```
   Help
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2009+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include "wienerhopf.h"
#include "wienerhopf_rs.h"
// model = 1 TSL
// model = 2  NIG
// model = 3  VGP
// model = 4  KOU

/*
 * A wrapper around pnl_real_ifft_inplace used by the ot
    her functions of this
 * file. The order of the storage is changed
 * if i is even then  a(2*i)+ I *  a(2i+1) is the i-th
 * coefficients of the Fourier transform
 * if i is odd then  a(N-2*i)+ I * a(N-2i+1) is the i-th
 * coefficients of the Fourier transform
 */
static void fft_real (PnlVect * a,int fft_size, int sign)
{
  int    i, opposite;
  double last, second;

  switch (sign)
    {
    case 1 : /* backward */
      second = pnl_vect_get (a, 1);
      opposite = 1;
      for ( i=1 ; i<fft_size - 1 ; i++ )
        {
          pnl_vect_set (a, i, opposite * pnl_vect_get (a,
    i + 1));
          opposite = - opposite;
        }
      pnl_vect_set (a , fft_size - 1, second);

      pnl_real_ifft_inplace (a->array, fft_size);
```

```
      break;
    case -1 : /* forward */
      pnl_real_fft_inplace (a->array, fft_size);
      last = pnl_vect_get (a, fft_size - 1);
      opposite = -1;
      for ( i=fft_size -1 ; i>1 ; i-- )
        {
          pnl_vect_set (a, i, opposite * pnl_vect_get (a,
    i-1));
          opposite = - opposite;
        }
      pnl_vect_set (a , 1, last);
      break;
    }
}


//=========================================================
    ========
static int findcoef(int model, double mu, double sigma,
    double lm1, double lp1,
        double num, double nup,
        double cnum,double cnup, double q, double r1,
        double T, double h, long int kmax,
        double er, long int Nt,
        PnlVect * al1)

{
  PnlVect *bl1;
  long int k;
  long int i;
  double mod;
  PnlVect *alin1;
  double xi,xip,xim,anp,anm,sxi,nup1,num1;
  double lpm1;

  double sg2;
  double cpl;     //  !!!!!!
  double cml;

  /*Memory allocation for space grid*/
```

```
  bl1=pnl_vect_create(2*kmax+1);

  ////////////////////////// this part depends on model
if(model==1/*TSL*/)
{

  lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  nup1=nup/2;
  num1=num/2;
  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=pow(xi*xi+lp1*lp1,nup1);
  xim=pow(xi*xi+lm1*lm1,num1);
  anp=nup*atan(xi/lp1);
  anm=num*atan(xi/lm1);
  bl1->array[1]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm);
  bl1->array[2*kmax]=-cnup*xip*sin(anp)-cnum*xim*sin(anm);
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
    {
      xip=pow(xi*xi+lp1*lp1,nup1);
      xim=pow(xi*xi+lm1*lm1,num1);
      anp=nup*atan(xi/lp1);
      anm=num*atan(xi/lm1);
      bl1->array[2*i]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(
    anm);
      bl1->array[2*i+1]=-cnup*xip*sin(anp)-cnum*xim*sin(an
    m);
      i++;
      xi=xi+sxi;
 }while(i<kmax);
}// END TSL
if(model==2/*NIG*/)
{
       lpm1=q-sqrt(-lp1*lm1)*cnup;
```

```
  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=0.5*atan(xim/xip);
  bl1->array[1]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos(
    anp);
  bl1->array[2*kmax]=cnup*pow(xip*xip+xim*xim, 0.25)*sin(
    anp);
  sxi=xi/kmax;
  xi=sxi;

       i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=0.5*atan(xim/xip);
    bl1->array[2*i]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*
    cos(anp);
    bl1->array[2*i+1]=cnup*pow(xip*xip+xim*xim, 0.25)*si
    n(anp);
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END NIG
if(model == 3/*VGP*/)
{
       lpm1=q-log(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=atan(xim/xip);
  bl1->array[1]=lpm1+cnup*log(xip*xip+xim*xim)/2;
  bl1->array[2*kmax]=cnup*anp-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

    i=1;
```

```
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=atan(xim/xip);
    bl1->array[2*i]=lpm1+cnup*log(xip*xip+xim*xim)/2;
    bl1->array[2*i+1]=cnup*anp-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END VGP
if(model == 4/*KOU*/)
{
  // nup==sigma
     sg2=sigma*sigma/2;
  cpl=cnup*lp1;     //  !!!!!!
