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

cov = sigma .* sigma' .* rho;

Q = 2*cov; R = mu;

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

sv = ones(n,1)/n;

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

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

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

1.1.2 Construction de la frontière efficiente

```
new; library pgraph;

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

sharpe = mu ./ sigma;

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

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

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

Q = 2*cov; R = mu;

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

sv = ones(n,1)/n;

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

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

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

1.1.3 Détermination du portefeuille optimal

```
new; library pgraph;

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

sharpe = mu ./ sigma;

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

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

cov = sigma .* sigma' .* rho;

Q = 2*cov; R = mu;

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

sv = ones(n,1)/n;

Target_Vol = 1/100;

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

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

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

```
  local vol;

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

  retp(vol-Target_Vol);
endp;

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

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

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

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

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

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

  c = (a+b)/2;

  retp(c);
endp;
```

1.2 Calcul du beta

```
new; library pgraph;

cls;
```

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

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

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

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

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

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

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

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

print "R2 = " R2;

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

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

1.3 Détermination de l'alpha et régression de style

```
new;
library pgraph;

cls;

loadm VL_FUND;
loadm VL_BENCHMARK;

loads NAME_FUND;
loads NAME_BENCHMARK;

/*
**> Test des dates
*/

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

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

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

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

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

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

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

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

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

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

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

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

  indx = indx + 1;
endo;
```

2 Méthode généralisée des moments

2.1 La procédure regGMM

```
/*
**> regGMM
**
** Objet : Méthode des moments généralisés.
**
** Format :  {theta,stderr,Mcov,Qmin} = regGMM(&h,sv,RR,r);
**
** Entrées :    &fct - scalaire, pointeur de la fonction des moments
**                sv - vecteur G*1, valeurs initiales pour l'algorithme d'optimisation
**                RR - matrice K*G, matrice $R$ de la restriction implicite $\beta = R\gamma + r$
**                 r - vecteur K*1, vecteur $r$ de la restriction implicite $\beta = R\gamma + r$
**
** Sorties : theta - vecteur K*1, vecteur des paramètres estimés
**          stderr - vecteur K*1, vecteur des écart-types des paramètres estimés
**            Mcov - matrice K*K, matrice de covariance des paramètres estimés
**            Qmin - scalaire, valeur de la fonction objective $Q\left(\theta\right)$
**
** Globales :
**          _ParNames - vecteur K*1, vecteur des noms de paramètres (défaut = 0)
**       _regGMM_invW - matrice m*m, valeur de la matrice des poids
**                      -- ou --
**                      0 pour utiliser la matrice de covariance de Newey et West (défaut = 0)
**      _regGMM_iters - scalaire, nombre d'itérations (défaut = 20)
**       _regGMM_lags - scalaire, nombre de retards pour la fenêtre de Bartlett (défaut = 0)
**        _regGMM_tol - scalaire, tolérance de la convergence (défaut = 0.001)
**       _regGMM_pinv - scalaire, 1 pour autoriser l'utilisation de l'inverse de Moore-Penrose
**                      (défaut = 0)
**        _regGMM_mtd - scalaire, type de méthode pour construire les poids (défaut = 1)
**                      1 pour prendre en compte les corrélations entre les moments
**                      2 pour ne pas les prendre en compte
**          _reg_cov - scalaire, méthode d'estimation de la matrice de covariance (défaut = 1)
**                      0 pour ne pas la calculer
**                      1, 2 ou 3 pour l'estimateur OPG
**        _reg_opalgr - scalaire, méthode d'optimisation (défaut = 2)
**                      1 pour la méthode SD (steepest descent)
**                      2 pour la méthode BFGS (Broyden, Fletcher, Goldfarb, Shanno)
**                      3 pour la méthode Scaled BFGS
**                      4 pour la méthode Self-Scaling DFP (Davidon, Fletcher, Powell)
**                      5 pour la méthode NEWTON (Newton-Raphson)
**                      6 pour la méthode Polak-Ribiere Conjugate Gradient
**    _reg_PrintIters - scalaire
**                      0 pour ne pas afficher les itérations de l'optimisation
**                      1 pour afficher les itérations de l'optimisation dans la fenêtre GAUSS
**                      2 pour afficher les itérations de l'optimisation en mode DOS Window
**        _reg_Output - chaîne de caractères, nom du fichier pour la sauvegarde des itérations
**           _reg_row - scalaire, nombre d'observations utilisé pour la lecture séquentielle
**                      de la base de données (défaut = 0)
**                      0 pour déterminer un nombre optimal (utilisation de la procédure getnr)
**           __output - 1 pour l'affichage des résultats (défaut = 1)
**            __title - chaîne de caractères, nom du modèle
**
**            _reg_df - scalaire, nombre de degrés de liberté
**      _regGMM_Jtest - vecteur 2*1, valeur et p-value du test J de Hansen
```

```
**
*/
```

