Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include "alfonsi.h"

double psik (double t, double k)
{
    if (k==0.) return t;
    return (1-exp(-k*t))/k;
}

double DiscLawMatch5(int generator)
{
    double u=pnl_rand_uni(generator);
    if (u<1./6.) return -sqrt(3);
    if (u<1./3.) return sqrt(3);
    return 0;
}

double DiscLawMatch7(int generator)
{

    double u=2.*pnl_rand_uni(generator)-1.;
    double res=sqrt(6);
    if (fabs(u)<((res-2)/(2*res))) res=sqrt(3+res);
    else res=sqrt(3-res);
    if (u<0) return -res;
    return res;

}

static double O3_1 (double t, double x, double a, double k,
     double sig)
{
    double aux;

    if (k==0) aux=t;
    else aux=(1-exp(-k*t))/k;
```

```
    return x*exp(-k*t)+(a-0.25*sig*sig)*aux;
}

static double O3_2 (double t, double x, double sig)
{
    double aux=MAX(sqrt(x)+0.5*sig*t,0);
    return SQR(aux);
}

static double O3_3 (double t, double x,double a, double k,
    double sig,int ordre)
{
    double aux=(a-0.25*sig*sig+k*x)*0.5*sig*sig;

    if (aux>0) return x+sqrt(aux)*t+k*sig*sig*0.125*t*t;
    if (aux<0) return x+sqrt(-aux)*t-k*sig*sig*0.125*t*t;
    return x;
}

void Heston01(double *x1, double *x2, double *x3, double *x
    4, double dt, double dw,
              double a, double k, double sig, double mu,
    double rho, double Kseuil,int generator,int flag_cir)
{
    double dx=0.,aux,ratio,p;
    double sig2=SQR(sig);
    double pp,ee;
    double u1, u2, u3;
    double s,res,dt2,dw2;
    int ordre;
    double rd=0;

    if (flag_cir==1)
    {
        if (*x1>Kseuil*dt)
        {
            aux=exp(-k*0.5*dt);
            if (k==0.)
                u1=(a-SQR(0.5*sig))*dt*0.5;
            else   u1=(a-SQR(0.5*sig))*(1-aux)/k;
            dx=MAX(aux*SQR(sqrt(u1+aux*(*x1))+0.5*sig*dw)+
```

```
u1-*x1,-*x1);
    }
    else
    {
        aux=exp(-k*dt);
        u1=(*x1)*aux+a*(1-aux)/k;
        ratio=(SQR(*x1*aux)+(2*a+sig*sig)*(((1-aux*aux)
/(2*k))*a/k + (*x1-a/k)*(1-aux)*aux/k) )/(u1*u1);
        p=0.5*(1-sqrt(1-1/ratio));

        if (pnl_rand_uni(generator)< p) dx=u1/(2*p)-*x1
;
        else  dx=u1/(2*(1-p))-*x1;
    }
}
else  if (flag_cir==2)
{
    if (*x1<Kseuil*dt)
    {
        ee=exp(-k*dt);
        if (k==0.) pp=dt;
        else pp=(1-ee)/k;
        u1=*x1*ee+a*pp;
        u2=u1*u1+sig2*pp*(0.5*a*pp+*x1*ee);
        u3=u1*u2+sig2*pp*(2**x1**x1*ee*ee+pp*(a+0.5*si
g2)*(3**x1*ee+a*pp));
        s=(u3-u1*u2)/(u2-u1*u1);
        p=(u1*u3-u2*u2)/(u2-u1*u1);
        p=sqrt(s*s-4*p);
        u2=0.5*(s-p);
        u3=u2+p;

        if (pnl_rand_uni(generator)< (u1-u2)/(u3-u2))
dx=u3-*x1;
        else dx=u2-*x1;
    }
    else
    {
        if (a-0.25*sig2>0) ordre=1;
        else ordre=0;
        // On intègre 2
```

```
if (k==0.) dt2=dt;
else dt2=(exp(k*dt)-1)/k;
dw2=sqrt(dt2/dt)*dw;