  cml=cnum*lm1;

  //lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi+lp1*lp1;
  xim=xi*xi+lm1*lm1;
  bl1->array[1]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
  bl1->array[2*kmax]=xi*(cpl/xip+cml/xim)-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi+lp1*lp1;
    xim=xi*xi+lm1*lm1;
    bl1->array[2*i]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
    bl1->array[2*i+1]=xi*(cpl/xip+cml/xim)-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END KOU
```

```
  ////////////////////////////////////////////////////

  alin1=pnl_vect_create(2*kmax);

  alin1->array[0]=q/bl1->array[0];
  mod=bl1->array[2*kmax]*bl1->array[2*kmax]+bl1->array[1]*
    bl1->array[1];
  alin1->array[1]=q*bl1->array[1]/mod;
  //alin1->array[1]=q/bl1->array[1];

  i=1;
  do
    {   mod=bl1->array[2*i+1]*bl1->array[2*i+1]+bl1->array[
    2*i]*bl1->array[2*i];
      alin1->array[2*i+1]=-q*bl1->array[2*i+1]/mod;
      alin1->array[2*i]=q*bl1->array[2*i]/mod;
      i++;
    }while(i<kmax);


  k=0;
  while(k<2*kmax)
    {
      k++;
      al1->array[k]=alin1->array[k-1];
 }
  pnl_vect_free(&alin1);
  pnl_vect_free(&bl1);
  return 1;
}


static int findcoefnew(int model, double mu, double sigma,
    double lm1, double lp1,
        double num, double nup,
        double cnum,double cnup, double q, double r1,
        double T, double h, long int kmax,
        double er, long int Nt,
        PnlVect * al1)

{
```

```
  long int Nx;
  PnlVect *bl1;
  PnlVect *alin1;
  double sg2;
  double cpl;
  double cml;

  long int k;
  long int i;
  double mod;

  double xi,xip,xim,anp,anm,sxi,nup1,num1;
  long int Nmax=2*kmax;
  double lpm1;

  Nx=Nmax; /*number of space points*/

  /*Memory allocation for space grid*/

  bl1 = pnl_vect_create(Nx+1);
  alin1 = pnl_vect_create(Nx);

  ////////////////////////// this part depends on the model
if(model==1/*TSL*/)
{

  lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum+fabs(mu)/h;

  nup1=nup/2.0;
  num1=num/2.0;
  LET(bl1,0)=q;
  xi=-M_PI/h;
  xip=pow(xi*xi+lp1*lp1,nup1);
  xim=pow(xi*xi+lm1*lm1,num1);
  anp=nup*atan(xi/lp1);
  anm=num*atan(xi/lm1);
  LET(bl1,1)=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm)+fabs(
    mu)/h;
  LET(bl1,2*kmax)=-cnup*xip*sin(anp)-cnum*xim*sin(anm);
  sxi=xi/kmax;
  xi=sxi;
```

```
  i=1;
  do
    {
      xip=pow(xi*xi+lp1*lp1,nup1);
      xim=pow(xi*xi+lm1*lm1,num1);
      anp=nup*atan(xi/lp1);
      anm=num*atan(xi/lm1);
      LET(bl1,2*i)=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm)
    -fabs(mu)*cos(xi*h)/h;
      LET(bl1,2*i+1)=-cnup*xip*sin(anp)-cnum*xim*sin(anm)-
    mu*sin(xi*h)/h;
      i++;
      xi=xi+sxi;
 }while(i<kmax);
}// END TSL
if(model==2/*NIG*/)
{
       lpm1=q-sqrt(-lp1*lm1)*cnup;

  LET(bl1,0)=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=0.5*atan(xim/xip);
  LET(bl1,1)=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos(anp)
    ;
  LET(bl1,2*kmax)=cnup*pow(xip*xip+xim*xim, 0.25)*sin(anp)
    ;
  sxi=xi/kmax;
  xi=sxi;

       i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=0.5*atan(xim/xip);
    LET(bl1,2*i)=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos
    (anp);
    LET(bl1,2*i+1)=cnup*pow(xip*xip+xim*xim, 0.25)*sin(an
```

```
    p);
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END NIG
if(model == 3/*VGP*/)
{
       lpm1=q-log(-lp1*lm1)*cnup;

  LET(bl1,0)=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=atan(xim/xip);
  LET(bl1,1)=lpm1+cnup*log(xip*xip+xim*xim)/2;
  LET(bl1,2*kmax)=cnup*anp-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

     i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=atan(xim/xip);
    LET(bl1,2*i)=lpm1+cnup*log(xip*xip+xim*xim)/2;
    LET(bl1,2*i+1)=cnup*anp-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END VGP
if(model == 4/*KOU*/)
{
    sg2=sigma*sigma/2;
  cpl=cnup*lp1;
  cml=cnum*lm1;

  LET(bl1,0)=q;
  xi=-M_PI/h;
  xip=xi*xi+lp1*lp1;
  xim=xi*xi+lm1*lm1;
```

```
  LET(bl1,1)=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
  LET(bl1,2*kmax)=xi*(cpl/xip+cml/xim)-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi+lp1*lp1;
    xim=xi*xi+lm1*lm1;
    LET(bl1,2*i)=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