2.2 MCO et variables instrumentales

```
new; library optmum,pgraph;

#include regGMM.src

Nobs = 500; x = 10*rndu(Nobs,4); beta = rndn(4,1); sigma = 2; y =
x*beta + rndn(Nobs,1)*sigma;

sv = beta|sigma; /* starting values */

proc H(theta);
  local beta,sigma,h1,M,i;

  M = zeros(Nobs,6);

  beta = theta[1:4];
  sigma = theta[5];

  /* first moment */
  h1 = y - x*beta;
  M[.,1] = h1;
  /* second moment */
  M[.,2] = h1.*h1 - sigma^2;

  i = 1;
  do until i > 4;
    M[.,2+i] = h1.*x[.,i];
    i = i + 1;
  endo;

  retp(M);
endp;

{theta,stderr,Mcov,Qmin} = regGMM(&h,sv,0,0);

ols = y/x; u = y-x*ols; ols = ols | stdc(u);

print sv~theta~ols;
```

2.3 Les processus ARCH

```
new; library optmum,pgraph;

#include regGMM.src

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

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

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

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

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

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

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

  retp(logl);
endp;

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

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

  M = zeros(Nobs,4);

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

  /* first moment */

  M[.,1] = u;

  /* computing h(t)^2 */

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

  /* second moment */

  M[.,2] = u2-h2;

  /* u(t) uncorrelated with x(t) */

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

  /* u(t)^2-h(t)^2 uncorrelated with u(t-i)^2, i = 1 */

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

  retp(M);
endp;

sv = 2|3|0.5|0.6;

{theta1,fmin,grd,retcode} = optmum(&negLogL,sv);

{theta2,stderr,Mcov,Qmin} = regGMM(&arch,sv,0,0);
```

```
parnm = "beta1"|"beta2"|"alpha0"|"alpha1"; print; print; print "
True values        TDML            GMM"; call
printfmt(parnm~sv~theta1~theta2,0~1~1~1);
```

2.4 Les processus de diffusion

2.4.1 Le mouvement brownien géométrique

```
new; library optmum,pgraph;

#include regGMM.src

X0 = 10; mu = 0.2; sigma = 0.5; h = 0.1; Nobs = 500; sv = mu|sigma;

/* Generate a geometric Brownian motion */

t = seqa(0,h,Nobs); xt = x0*exp(   (mu-0.5*sigma^2)*t
               + sigma*cumsumc(0|rndn(Nobs-1,1)*sqrt(h))
          );

proc ml(theta);
  local e,mu,sigma,epsilon,logl;

  e = ln(xt[2:Nobs]./xt[1:Nobs-1]);

  mu = theta[1];
  sigma = theta[2];

  epsilon = e - (mu-0.5*sigma^2)*h;

  logl = -0.5*ln(2*pi) - 0.5*ln(sigma^2*h) - 0.5*epsilon^2/(sigma^2*h);

  retp(logl);
endp;

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

proc gmm(theta);
  local e,mu,sigma,epsilon,M;

  e = ln(xt[2:Nobs]./xt[1:Nobs-1]);

  mu = theta[1];
  sigma = theta[2];

  epsilon = e - (mu-0.5*sigma^2)*h;

  M = epsilon~(epsilon^2-(sigma^2*h));

  retp(M);
endp;

parnm = "mu"|"sigma";

{theta1,fmin,grd,retcode} = optmum(&negLogL,sv);
```

```
{theta2,stderr,Mcov,Qmin} = regGMM(&gmm,sv,0,0);

print; print; print "                  True values        TDML
GMM"; call printfmt(parnm~sv~theta1~theta2,0~1~1~1);
```