//  else ordre=-1;

rd=3*pnl_rand_uni(generator);
if (rd<1)
{
    if (rd<0.5) s=-1;
    else s=1;
    if (ordre==1)
    {
        res=O3_3(s*dt2,*x1,a,0,sig,ordre);
        res=O3_2(dw2,res,sig);
        res=O3_1(dt2,res,a,0,sig);
    }
    else
    {
        res=O3_3(s*dt2,*x1,a,0,sig,ordre);
        res=O3_1(dt2,res,a,0,sig);
        res=O3_2(dw2,res,sig);
    }

    dx=exp(-k*dt)*res-*x1;
}
else
{
    if (rd<2)
    {
        if (rd-1<0.5) s=-1;
        else s=1;
        if (ordre==1)
        {
            res=O3_2(dw2,*x1,sig);
            res=O3_3(s*dt2,res,a,0,sig,ordre);
            res=O3_1(dt2,res,a,0,sig);
        }
        else
        {
```

```
                    res=O3_1(dt2,*x1,a,0,sig);
                    res=O3_3(s*dt2,res,a,0,sig,ordre);
                    res=O3_2(dw2,res,sig);
                }
                dx=exp(-k*dt)*res-*x1;
            }
            else
                if (rd>=2.)
                {
                    if (rd-2.<0.5) s=-1;
                    else s=1;
                    if (ordre==1)
                    {
                        res=O3_2(dw2,*x1,sig);
                        res=O3_1(dt2,res,a,0,sig);
                        res=O3_3(s*dt2,res,a,0,sig,ordr
e);
                    }
                    else
                    {
                        res=O3_1(dt2,*x1,a,0,sig);
                        res=O3_2(dw2,res,sig);
                        res=O3_3(s*dt2,res,a,0,sig,ordr
e);
                    }

                    dx=exp(-k*dt)*res-*x1;
                }
        }
    }
}

*x2=*x2+(*x1+0.5*dx)*dt;
*x4=*x4+0.5*(*x3)*dt;
*x3=*x3*exp((mu-rho*a/sig)*dt+rho*dx/sig+(rho*k/sig-0.5
)*(*x1+0.5*dx)*dt);
*x4=*x4+0.5*(*x3)*dt;
*x1=*x1+dx;

return;
}
```

```
void Heston02 (double *x1, double *x3,double dw2, double rh
    o)
{
    *x3=(*x3)*exp(sqrt((1-rho*rho)*(*x1))*dw2);

    return;
}

void fct_Heston(double *x1, double *x2, double *x3, double
    *x4, double dt, double dw,double dw2,double a, double k,
    double sig, double mu, double rho, double Kseuil,int generator,i
    nt flag_cir)
{
    if (pnl_rand_uni(generator)>0.5)
    {
        Heston02 ( x1, x3, dw2,  rho);
        Heston01 ( x1, x2, x3, x4, dt, dw, a,  k, sig,  mu,
      rho, Kseuil,generator,flag_cir);
    }
    else
    {
        Heston01 ( x1, x2, x3, x4, dt, dw, a,  k, sig,  mu,
      rho, Kseuil,generator,flag_cir);
        Heston02 ( x1, x3, dw2,  rho);
    }
    return;
}



/** Simulation of the Heston model, using the method propos
    ed by Aurélien Alfonsi.
* @param flag_SpotPaths flag to decides whether to store
    SpotPaths or not.
* @param SpotPaths will contain the paths simulation of th
    e spot
* @param flag_VarPaths flag to decides whether to store
    VarPaths or not.
* @param VarPaths will contain the paths simulation of the variance
* @param flag_AveragePaths flag to decides whether to sto
```

```
    re AveragePaths or not.
* @param AveragePaths will contain the paths simulation of
    the average of spot.
* @param S0 initial value of the spot.
* @param T last date in the simulation.
* @param r interest rate, divid dividend rate.
* @param V0, k, theta, sigma and rho: Heston parameters (
    Initial variance, mean reversion, long-run variance,
    volatility of variance and correlation)
* @param NbrMCsimulation number of simulated paths
* @param NbrDates number of sample in each path to be sto
    red in SpotPaths and VarPaths.
* @param NbrStepPerPeriod number of steps of discretization
     between T(i) and T(i+1)
* @param generator the index of the random generator to be
    used
* @param flag_cir parameter of Alfonsi's method
*/
int HestonSimulation_Alfonsi(int flag_SpotPaths, PnlMat *
    SpotPaths, int flag_VarPaths, PnlMat *VarPaths, int flag_Av
    eragePaths, PnlMat *AveragePaths, double S0, double T,
    double r, double divid, double V0,double k,double theta,double
    sigma,double rho, long NbrMCsimulation, int NbrDates, int
    NbrStepPerPeriod, int generator,int flag_cir)
{
    long i, j, m;
    double g1,g2, h, sqrt_h, w_t_1,w_t_2, aaa, Kseuil,aux,
    mu, t;

    double X1a, X2a, X3a, X4a;

    h = T /(double)((NbrDates-1)*NbrStepPerPeriod);
    sqrt_h = sqrt(h);
    aaa=k*theta;
    mu=r-divid;

    if (flag_cir==1)
        Kseuil=MAX((0.25*SQR(sigma)-aaa)*psik(h*0.5,k),0.);
    else
    {
        if (k==0)
```