    LET(bl1,2*i+1)=xi*(cpl/xip+cml/xim)-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END KOU
if(model == 6/*BS or Heston*/)
{
  // num==var
  double sg2=sigma/2;
  if (sg2>0.00001)
  {  LET(bl1,0)=q;
    xi=-M_PI/h;
    xip=xi*xi;
    LET(bl1,1)=q+sg2*xip;
    LET(bl1,2*kmax)=-mu*xi;
    sxi=xi/kmax;
    xi=sxi;

    i=1;
    do
    {
    xip=xi*xi;
    LET(bl1,2*i)=q+sg2*xip;
    LET(bl1,2*i+1)=-mu*xi;
    i++;
    xi=xi+sxi;
    }while(i<kmax);
  }
  else
```

```
    {
    lpm1=q+fabs(mu)/h;
    LET(bl1,0)=q;
    xi=-M_PI/h;
    xip=xi*xi;
    LET(bl1,1)=lpm1+sg2*xip+fabs(mu)/h;
    LET(bl1,2*kmax)=0;
    sxi=xi/kmax;
    xi=sxi;

    i=1;
    do
    {
    xip=xi*xi;
    LET(bl1,2*i)=lpm1+sg2*xip-fabs(mu)*cos(xi*h)/h;
    LET(bl1,2*i+1)=-mu*sin(xi*h)/h;
    i++;
    xi=xi+sxi;
    }while(i<kmax);
  }
}  //END Heston
  ////////////////////////////////////////////////////

  LET(alin1,0)=q/GET(bl1,0);
  mod=GET(bl1,2*kmax)*GET(bl1,2*kmax)+GET(bl1,1)*GET(bl1,1)
    ;
  LET(alin1,1)=q*GET(bl1,1)/mod;

  i=1;
  do
    {   mod=GET(bl1,2*i+1)*GET(bl1,2*i+1)+GET(bl1,2*i)*GET(
    bl1,2*i);
      LET(alin1,2*i+1)=-q*GET(bl1,2*i+1)/mod;
      LET(alin1,2*i)=q*GET(bl1,2*i)/mod;
      i++;
    }while(i<kmax);


  k=0;
  while(k<2*kmax)
    {
```

```
      k++;
      LET(al1,k)=GET(alin1,k-1);
 }
  pnl_vect_free(&alin1);
  pnl_vect_free( &bl1);
  return 1;
}
//=============================================================
    ========
static int findcoef_sw(int model, double mu, double sigma,
    double lm1, double lp1,
        double num, double nup,
        double cnum,double cnup, double q, double r1,
        double T, double h, long int kmax,
        double del, long int Nt,
        PnlVect *al1, PnlVect *bl1)

{
  //double *bl1;
  long int k;
  long int i;
  double mod;
  PnlVect *alin1;
  double xi,xip,xim,anp,anm,sxi,nup1,num1,del1;
  double lpm1;
  double dt1;

    /*Memory allocation for space grid*/

 // bl1=new double[2*kmax+1];

 dt1=Nt/T;
 del1=1/dt1-del;
// cout<<del1<<"   "<<dt1<<endl;
  ////////////////////////// this part depends on model
if(model==1/*TSL*/)
{

  lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum+fabs(mu)/h;

  nup1=nup/2.0;
```

```
  num1=num/2.0;
  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=pow(xi*xi+lp1*lp1,nup1);
  xim=pow(xi*xi+lm1*lm1,num1);
  anp=nup*atan(xi/lp1);
  anm=num*atan(xi/lm1);
  bl1->array[1]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm)+
    fabs(mu)/h;
  bl1->array[2*kmax]=-cnup*xip*sin(anp)-cnum*xim*sin(anm);
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
    {
      xip=pow(xi*xi+lp1*lp1,nup1);
      xim=pow(xi*xi+lm1*lm1,num1);
      anp=nup*atan(xi/lp1);
      anm=num*atan(xi/lm1);
      bl1->array[2*i]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(
    anm)-fabs(mu)*cos(xi*h)/h;
      bl1->array[2*i+1]=-cnup*xip*sin(anp)-cnum*xim*sin(an
    m)-mu*sin(xi*h)/h;
      i++;
      xi=xi+sxi;
 }while(i<kmax);
}// END TSL
if(model==2/*NIG*/)
{
       lpm1=q-sqrt(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=0.5*atan(xim/xip);
  bl1->array[1]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos(
    anp);
  bl1->array[2*kmax]=cnup*pow(xip*xip+xim*xim, 0.25)*sin(
    anp);
```

```
  sxi=xi/kmax;
  xi=sxi;

       i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=0.5*atan(xim/xip);
    bl1->array[2*i]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*
    cos(anp);
    bl1->array[2*i+1]=cnup*pow(xip*xip+xim*xim, 0.25)*si
    n(anp);
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END NIG
if(model == 3/*VGP*/)
{
       lpm1=q-log(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=atan(xim/xip);
  bl1->array[1]=lpm1+cnup*log(xip*xip+xim*xim)/2;
  bl1->array[2*kmax]=cnup*anp-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

    i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=atan(xim/xip);
    bl1->array[2*i]=lpm1+cnup*log(xip*xip+xim*xim)/2;
    bl1->array[2*i+1]=cnup*anp-mu*xi;
    i++;
    xi=xi+sxi;
```

```
  }while(i<kmax);
}// END VGP
if(model == 4/*KOU*/)
{
  // nup==sigma
     double sg2=sigma*sigma/2;
  double cpl=cnup*lp1;     //  !!!!!!