2.4.2 Le processus d'Ornstein-Uhlenbeck

```
new;
library optmum,pgraph;

#include regGMM.src

X0 = 10;
a = 0.8;
b = 0.1;
sigma = 0.06;
h = 0.1;
Nobs = 1000;
sv = a|b|sigma;

/* Generate an Ornstein-Uhlenbeck process */

xt = zeros(Nobs,1);
xt[1] = x0;

k1 = exp(-a*h);
k2 = (1-exp(-2*a*h))/(2*a);

u = (sigma*sqrt(k2))*rndn(Nobs-1,1);
u = u+b*(1-k1);

i = 2;
do until i > Nobs;
  xt[i] = k1*xt[i-1]+u[i-1];
  i = i + 1;
endo;

proc ml(theta);
  local a,b,sigma,k1,k2,epsilon,logl;

  a = theta[1];
  b = theta[2];
  sigma = theta[3];

  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);

  epsilon = xt[2:Nobs]-k1.*xt[1:Nobs-1]-b*(1-k1);

  logl = -0.5*ln(2*pi) - 0.5*ln(k2) - 0.5*epsilon^2/k2;

  retp(logl);
endp;

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

proc gmm(theta);
```

```
  local a,b,sigma,k1,k2,epsilon,M;

  a = theta[1];
  b = theta[2];
  sigma = theta[3];

  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);

  epsilon = xt[2:Nobs]-k1.*xt[1:Nobs-1]-b*(1-k1);

  M = epsilon~(epsilon^2-k2)~(epsilon.*xt[1:Nobs-1]);

  retp(M);
endp;

parnm = "a"|"b"|"sigma";

{theta1,fmin,grd,retcode} = optmum(&negLogL,sv);

{theta2,stderr,Mcov,Qmin} = regGMM(&gmm,sv,0,0);

print; print;
print "                  True values        TDML            GMM";
call printfmt(parnm~sv~theta1~theta2,0~1~1~1);
```

2.5 Méthode des moments simulés : l'exemple du processus de volatilité stochastique de Heston

```
/*
**> rndHeston
*/

proc (2) = rndHeston(S0,V0,muS,kappa,muV,epsilon,rho,t,Ns,cn);
  local Nt,St,Vt,dt,sqrt_dt,eSt,eVt,Sigma0;

  Nt = rows(t);

  if cn == 1;
    St = zeros(Ns,Nt);
    Vt = zeros(Ns,Nt);
  endif;

  for i(1,Nt,1);
    if i == 1;
      dt = t[i] - 0.0;
    else;
      dt = t[i] - t[i-1];
    endif;
    sqrt_dt = sqrt(dt);

    eSt = rndn(Ns,1);
    eVt = rho .* eSt + sqrt(1-rho^2) .* rndn(Ns,1);

    V0 = substute(V0,V0 .<= 0, __macheps);
    Sigma0 = sqrt(V0);
```

```
    S0 = S0 + muS .* dt .* S0 + Sigma0 .* S0 .* sqrt_dt .* eSt;
    V0 = real(V0 + kappa .* (muV - V0) .* dt + epsilon .* Sigma0 .* sqrt_dt .* eVt);
    if cn == 1;
      St[.,i] = S0;
      Vt[.,i] = V0;
    endif;
  endfor;

  if cn == 1;
    retp(St',Vt');
  else;
    retp(S0,V0);
  endif;
endp;

proc _SimulatedMoments_Heston(theta);
  local V0,muS,muV,kappa,epsilon,rho;
  local St,Vt,Rt,SM,h;

  {V0,muS,muV,kappa,epsilon,rho} = _GMMTheta_To_HestonParams(theta);

  rndseed _heston_seed;
  {St,Vt} = rndHeston(100,V0,muS,kappa,muV,epsilon,rho,seqa(0,_heston_dt,_heston_Nt),_heston_Ns,1);
  Rt = (trimr(ln(St) - lag1(ln(St)),1,0));

  SM = zeros(_heston_Ns,4);
  h = meanc(Rt);
  SM[.,1] = h;
  h = Rt - h';
  SM[.,2] = meanc(h .* h);
  SM[.,3] = meanc(h .* h .* h);
  SM[.,4] = meanc(h .* h .* h .* h);

  SM = meanc(SM);

  retp(SM);
endp;

proc _Heston_GMM(theta);
  local SM,H;

  SM = _SimulatedMoments_Heston(theta);
  H = zeros(rows(_heston_x),4);
  H[.,1] = _heston_x - SM[1];
  H[.,2] = H[.,1]^2 - SM[2];
  H[.,3] = H[.,1]^3 - SM[3];
  H[.,4] = H[.,1]^4 - SM[4];
  H = H ./ (_Heston_dt^(1~1~1.5~2));

  retp(real(H));
endp;
```