```
        Kseuil=1;
    else Kseuil=(exp(k*h)-1)/(h*k);
    if (sigma*sigma <= 4*k*theta/3)
    {

        Kseuil=Kseuil*sigma*sqrt(k*theta-sigma*sigma/4)
/sqrt(2);
    }
    if (sigma*sigma > 4*k*theta/3 && sigma*sigma <= 4*
k*theta)
    {
        aux=(0.5*sigma*sqrt(3+sqrt(6))+sqrt(sigma*sigma
/4 - k*theta+sigma*sqrt(-sigma*sigma/4+ k*theta)/sqrt(2)))
;
        Kseuil=Kseuil*SQR(aux);
    }
    if (sigma*sigma > 4*k*theta)
    {
        aux=0.5*sigma*sqrt(3+sqrt(6))+ sqrt(sigma*sqrt(
sigma*sigma/4- k*theta)/sqrt(2));
        Kseuil=Kseuil*(sigma*sigma/4 - k*theta + SQR(au
x));
    }
    if (sigma*sigma == 4*k*theta) Kseuil=0;
}

if(flag_SpotPaths==1) pnl_mat_resize(SpotPaths, NbrDate
s, NbrMCsimulation);
if(flag_VarPaths==1) pnl_mat_resize(VarPaths, NbrDates,
 NbrMCsimulation);
if(flag_AveragePaths==1) pnl_mat_resize(AveragePaths,
NbrDates, NbrMCsimulation);

for (m=0; m<NbrMCsimulation; m++)
{
    /* Begin of the N iterations */
    t=0.;
    X1a=V0; // X1a: Variance
    X2a=0;  // X2a: Integral of Variance
    X3a=S0; // X3a: Spot
    X4a=0;  // X4a: Integral of Spot
```

```
        if(flag_VarPaths==1) MLET(VarPaths, 0, m) = X1a;
        if(flag_SpotPaths==1) MLET(SpotPaths, 0, m) = X3a;
        if(flag_AveragePaths==1) MLET(AveragePaths, 0, m) =
     X3a; // at time 0, Average is just equal to initial spot.

        for (i=1 ; i<=NbrDates-1 ; i++)
        {
            for (j=0 ; j<NbrStepPerPeriod; j++)
            {
                t += h;
                /*Discrete law obtained by matching of first
                five moments of a gaussian r.v.*/
                if (flag_cir==1)
                    g1=DiscLawMatch5(generator);
                else
                    g1=DiscLawMatch7(generator);
                w_t_1=sqrt_h*g1;

                g2= pnl_rand_normal(generator);
                w_t_2=sqrt_h*g2;

                fct_Heston(&X1a,&X2a,&X3a,&X4a, h,w_t_1,w_
    t_2,aaa,k,sigma,mu,rho,Kseuil,generator,flag_cir);
            }

            if(flag_VarPaths==1) MLET(VarPaths, i, m) = X1
    a;
            if(flag_SpotPaths==1) MLET(SpotPaths, i, m) = X
    3a;
            if(flag_AveragePaths==1) MLET(AveragePaths, i,
    m) = X4a/t;
        }