  double cml=cnum*lm1;

  //lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi+lp1*lp1;
  xim=xi*xi+lm1*lm1;
  bl1->array[1]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
  bl1->array[2*kmax]=xi*(cpl/xip+cml/xim)-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi+lp1*lp1;
    xim=xi*xi+lm1*lm1;
    bl1->array[2*i]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
    bl1->array[2*i+1]=xi*(cpl/xip+cml/xim)-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END KOU
if(model == 5/*Merton*/)//nup=delta; cnum=lambda;cnup=gam0;
{
  // nup==sigma
     double sg2=sigma*sigma/2;
  double del2=nup*nup/2;     //  !!!!!!
  double gam1=cnup;

  //lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;
```

```
  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi;
  bl1->array[1]=q+sg2*xip+cnum*(1-exp(-del2*xip)*cos(gam1*
    xi));
  bl1->array[2*kmax]=-cnum*exp(-del2*xip)*sin(gam1*xi)-mu*
    xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi;
    bl1->array[2*i]=q+sg2*xip+cnum*(1-exp(-del2*xip)*cos(
    gam1*xi));
    bl1->array[2*i+1]=-cnum*exp(-del2*xip)*sin(gam1*xi)-
    mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END MERTON
  ////////////////////////////////////////////////////

  alin1=pnl_vect_create(2*kmax);

  alin1->array[0]=q/bl1->array[0];
  bl1->array[0]=exp(del1*(bl1->array[0]-dt1));
  mod=bl1->array[2*kmax]*bl1->array[2*kmax]+bl1->array[1]*
    bl1->array[1];
  alin1->array[1]=q*bl1->array[1]/mod;
  bl1->array[1]=exp(del1*(bl1->array[1]-dt1));
  //alin1->array[1]=q/bl1->array[1];
  i=1;
  do
    { mod=bl1->array[2*i+1]*bl1->array[2*i+1]+bl1->array[2*
    i]*bl1->array[2*i];
      alin1->array[2*i+1]=-q*bl1->array[2*i+1]/mod;
      alin1->array[2*i]=q*bl1->array[2*i]/mod;
    bl1->array[2*i]=exp(del1*(bl1->array[2*i]-dt1));
```

```
    mod=del1*bl1->array[2*i+1];
    bl1->array[2*i+1]=bl1->array[2*i]*sin(mod);
    bl1->array[2*i]=bl1->array[2*i]*cos(mod);
    i++;
    }while(i<kmax);

  k=0;
  while(k<2*kmax)
    {
      k++;
      al1->array[k]=alin1->array[k-1];
 }
  pnl_vect_free(&alin1);
  return 1;
}
//-------------------------------------------findfactor---
    ------------
static int findfactor(double np, double nm, double h, long
    int kmax,  double lp1, double lm1, PnlVect *alin1, PnlVect *
    tp1, PnlVect *tm1)
{
   PnlVect *tb1, *tbp1, *tbm1, *ltp1, *ltm1;

 long int i;

  double xi, xip, xim, anp, anm, sxi, abp, abm, nup1, num1;
 double t1, t2;
 double angle;
  double mod;

long int Nmax=2*kmax;

  tbp1= pnl_vect_create(Nmax+2);
 ltp1= pnl_vect_create(Nmax+2);
 ltm1= pnl_vect_create(Nmax+2);
 tb1= pnl_vect_create(Nmax+2);
 tbm1= pnl_vect_create(Nmax+2);

nup1 = 0.;
num1 = 0.;
```

```
if (np==nm)
  {
    nup1=np/2.0;
    num1=nm/2.0;
  }
if (np>nm)
  {
    nup1=np;
 num1=0.0;
  }
if (np<nm)
  {
    nup1=0.0;
 num1=nm;
  }

    abp=pow(lp1,nup1);
    abm=pow(-lm1,num1);

    LET(ltp1,0)=1;