3 Analyse spectrale

```
proc (1) = fourier(x);
  local d;
```

```
  d = dfft(x); // d = fft(x);
  d = rows(d)*d;
  retp(d);
endp;

proc (1) = inverse_fourier(d);
  local x;
  x = dffti(d); // x = ffti(d);
  x = x/rows(x);
  retp(x);
endp;

proc (2) = PDGM(x);
  local N,d,T,I,lambda;
  x = packr(x);
  N = rows(x);
  d = fourier(x);
  T = rows(d);
  I = abs(d)^2; I= I/N;
  lambda = 2*pi*seqa(0,1,T)/T;
  retp(lambda,I);
endp;

proc (2) = fourier2(x,y);
  local z,d,N,dx,dy;
  local c1,c2,d_;

  z = complex(x,y);
  d = fourier(z);

  N = rows(d);
  dx = zeros(N,1); dy = dx;

  dx[1] = real(d[1]);
  dy[1] = imag(d[1]);

  c1 = complex(0.5,0); c2 = complex(0,-0.5);

  d = trimr(d,1,0);
  d_ = rev(d);
  d_ = conj(d_);

  dx[2:N] = c1*(d+d_);
  dy[2:N] = c2*(d-d_);

  retp(dx,dy);
endp;

proc (2) = fourier3(x);
  local N,K,D,i,x1,x2,d1,d2,Nstar,lambda;

  x = packr(x);
  N = rows(x);
  K = cols(x);
  D = {};
```

```
  if iscplx(x);

    i = 1;
    do until i > K;
      x1 = x[.,i];
      d1 = fourier(x1);
      D = D~d1;
      i = i + 1;
    endo;

  else;

    i = 1;
    do until i > K-1;
      x1 = x[.,i];
      x2 = x[.,i+1];
      {d1,d2} = fourier2(x1,x2);
      D = d~d1~d2;
      i = i+2;
    endo;

    if i == K;
      x1 = x[.,K];
      d1 = fourier(x1);
      D = D~d1;
    endif;

  endif;

  Nstar = rows(D);
  lambda = 2*pi*seqa(0,1,Nstar)/Nstar;

  retp(lambda,D);
endp;

proc (2) = PDGM2(x);
  local N,K,lambda,D,Nstar,Ix,j,Dj;

  x = packr(x);
  N = rows(x);
  K = cols(x);

  {lambda,D} = fourier3(x);
  Nstar = rows(D);

  Ix = zeros(Nstar,K^2);

  j = 1;
  do until j > Nstar;
    Dj = D[j,.];
    Dj = Dj.';
    Ix[j,.] = vec(Dj*Dj').';
    j = j + 1;
  endo;

  Ix = Ix/N;
```

```
  retp(lambda,Ix);
endp;

proc (2) = CPDGM(x,y);
  local data,N,dx,dy,T,Ixy,lambda;
  data = packr(x~y); x = data[.,1]; y = data[.,2];
  N = rows(x);
  dx = fourier(x); dy = fourier(y); T = rows(dx);
  Ixy = dx.*conj(dy); Ixy = Ixy/N;
  lambda = 2*pi*seqa(0,1,T)/T;
  retp(lambda,Ixy);
endp;

proc (3) = CSpectrum(Ix,Iy,Ixy);
  local N;
  local w,alpha,phi;

  N = rows(Ix);
  w = Ixy./sqrt(Ix.*Iy);
  {alpha,phi} = topolar(w);

  retp(w,alpha,phi);
endp;
```