    }
    /* End of the NbrMCsimulation iterations */
    return OK;
}


/** Simulation of the Bates model, using the method propos
```

```
    ed by Aurélien Alfonsi.
* @param flag_SpotPaths flag to decides whether to store
    SpotPaths or not.
* @param SpotPaths will contain the paths simulation of th
    e spot
* @param flag_VarPaths flag to decides whether to store
    VarPaths or not.
* @param VarPaths will contain the paths simulation of the variance
* @param flag_AveragePaths flag to decides whether to sto
    re AveragePaths or not.
* @param AveragePaths will contain the paths simulation of
    the average of spot.
* @param S0 initial value of the spot.
* @param T last date in the simulation.
* @param r interest rate, divid dividend rate.
* @param V0, k, theta, sigma, rho, mu_jump, gamma2 and lam
    bda,: Bates parameters (Initial variance, mean reversion,
    long-run variance, volatility of variance, correlation, Lam
    bda, Mean of Jumps and Variance of Jumps)
* @param NbrMCsimulation number of simulated paths
* @param NbrDates number of sample in each path to be sto
    red in SpotPaths and VarPaths.
* @param NbrStepPerPeriod number of steps of discretization
     between T(i) and T(i+1)
* @param generator the index of the random generator to be
    used
* @param flag_cir parameter of Alfonsi's method
*/
int BatesSimulation_Alfonsi (int flag_SpotPaths, PnlMat *
    SpotPaths, int flag_VarPaths, PnlMat *VarPaths, int flag_Av
    eragePaths, PnlMat *AveragePaths, double S0, double T,
    double r, double divid, double V0, double k, double theta,
    double sigma, double rho, double mu_jump, double gamma2, double
    lambda, long NbrMCsimulation, int NbrDates, int NbrStepPerP
    eriod, int generator, int flag_cir)
{
    int i, j;
    long m;
    double g1, g2;
    double t_i, h, sqrt_h;
    double X1a, X2a, X3a, X4a;
```

```
double w_t_1, w_t_2;
double aaa, Kseuil,aux, mu;
double prev_jump, next_jump, h2,sqrt_h2,jump, correctio
n_mg, mu2,sg_jump;

h = T /(double)((NbrDates-1)*NbrStepPerPeriod);
sqrt_h = sqrt(h);
aaa = k*theta;
mu=r-divid;
prev_jump=0;

sg_jump=sqrt(gamma2);
correction_mg=lambda*(exp(mu_jump+0.5*gamma2)-1);
mu2=mu-correction_mg;
if (flag_cir==1)
    Kseuil=MAX((0.25*SQR(sigma)-aaa)*psik(h*0.5,k),0.);
else
{
    if (k==0)
        Kseuil=1;
    else Kseuil=(exp(k*h)-1)/(h*k);
    if (sigma*sigma <= 4*k*theta/3)
    {

        Kseuil=Kseuil*sigma*sqrt(k*theta-sigma*sigma/4)
/sqrt(2);
    }
    if (sigma*sigma > 4*k*theta/3 && sigma*sigma <= 4*
k*theta)
    {
        aux=(0.5*sigma*sqrt(3+sqrt(6))+sqrt(sigma*sigma
/4 - k*theta+sigma*sqrt(-sigma*sigma/4+ k*theta)/sqrt(2)))
;
        Kseuil=Kseuil*SQR(aux);
    }
    if (sigma*sigma > 4*k*theta)
    {
        aux=0.5*sigma*sqrt(3+sqrt(6))+ sqrt(sigma*sqrt(
sigma*sigma/4- k*theta)/sqrt(2));
        Kseuil=Kseuil*(sigma*sigma/4 - k*theta + SQR(au
x));
```

```
    }
    if (sigma*sigma == 4*k*theta) Kseuil=0;
}

if(flag_SpotPaths==1) pnl_mat_resize(SpotPaths, NbrDate
s, NbrMCsimulation);
if(flag_VarPaths==1) pnl_mat_resize(VarPaths, NbrDates,
 NbrMCsimulation);
if(flag_AveragePaths==1) pnl_mat_resize(AveragePaths,
NbrDates, NbrMCsimulation);

for (m=0; m<NbrMCsimulation; m++)
{
    /* Begin of the N iterations */
    t_i=0.;
    X1a=V0; // X1a: Volatility
    X2a=0;  // X1a: Integral of Volatility
    X3a=S0; // X1a: Spot
    X4a=0;  // X1a: Integral of Spot

    if(flag_VarPaths==1) MLET(VarPaths, 0, m) = X1a;
    if(flag_SpotPaths==1) MLET(SpotPaths, 0, m) = X3a;
    if(flag_AveragePaths==1) MLET(AveragePaths, 0, m) =
 X3a; // at time 0, Average is just equal to initial spot.

    next_jump=-log(pnl_rand_uni(generator))/lambda;

    for (i=1 ; i<=NbrDates-1 ; i++)
    {
        for (j=0 ; j<NbrStepPerPeriod; j++)
        {
            t_i+=h;