    LET(ltm1,0)=1;
    xi=-M_PI/h;

    xip=pow(xi*xi+lp1*lp1,nup1/2.0)/abp;
    xim=pow(xi*xi+lm1*lm1,num1/2.0)/abm;
    anp=nup1*atan(xi/lp1);
    anm=num1*atan(xi/lm1);
    LET(ltp1,1)=xip*cos(anp);
    LET(ltm1,1)=xim*cos(anm);
    LET(ltp1,2*kmax)=xip*sin(anp);
    LET(ltm1,2*kmax)=xim*sin(anm);
    sxi=xi/kmax;
    xi=sxi;

   i=1;
do
{
 xip=pow(xi*xi+lp1*lp1,nup1/2.0)/abp;
 xim=pow(xi*xi+lm1*lm1,num1/2.0)/abm;
 anp=nup1*atan(xi/lp1);
 anm=num1*atan(xi/lm1);
```

```
 LET(ltp1,2*i)=xip*cos(anp);
 LET(ltp1,2*i+1)=xip*sin(anp);
 LET(ltm1,2*i)=xim*cos(anm);
 LET(ltm1,2*i+1)=xim*sin(anm);
 i++;
 xi=xi+sxi;
}while(i<kmax);

LET(alin1,1)=GET(alin1,1)*(GET(ltp1,1)*GET(ltm1,1)-GET(lt
   p1,2*kmax)*GET(ltm1,2*kmax));
for(i = 1; i <= kmax-1; i++)
  {
    t1 = GET(alin1,2*i);
    t2 = GET(alin1,2*i+1);
    LET(alin1,2*i)= t1*GET(ltm1,2*i)-t2*GET(ltm1,2*i+1);
    LET(alin1,2*i+1)= t2*GET(ltm1,2*i)+t1*GET(ltm1,2*i+1);
  }
for(i = 1; i <= kmax-1; i++)
  {
       t1 = GET(alin1,2*i);
       t2 = GET(alin1,2*i+1);
       LET(alin1,2*i)= t1*GET(ltp1,2*i)-t2*GET(ltp1,2*i+1)
   ;
       LET(alin1,2*i+1)= t2*GET(ltp1,2*i)+t1*GET(ltp1,2*i+
   1);
  }

LET(ltp1,0)=1/GET(ltp1,0);
LET(ltm1,0)=1/GET(ltm1,0);
LET(ltp1,1)=GET(ltp1,1)/(GET(ltp1,1)*GET(ltp1,1)+GET(ltp1,
   2*kmax)*GET(ltp1,2*kmax));
LET(ltm1,1)=GET(ltm1,1)/(GET(ltm1,1)*GET(ltm1,1)+GET(ltm1,
   2*kmax)*GET(ltm1,2*kmax));

i=1;
do
  {   mod=GET(ltp1,2*i+1)*GET(ltp1,2*i+1)+GET(ltp1,2*i)*
   GET(ltp1,2*i);
    LET(ltp1,2*i+1)=-GET(ltp1,2*i+1)/mod;
    LET(ltp1,2*i)=GET(ltp1,2*i)/mod;
```

```
    mod=GET(ltm1,2*i+1)*GET(ltm1,2*i+1)+GET(ltm1,2*i)*GET(
   ltm1,2*i);
    LET(ltm1,2*i+1)=-GET(ltm1,2*i+1)/mod;
    LET(ltm1,2*i)=GET(ltm1,2*i)/mod;
    i++;
  }while(i<kmax);

LET(tb1,0)=log(GET(alin1,0));
LET(tb1,1)=log(GET(alin1,1));

i=1;
do
  {   mod=GET(alin1,2*i+1)*GET(alin1,2*i+1)+GET(alin1,2*i)
   *GET(alin1,2*i);
    LET(tb1,2*i)=0.5*log(mod);
    if (GET(alin1,2*i)==0)
      {
  if (GET(alin1,2*i+1)>0)
    {LET(tb1,2*i+1)=M_PI/2.0;}
  else
    {LET(tb1,2*i+1)=-M_PI/2.0;}
      }
    else
      { angle=atan(GET(alin1,2*i+1)/GET(alin1,2*i));
  if (GET(alin1,2*i)>0) {LET(tb1,2*i+1)=angle;}
  if (GET(alin1,2*i)<0) {if (GET(alin1,2*i+1)<0) {LET(tb1
   ,2*i+1)=angle-M_PI;}
    else {LET(tb1,2*i+1)=angle+M_PI;}
  }
      }
    i++;
  }while(i<kmax);


fft_real(tb1, 2*kmax, 1);

 i=1;
LET(tbp1,0)=0;
LET(tbm1,0)=0;
do
{ LET(tbp1,0)=GET(tbp1,0)-GET(tb1,i);
```

```
  LET(tbp1,i)=GET(tb1,i);
  LET(tbm1,i)=0;
  i++;
}while(i<kmax);
do
  { LET(tbm1,i)=GET(tb1,i);
    LET(tbp1,i)=0;
    LET(tbm1,0)=GET(tbm1,0)-GET(tb1,i);
    i++;
  }while(i<2*kmax);

fft_real(tbp1, 2*kmax, -1);
fft_real(tbm1, 2*kmax, -1);

LET(tp1,0)=exp(GET(tbp1,0));
LET(tp1,1)=exp(GET(tbp1,1));
LET(tm1,0)=exp(GET(tbm1,0));
LET(tm1,1)=exp(GET(tbm1,1));

i=1;
do
  {   mod=exp(GET(tbp1,2*i));
    LET(tp1,2*i)=mod*cos(GET(tbp1,2*i+1));
    LET(tp1,2*i+1)=mod*sin(GET(tbp1,2*i+1));