3.1 Périodogramme et représentation spectrale

```
/*
**  TONG [1990], Non-linear Time Series, Oxford University Press
**
**  PRIESTLEY [1994], Spectral Analysis and Times Series,
**  Academic Press, London
**
**  Lynx data (1821-1934)
**  Tong, page 470
*/

new;
library pgraph;

#include spectral.src

load lynx[] = lynx.asc;

cls;

y = log(lynx);
Nobs = rows(y);

t = seqa(1821,1,1934-1821+1);

pqgwin many;

graphset;                              /* Priestley, page 384 */
   title("Lynx data");
  _pdate = ""; _pnum = 2;
```

```
  xlabel("date");
  xy(t,y);

{lambda,I} = PDGM(y);

I[1] = miss(0,0);

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

graphset;                              /* Priestley, page 412 */
  _pdate = ""; _pnum = 2; _pframe = 0;
  fonts("simplex simgrma");
  title("\216Periodogram of the Lynx data");
  xtics(0,pi,pi/4,0);
  lab = " 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201";
  asclabel(lab,0);
  xlabel("\216frequency");
  xy(lambda[1:q],I[1:q]);

/*
**  Fisher's g-statistic
**
**  Priestley, page 407, formula 6.1.66
*/

g = maxc(I[2:57])/sumc(I[2:57]);
gstar = 2*Nobs*g;                      /* Priestley, page 407, formula 6.1.64 */
pvalue = Nobs*exp(-gstar/2);           /* Priestley, page 407, formula 6.1.65 */

print "Test for Periodogram Ordinates";
print chrs(45*ones(60,1)); print;
s = maxindc(I[2:57]);
print ftos(lambda[s],"Test for periodic component at the frequency %lf",4,2);
print;
print ftos(g,     "Fisher's g-statistic : %lf",6,5);
print ftos(gstar, "g* :                   %lf",6,5);
print ftos(pvalue,"p-value :              %lf",6,5);
```

3.2 Covariogramme déduit d'une densité spectrale

```
new;
library pgraph;

#include spectral.src

Nobs = 500;
s = seqa(1,1,Nobs);

let SIGMA = {1 0.5,0.5 1};
let Phi = {0.6 0.3,0 0.5};

x = zeros(2,Nobs);
P = chol(Sigma)';
t = 2;
do until t > Nobs;
  x[.,t] = Phi*x[.,t-1] + P*rndn(2,1);
```

```
  t = t + 1;
endo;

x = x';
x = x - meanc(x)';

x1 = x[.,1];
x2 = x[.,2];

{lambda,I} = PDGM(x1);
cov = real(inverse_fourier(I));

R1 = autoc(x1,10);

pqgwin many;

graphset;
    _pdate = "";
    title("Covariogram");
    _pltype = 6|1;
    xlabel("Lag");
    xy(seqa(0,1,11),R1~cov[1:11]);


proc autoc(x,k);
  local t,rho;
  x=packr(x); t=rows(x); x=x-meanc(x);
  rho=rev(conv(x,rev(x),t-k,t));
  retp(rho/t);
endp;
```

3.3 Covariogramme croisé et densité spectrale multidimensionnelle

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

rndseed 123456;

_fourier = 1;

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

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

{lambda,I} = PDGM2(y);
g = sgf_SSM(lambda);

/* first component */

CV1a = real(inverse_fourier(I[.,1]));
CV1b = real(inverse_fourier(g[.,1]));
```

```
/* second component */

CV2a = real(inverse_fourier(I[.,4]));
CV2b = real(inverse_fourier(g[.,4]));

/* first component/second component */

CV3a = real(inverse_fourier(I[.,3]));
CV3b = real(inverse_fourier(g[.,3]));