            /*Discrete law obtained by matching of fir
st five moments of a gaussian r.v.*/
            if (next_jump > t_i)
            {
                if (flag_cir==1)
                    g1=DiscLawMatch5(generator);
                else
                    g1=DiscLawMatch7(generator);
```

```
                w_t_1=sqrt_h*g1;

                g2= pnl_rand_normal(generator);
                w_t_2=sqrt_h*g2;

                fct_Heston(&X1a,&X2a,&X3a,&X4a, h,w_t_1
,w_t_2,aaa,k,sigma,mu2,rho,Kseuil,generator,flag_cir);
            }
            else
            {
                h2=next_jump-(t_i-h);
                sqrt_h2=sqrt(h2);

                while (next_jump <= t_i)
                {

                    if (flag_cir==1)
                        g1=DiscLawMatch5(generator);
                    else
                        g1=DiscLawMatch7(generator);
                    w_t_1=sqrt_h2*g1;

                    g2 = pnl_rand_normal(generator);
                    w_t_2 = sqrt_h2*g2;
                    fct_Heston(&X1a,&X2a,&X3a,&X4a, h2,
w_t_1,w_t_2,aaa,k,sigma,mu2,rho,Kseuil,generator,flag_cir);
                    prev_jump = next_jump;
                    next_jump = next_jump-log(pnl_rand_
uni(generator))/lambda;
                    h2 = next_jump-prev_jump;
                    sqrt_h2 = sqrt(h2);
                    jump = exp(mu_jump+sg_jump*pnl_rand
_normal(generator));
                    X3a = X3a*jump;
                }

                h2=t_i-prev_jump;
                sqrt_h2=sqrt(h2);


                if (flag_cir==1)
```

```
                        g1=DiscLawMatch5(generator);
                    else
                        g1=DiscLawMatch7(generator);
                    w_t_1=sqrt_h2*g1;

                    g2= pnl_rand_normal(generator);
                    w_t_2=sqrt_h2*g2;
                    fct_Heston(&X1a,&X2a,&X3a,&X4a, h2,w_t_
    1,w_t_2,aaa,k,sigma,mu2,rho,Kseuil,generator,flag_cir);
                }
            }

            if(flag_VarPaths==1) MLET(VarPaths, i, m) = X1
    a;
            if(flag_SpotPaths==1) MLET(SpotPaths, i, m) = X
    3a;
            if(flag_AveragePaths==1) MLET(AveragePaths, i,
    m) = X4a/t_i;
        }
    }

    return OK;
}



/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////
/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////
// Heston model "HestonSimulation_Alfonsi". Indeed, this    version provides Va
// integral of the variance.
/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////
/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////
/** Simulation of the Heston model, using the method propos
    ed by Aurélien Alfonsi.
* @param flag_SpotPaths flag to decides whether to store
    SpotPaths or not.
* @param SpotPaths will contain the paths simulation of th
```

```
    e spot
* @param flag_VarPaths flag to decides whether to store
    VarPaths or not.
* @param VarPaths will contain the paths simulation of the variance
* @param flag_AveragePaths flag to decides whether to sto
    re AveragePaths or not.
* @param AveragePaths will contain the paths simulation of
    the average of spot.z
***************** This is the new parameter added to the
    basic version ***********************
* @param VarianceInt will contain the paths simulation of
    the integral of the variance.
*** Contrary to the other parameters, we do not use a flag
    to decide whether to return VarianceInt
*** or not. Thus, we always return this parameter because
    it is always needed.
***********************************************************
    **************************************
* @param S0 initial value of the spot.
* @param T last date in the simulation.
* @param r interest rate, divid dividend rate.
* @param V0, k, theta, sigma and rho: Heston parameters (
    Initial variance, mean reversion, long-run variance,
    volatility of variance and correlation)
* @param NbrMCsimulation number of simulated paths
* @param NbrDates number of sample in each path to be sto
    red in SpotPaths and VarPaths.
* @param NbrStepPerPeriod number of steps of discretization
     between T(i) and T(i+1)
* @param generator the index of the random generator to be
    used
* @param flag_cir parameter of Alfonsi's method
*/

int HestonSimulation_Alfonsi_Modified(int flag_SpotPaths,
    PnlMat *SpotPaths, int flag_VarPaths, PnlMat *VarPaths,
    int flag_AveragePaths, PnlMat *AveragePaths, PnlMat *
    VarianceInt, double S0, double T, double r, double divid,double V0
    , double k, double theta,double sigma,double rho,long NbrM
    Csimulation, int NbrDates, int NbrStepPerPeriod,int     generator,int flag_c
{
```