    mod=exp(GET(tbm1,2*i));
    LET(tm1,2*i)=mod*cos(GET(tbm1,2*i+1));
    LET(tm1,2*i+1)=mod*sin(GET(tbm1,2*i+1));
    i++;
  }while(i<kmax);

LET(tp1,0)= GET(tp1,0)*GET(ltp1,0);
LET(tp1,1)= GET(tp1,1)*GET(ltp1,1);

LET(tm1,0)= GET(tm1,0)*GET(ltm1,0);
LET(tm1,1)= GET(tm1,1)*GET(ltm1,1);

 for(i = 1; i <= kmax-1; i++)
   {
     t1 = GET(tm1,2*i);
     t2 = GET(tm1,2*i+1);
     LET(tm1,2*i)= t1*GET(ltm1,2*i)-t2*GET(ltm1,2*i+1);
```

```
      LET(tm1,2*i+1)= t2*GET(ltm1,2*i)+t1*GET(ltm1,2*i+1);
    }
  for(i = 1; i <= kmax-1; i++)
    {
      t1 = GET(tp1,2*i);
      t2 = GET(tp1,2*i+1);
      LET(tp1,2*i)= t1*GET(ltp1,2*i)-t2*GET(ltp1,2*i+1);
      LET(tp1,2*i+1)= t2*GET(ltp1,2*i)+t1*GET(ltp1,2*i+1);
    }

 pnl_vect_free(&tbp1);
 pnl_vect_free(&ltp1);
 pnl_vect_free(&ltm1);
 pnl_vect_free(&tb1);
 pnl_vect_free(&tbm1);

 return 1;
}//end findfactor

/*////////////////////// BARRIER AND DIGITAL //////////////
    /////////////////*/

 int fastwienerhopf(int model, double mu, double qu,
    double om, int am, int upordown, int ifCall, double Spot,
    double lm1, double lp1,
            double num,double nup, double cnum,double cn
    up,
            double r, double divid,
            double T, double h, double Strike1,
            double bar,double rebate,
            double er, long int step,
            double *ptprice, double *ptdelta)
{

  PnlVect *t, *y, * v1, *vv1,*tp1,*tm1,
    *tb1, *tbp1,*tbm1,*ltp1,*ltm1, *alin1,*eom, *reb, *swit
    charrow,*al1;
  long int N1=0, Nx, kmax, j, L;

  double pp;
  double xi,xip,xim,anp,anm,sxi,nup1=1.,num1=1., abp, abm;
```

```
 double t1, t2;
 double angle;
 double a2_1=0.;
 double mut;
 double mod;
 long int i;
 long int k;
 long int Nmax, N01;
 double dt;

 double q;
 double mu1=0., str, kx;
 double Sl, Sm, Sr;
 double pricel, pricem, pricer;
 double A, B, C;
 double sigma;

if(upordown==0)
{
 if(Spot<=bar)
  {
    *ptprice = rebate;
    *ptdelta = 0.;
   return OK;
 }
 if( Spot>=0.8*bar*exp( er*log(2.) ) )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}
else
{
 if(Spot>=bar)
  {
    *ptprice = rebate;
    *ptdelta = 0.;
   return OK;
```

```
  }
 if( Spot<=bar*exp( -er*log(2.) )*1.2 )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}

str=log(Strike1/bar);

k=64;

kx=er*log(2)/h;
while(k<kx)
{
     k=k*2;
}
kmax=k;

N1=step;
Nmax=2*kmax;
NO1=kmax;          //  !!!kmax

//Memory allocation for space grid
y=pnl_vect_create_from_zero(Nmax+1);//space grid points
v1=pnl_vect_create_from_zero(Nmax+1);//prices at previous
   time step
vv1=pnl_vect_create_from_zero(2*kmax+2);
tp1=pnl_vect_create_from_zero(2*kmax+2);
alin1=pnl_vect_create_from_zero(2*kmax+2);
tbp1=pnl_vect_create_from_zero(2*kmax+2);
ltp1=pnl_vect_create_from_zero(2*kmax+2);
ltm1=pnl_vect_create_from_zero(2*kmax+2);
tb1=pnl_vect_create_from_zero(2*kmax+2);
tbm1=pnl_vect_create_from_zero(2*kmax+2);
tm1=pnl_vect_create_from_zero(2*kmax+2);