/* second component/first component */

CV4a = real(inverse_fourier(I[.,2]));
CV4b = real(inverse_fourier(g[.,2]));

pqgwin many;


graphset;
    _pdate = "";
    title("Covariogram of the first component of the multivariate process");
    _pltype = 6|1;
    _plegstr = "estimated\0theoretical";
    _plegctl = {2 6 5 3};
    xlabel("Lag");
    xy(seqa(0,1,11),CV1a[1:11]~CV1b[1:11]);

graphset;
    _pdate = "";
    title("Covariogram of the second component of the multivariate process");
    _pltype = 6|1;
    _plegstr = "estimated\0theoretical";
    _plegctl = {2 6 5 1.25};
    xlabel("Lag");
    xy(seqa(0,1,11),CV2a[1:11]~CV2b[1:11]);

graphset;
    _pdate = "";
    title("Cross-covariogram of the multivariate process"\
          "\Lcov(y]1[(t),y]2[(t-lag))");
    _pltype = 6|1;
    _plegstr = "estimated\0theoretical";
    _plegctl = {2 6 5 3.25};
    xlabel("Lag");
    xy(seqa(0,1,11),CV3a[1:11]~CV3b[1:11]);


graphset;
    _pdate = "";
    title("Cross-covariogram of the multivariate process"\
          "\Lcov(y]2[(t),y]1[(t-lag))");
    _pltype = 6|1;
    _plegstr = "estimated\0theoretical";
    _plegctl = {2 6 5 4};
    xlabel("Lag");
```

```
    xy(seqa(0,1,11),CV4a[1:11]~CV4b[1:11]);
```

3.4 Estimation dans le domaine des fréquences : la méthode du maximum de vraisemblance de Whittle

3.4.1 le cas du bruit blanc

```
new;

#include spectral.src

sigma = 1; x = rndn(1000,1)*sigma;

{lambda,I} = PDGM(x);

proc mlfn(theta);
  local sigma,g,logl;

  sigma = sqrt(theta^2);
  g = sigma^2;
  logl = -0.5*ln(2*pi) - 0.5*ln(g) - 0.5*(I./g);

  retp(logl);
endp;

fn NegLoglik(theta) = -sumc(mlfn(theta));

{theta,fmin,grd,retcode} = optmum(&NegLogLik,0.5); sigma =
sqrt(theta^2);

cls;

print "Estimated FDML sigma = " sigma; print "Empirical std dev = "
stdc(x);
```

3.4.2 la densité spectrale exponentielle de Bloomfield

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

declare external r;

rndseed 123;
y = recserar(rndn(500,1),0|0,0.5|0.4);

_tsm_Mcov = 1;

r = 4;
sv = ones(r+1,1);
_tsm_parnm = "gamma1"|"gamma2"|"gamma3"|"gamma4"|"sigma";

{theta,stderr,Mcov,Logl} = FD_ml(y,&sgf,sv);

{lambda,I} = PDGM(y);
_smoothing = 5|5|0|0.23;   /* Parzen lag window with bandwidth = 2 */
I = smoothing(I);
```

```
g = sgf(theta,lambda);
q = trunc(rows(lambda)/2);

graphset;
  _pdate = ""; _pnum = 2; fonts("simplex simgrma");
  xlabel("frequency");
  _plegstr = "Periodogram of the data"\
          "\000Estimated spectral generating function";
  _plegctl = {2 5 3 4};
  xtics(0,pi,pi/4,0);
  lab = " 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201";
  asclabel(lab,0);
  xy(lambda[1:q],I[1:q]~g[1:q]);


proc sgf(theta,lambda);       /* Dzhaparidze, page 125 */
  local N,gamma_,sigma,j,w,g;
  N = rows(lambda);
  gamma_ = theta[1:r];
  sigma = theta[r+1];
  j = seqa(1,1,r);
  w = lambda.*j';
  w = cos(w);
  g = (sigma^2)*exp(2*w*gamma_);
  retp(g);
endp;
```

3.4.3 le cas du modèle AR(1) + MA(1)

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

rndseed 123456;

z = Process;

sv = 0.5|0.25|0.7|1;
_tsm_parnm = "phi1"|"sigma_u"|"theta1"|"sigma_v";