```
long i, j, m;
double g1,g2, h, sqrt_h, w_t_1,w_t_2, aaa, Kseuil,aux,
mu, t;

double X1a, X2a, X3a, X4a;

h = T /(double)((NbrDates-1)*NbrStepPerPeriod);
sqrt_h = sqrt(h);
aaa=k*theta;
mu=r-divid;

if (flag_cir==1)
    Kseuil=MAX((0.25*SQR(sigma)-aaa)*psik(h*0.5,k),0.);
else
{
    if (k==0)
        Kseuil=1;
    else Kseuil=(exp(k*h)-1)/(h*k);
    if (sigma*sigma <= 4*k*theta/3)
    {

        Kseuil=Kseuil*sigma*sqrt(k*theta-sigma*sigma/4)
/sqrt(2);
    }
    if (sigma*sigma > 4*k*theta/3 && sigma*sigma <= 4*
k*theta)
    {
        aux=(0.5*sigma*sqrt(3+sqrt(6))+sqrt(sigma*sigma
/4 - k*theta+sigma*sqrt(-sigma*sigma/4+ k*theta)/sqrt(2)))
;
        Kseuil=Kseuil*SQR(aux);
    }
    if (sigma*sigma > 4*k*theta)
    {
        aux=0.5*sigma*sqrt(3+sqrt(6))+ sqrt(sigma*sqrt(
sigma*sigma/4- k*theta)/sqrt(2));
        Kseuil=Kseuil*(sigma*sigma/4 - k*theta + SQR(au
x));
    }
    if (sigma*sigma == 4*k*theta) Kseuil=0;
}
```

```
//   No need to resize in this version, indeed, the size
   is already fixed
//   in the principal function MalliavinImproved_Heston.
//if(flag_SpotPaths==1) pnl_mat_resize(SpotPaths, NbrD
  ates, NbrMCsimulation);
  //if(flag_VarPaths==1) pnl_mat_resize(VarPaths, NbrDate
  s, NbrMCsimulation);
//pnl_mat_resize(VarianceInt, NbrDates, NbrMCsimulation)
  ;
  if(flag_AveragePaths==1) pnl_mat_resize(AveragePaths,
  NbrDates, NbrMCsimulation);


  for (m=0; m<NbrMCsimulation; m++)
  {
      // Begin of the N iterations
      t=0.;
      X1a=V0; // X1a: Variance
      X2a=0;  // X2a: Integral of Variance
      X3a=S0; // X3a: Spot
      X4a=0;  // X4a: Integral of Spot

      if(flag_VarPaths==1) MLET(VarPaths, 0, m) = X1a;
      if(flag_SpotPaths==1) MLET(SpotPaths, 0, m) = X3a;
      if(flag_AveragePaths==1) MLET(AveragePaths, 0, m) =
   X3a; // at time 0, Average is just equal to initial spot.
  MLET(VarianceInt, 0, m) = X2a;

      for (i=1 ; i<=NbrDates-1 ; i++)
      {
          for (j=0 ; j<NbrStepPerPeriod; j++)
          {
              t += h;
              //Discrete law obtained by matching of first
              //five moments of a gaussian r.v.
              if (flag_cir==1)
                  g1=DiscLawMatch5(generator);
              else
                  g1=DiscLawMatch7(generator);
```

```
                w_t_1=sqrt_h*g1;

                g2= pnl_rand_normal(generator);
                w_t_2=sqrt_h*g2;

                fct_Heston(&X1a,&X2a,&X3a,&X4a, h,w_t_1,w_
    t_2,aaa,k,sigma,mu,rho,Kseuil,generator,flag_cir);
            }

            if(flag_VarPaths==1) MLET(VarPaths, i, m) = X1
    a;
            if(flag_SpotPaths==1) MLET(SpotPaths, i, m) = X
    3a;
            if(flag_AveragePaths==1) MLET(AveragePaths, i,
    m) = X4a/t;
      MLET(VarianceInt, i, m) = X2a;
        }

    }
    // End of the NbrMCsimulation iterations
    return OK;
}


#endif //PremiaCurrentVersion
```

References