al1=pnl_vect_create_from_zero(2*kmax+2);
eom=pnl_vect_create_from_zero(Nmax+1);
```

```
 reb=pnl_vect_create_from_zero(Nmax+1);
 t=pnl_vect_create_from_zero(N1+2); //time points

 lp1+=om;
 lm1+=om;

 /*Time step*/
 dt=T/N1;
 t->array[1]=0;

  q=qu+1/dt;

 //============================ compute coefficients
if(model == 1){
 mu1=mu;
 sigma=0.;
}
if(model == 2){
 mu1=mu;
 sigma=0.;
}
if(model == 3){
 mu1=0.0;
 sigma=0.;
 if((mu>0.0)&&(mu<0.1)) {mu1=mu-0.1;}
 if((mu<=0.0)&&(mu>-0.1)) {mu1=mu+0.1;}
 q+= mu1*om;
}
if(model == 4){/*KOU*/
 mu1=0.0;//mu

 sigma=nup;
 mu+=sigma*sigma*om;
 nup=num;
 cnum*=(lm1-om)/lm1;
 cnup*=(lp1-om)/lp1;
}
findcoef(model, mu-mu1, sigma, lm1, lp1, num, nup, cnum, cn
    up, q, r,T, h, kmax, er, N1, al1);
 //=========================================
a2_1=q*dt;
```

```
 Nx=Nmax; /*number of space points*/
 i=0;
 do
   {
     alin1->array[i]=al1->array[i+1];
     i++;
   }while(i<2*kmax);

//if(model==1){
 if (nup==num)
   {
     nup1=nup/2.0;
     num1=num/2.0;
   }
 if (nup>num)
   {
     nup1=nup;//
  num1=0.0;
   }
 if (nup<num)
   {
     nup1=0.0;//
  num1=num;
   }
     abp=pow(lp1,nup1);
     abm=pow(-lm1,num1);

     ltp1->array[0]=1;
     ltm1->array[0]=1;
     xi=-M_PI/h;

     xip=pow(xi*xi+lp1*lp1,nup1/2)/abp;
     xim=pow(xi*xi+lm1*lm1,num1/2)/abm;
     anp=nup1*atan(xi/lp1);
     anm=num1*atan(xi/lm1);
     ltp1->array[1]=xip*cos(anp);
     ltm1->array[1]=xim*cos(anm);
     ltp1->array[2*kmax]=xip*sin(anp);
     ltm1->array[2*kmax]=xim*sin(anm);
     sxi=xi/kmax;
```

```
    xi=sxi;

    i=1;
    do
      {
  xip=pow(xi*xi+lp1*lp1,nup1/2)/abp;
  xim=pow(xi*xi+lm1*lm1,num1/2)/abm;
  anp=nup1*atan(xi/lp1);
  anm=num1*atan(xi/lm1);
  ltp1->array[2*i]=xip*cos(anp);
  ltp1->array[2*i+1]=xip*sin(anp);
  ltm1->array[2*i]=xim*cos(anm);
  ltm1->array[2*i+1]=xim*sin(anm);
  i++;
  xi=xi+sxi;
      }while(i<kmax);

alin1->array[1]=alin1->array[1]*(ltp1->array[1]*ltm1->arra
   y[1]-ltp1->array[2*kmax]*ltm1->array[2*kmax]);
for(i = 1; i <= kmax-1; i++)
  {
    t1 = alin1->array[2*i];
    t2 = alin1->array[2*i+1];
    alin1->array[2*i]= t1*ltm1->array[2*i]-t2*ltm1->array[
   2*i+1];
    alin1->array[2*i+1]= t2*ltm1->array[2*i]+t1*ltm1->arra
   y[2*i+1];
  }
for(i = 1; i <= kmax-1; i++)
  {
       t1 = alin1->array[2*i];
       t2 = alin1->array[2*i+1];
       alin1->array[2*i]= t1*ltp1->array[2*i]-t2*ltp1->ar
   ray[2*i+1];
       alin1->array[2*i+1]= t2*ltp1->array[2*i]+t1*ltp1->
   array[2*i+1];
  }

ltp1->array[0]=1/ltp1->array[0];
ltm1->array[0]=1/ltm1->array[0];
ltp1->array[1]=ltp1->array[1]/(ltp1->array[1]*ltp1->array[
```

```
   1]+ltp1->array[2*kmax]*ltp1->array[2*kmax]);
ltm1->array[1]=ltm1->array[1]/(ltm1->array[1]*ltm1->array[
   1]+ltm1->array[2*kmax]*ltm1->array[2*kmax]);
i=1;
do
  {   mod=ltp1->array[2*i+1]*ltp1->array[2*i+1]+ltp1->arra
   y[2*i]*ltp1->array[2*i];
    ltp1->array[2*i+1]=-ltp1->array[2*i+1]/mod;