{coeff,stderr,Mcov,Logl} = FD_ml(z,&sgf,sv);

proc sgf(coeff,lambda);
  local phi1,theta1,sigma_u,sigma_v;
  local w,g;
  phi1 = coeff[1];
  sigma_u = coeff[2];
  theta1 = coeff[3];
  sigma_v = coeff[4];
  w = cos(lambda);
  g = (sigma_u^2)./(1-2*phi1*w+phi1^2) +
      (sigma_v^2).*(1-2*theta1*w+theta1^2);
  retp(g);
endp;

proc process;
  local Nobs,u,x,v,y,z;
```

```
  Nobs = 1000;
  u = rndn(Nobs,1)*0.25;
  x = recserar(u,0,0.50);
  v = rndn(Nobs,1)*1;
  y = v - 0.7*(0|trimr(v,0,1));
  z = x + y;
  retp(z);
endp;
```

3.4.4 le cas d'un ARMA non stationnaire bruité

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

rndseed 123;

sv = 0.5|1|0.5;
sv = 1.5|0.5|2;

Nobs = 500;

_tsm_parnm = "theta1"|"sigma_u"|"sigma_v";
_tsm_Mcov = 1;

z = Process;

{coeff,stderr,Mcov,Logl} = FD_ml(z-lag1(z),&sgf,sv);

proc sgf(coeff,lambda);
  local phi1,theta1,sigma_u,sigma_v;
  local w,g;
  theta1 = coeff[1];
  sigma_u = coeff[2];
  sigma_v = coeff[3];
  w = cos(lambda);
  g = (sigma_u^2).*(1-2*theta1*w+theta1^2) +
      (sigma_v^2).*(2*(1-w));
  retp(g);
endp;

proc process;
  local u,x,v,y,z;
  u = rndn(Nobs,1)*1;
  x = recserar(u - 0.5*(0|trimr(u,0,1)),0,1);
  v = rndn(Nobs,1)*0.5;
  y = v;
  z = x + y;
  retp(z);
endp;
```

3.4.5 La densité spectrale d'un modèle espace-état

```
/*
**  State space model of an ARMA(1,1) process with noise
**
**  Suppose that
```

```
**     z(t) = y(t) + e(t)
**  and
**     y(t) = phi1*y(t-1) + u(t) -theta1*u(t-1)
**
**  e(t) is the noise (the measure error for example)
**  y(t) is the ARMA(1,1) process
**  z(t) is the ARMA(1,1) process with noise
**
**  The state space form is:
**
**                          |      |
**              |      | | y(t) |
**     z(t)  =  | 1  0 | |      |  + e(t)
**              |      | | u(t) |
**                          |      |
**
**    |      |    |               |  |        |    |   |
**    | y(t) |    | phi1  -theta1 |  | y(t-1) |    | 1 |
**    |      |  = |               |  |        | +  |   | u(t)
**    | u(t) |    |   0       0   |  | u(t-1) |    | 1 |
**    |      |    |               |  |        |    |   |
**
**  Maximum Likelihood in the frequency domain
**
*/

new;
library tsm,optmum,pgraph;
TSMset;

/*
**  Simulation of the process with
**
**  var[e(t)] = 0.25, phi1 = 0.95, theta1 = 0.5 and var[u(t)] = 1
**  y(0) = 40;
*/

rndseed 123456;

t_ = seqa(1,1,500);

u = rndn(500,1)*sqrt(1);     /* Simulate the u(t) process */
u_ = u~(0|trimr(u,0,1));
u_ = u_*(1|-0.5);
y = recserar(u_,40,0.95);   /* Simulate the y(t) process */
e = rndn(500,1)*sqrt(0.25);
zt = y + e;                 /* Simulate the z(t) process */


proc sgf1(beta,lambda);
  local phi1,theta1,sig_u,sig_e;
  local w,w1,w2,g;
  phi1 = beta[1];
  theta1 = beta[2];
  sig_u = beta[3];
  sig_e = beta[4];
```

```
  w = cos(lambda);
  w1 = 1 - 2*theta1*w + theta1^2;
  w2 = 1 - 2*phi1*w + phi1^2;
  g = (w1./w2)*sig_u^2 + sig_e^2;
  retp(g);
endp;


/*
** Procedure to compute the sgf of the SSM
*/

proc sgf2(beta,lambda);
  local phi1,theta1,sig_u,sig_e;
  local T,Q,H,Z,d,c,R;
  local G;

  phi1 = beta[1];
  theta1 = beta[2];
  sig_u = beta[3];
  sig_e = beta[4];
  T = (phi1~-theta1)|(0~0);
  Q = sig_u^2; H = sig_e^2;
  Z = {1 0}; d = 0;
  c = {0,0}; R = {1,1};

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

sv = 0.95|0.5|sqrt(1|0.25);

_tsm_Mcov = 0;
_tsm_optmum = 1;
_tsm_parnm = "phi1"|"theta1"|"sig_e"|"sig_u";

{theta1,stderr,Mcov,Logl} =  FD_ml(zt,&sgf1,sv);

{theta2,stderr,Mcov,Logl} =  FD_ml(zt,&sgf2,sv);

print "             FDML with sgf1   FDML with sgf2";
call printfmt(_tsm_parnm~theta1~theta2,0~1~1);
print;
print "The first estimation is much faster, because the second sgf is
computed within a loop.";
```

3.5 Modélisation de la volatilité implicite ATM de change

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

#include FXO.src
```

```
let string Currency = "EUR/JPY" "EUR/USD" "GBP/EUR" "GBP/USD" "JPY/USD";
let string varNames = "12M ATM";

{dt,vol} = FXO_LoadData1M(Currency,varNames);
vol = selif(vol,trunc(dt/100) .>= 1998);
dt = selif(dt,trunc(dt/100) .>= 1998);

__output = 0;
_tsm_Mcov = 2;

s = 12;
SeasonalComponent = zeros(12,5);

i = 1;
do until i > 5;

  y = vol[.,i];
  let sv = 0.01 0.2 0.5;
  {theta,stderr,Mcov,Logl} = BTSM_cml(y,s,sv,0,0);
  call SSM_build(SSM_BTSM(s,theta),0);

  a0 = y[1]|zeros(s-1,1);
  P0 = zeros(s,s);

  iter = 1;
  do until iter > 5;
    call KFiltering(y,a0,P0);
    a = KF_matrix(3);
    ssc = a[.,2];
//  asc = meanc(reshape(ssc,8,12));
    asc = FXO_Meanc(ssc);
    a0 = y[1]|rev(asc[2:12]);
    iter = iter + 1;
  endo;

  ssc1 = ssc;
  asc1 = asc;
  volatility = theta;

  proc mlProc(theta);
    local beta,asc,a0,Logl;
    beta = volatility;
    asc = theta;
    a0 = y[1]|asc;
    call SSM_build(SSM_BTSM(s,beta),0);
    call KFiltering(y,a0,P0);
    Logl = KF_ML;
    retp(Logl);
  endp;

  sv = rev(asc1[2:12]);

  __output = 1;
  _reg_PrintIters = 0;
  {theta,stderr,cov,logl} = TD_ML(&mlProc,sv);
```

```
  beta = volatility;
  asc = theta;
  a0 = y[1]|asc;
  call SSM_build(SSM_BTSM(s,beta),0);
  call KFiltering(y,a0,P0);
  a = KF_matrix(3);
  ssc = a[.,2];
// asc = meanc(reshape(ssc,8,12));
  asc = FXO_Meanc(ssc);

  SeasonalComponent[.,i] = FXO_Meanc(submat(KF_matrix(3),0,2));

  i = i + 1;
endo;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|3|1; _plwidth = 10; _pcross = 1;
  _plctrl = 1; _pstype = 8|9|10|11|12; _pnumht = 0.18;
  _pcolor = 10|7|12|13|2;
  xtics(1,12,1,0);
  ytics(-1.5,2.0,0.5,2);
  let xlab = "Jan" "Fev" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec";
  asclabel(xlab,0);
  _plegstr = "EUR/JPY\000EUR/USD\000GBP/EUR\000GBP/USD\000JPY/USD";
  _plegctl = {2 4 2 5.25};
  {varNames,tmp} = token(varNames);
  xy(seqa(1,1,12),SeasonalComponent);
```

Deuxième partie

Deuxième Séance : Dépendance et Copules

Troisième partie

Troisième Séance