    ltp1->array[2*i]=ltp1->array[2*i]/mod;

    mod=ltm1->array[2*i+1]*ltm1->array[2*i+1]+ltm1->array[
   2*i]*ltm1->array[2*i];
    ltm1->array[2*i+1]=-ltm1->array[2*i+1]/mod;
    ltm1->array[2*i]=ltm1->array[2*i]/mod;
    i++;
  }while(i<kmax);

tb1->array[0]=log(alin1->array[0]);
tb1->array[1]=log(alin1->array[1]);
i=1;
do
  {   mod=alin1->array[2*i+1]*alin1->array[2*i+1]+alin1->
   array[2*i]*alin1->array[2*i];
    tb1->array[2*i]=0.5*log(mod);
    if (alin1->array[2*i]==0)
      {
  if (alin1->array[2*i+1]>0)
    {tb1->array[2*i+1]=M_PI/2.0;}
  else
    {tb1->array[2*i+1]=-M_PI/2.0;}
      }
    else
      { angle=atan(alin1->array[2*i+1]/alin1->array[2*i]);
  if (alin1->array[2*i]>0) {tb1->array[2*i+1]=angle;}
  if (alin1->array[2*i]<0) {if (alin1->array[2*i+1]<0) {
   tb1->array[2*i+1]=angle-M_PI;}
    else {tb1->array[2*i+1]=angle+M_PI;}
  }
      }
    i++;
  }while(i<kmax);
```

```
i=0;
do
  {
    alin1->array[i]=al1->array[i+1];
    i++;
  }while(i<2*kmax);
fft_real(tb1, 2*kmax, 1);
i=1;
tbp1->array[0]=0;
tbm1->array[0]=0;
do
{ tbp1->array[0]=tbp1->array[0]-tb1->array[i];
  tbp1->array[i]=tb1->array[i];
  tbm1->array[i]=0;
  i++;
}while(i<kmax);
do
  { tbm1->array[i]=tb1->array[i];
    tbp1->array[i]=0;
    tbm1->array[0]=tbm1->array[0]-tb1->array[i];
    i++;
  }while(i<2*kmax);
fft_real(tbp1, 2*kmax, -1);
fft_real(tbm1, 2*kmax, -1);
tp1->array[0]=exp(tbp1->array[0]);
tp1->array[1]=exp(tbp1->array[1]);
tm1->array[0]=exp(tbm1->array[0]);
tm1->array[1]=exp(tbm1->array[1]);

i=1;
do
  {   mod=exp(tbp1->array[2*i]);
    tp1->array[2*i]=mod*cos(tbp1->array[2*i+1]);
    tp1->array[2*i+1]=mod*sin(tbp1->array[2*i+1]);
    mod=exp(tbm1->array[2*i]);
    tm1->array[2*i]=mod*cos(tbm1->array[2*i+1]);
    tm1->array[2*i+1]=mod*sin(tbm1->array[2*i+1]);
    i++;
  }while(i<kmax);

tp1->array[0]= tp1->array[0]*ltp1->array[0];
```

```
tp1->array[1]= tp1->array[1]*ltp1->array[1];

tm1->array[0]= tm1->array[0]*ltm1->array[0];
tm1->array[1]= tm1->array[1]*ltm1->array[1];

 for(i = 1; i <= kmax-1; i++)
   {
     t1 = tm1->array[2*i];
     t2 = tm1->array[2*i+1];
     tm1->array[2*i]= t1*ltm1->array[2*i]-t2*ltm1->array[2
   *i+1];
     tm1->array[2*i+1]= t2*ltm1->array[2*i]+t1*ltm1->arra
   y[2*i+1];
   }
 for(i = 1; i <= kmax-1; i++)
   {
     t1 = tp1->array[2*i];
     t2 = tp1->array[2*i+1];
     tp1->array[2*i]= t1*ltp1->array[2*i]-t2*ltp1->array[2
   *i+1];
     tp1->array[2*i+1]= t2*ltp1->array[2*i]+t1*ltp1->arra
   y[2*i+1];
   }

 /*Put Pay-off functions*/
 if(upordown==0)
   for(j=1;j<=kmax;j++)
     {
 y->array[j]=(j-N01-0.5)*h;
 vv1->array[j-1]=0;
 eom->array[j]=exp(-om*y->array[j]);
 reb->array[j]=rebate*dt*r*eom->array[j];
     }
 else
   for(j=kmax;j<=Nmax;j++)
     {
 y->array[j]=(j-N01+0.5)*h;
 vv1->array[j-1]=0;
 eom->array[j]=exp(-om*y->array[j]);
 reb->array[j]=rebate*dt*r*eom->array[j];
     }
```

```
if(upordown==0) // IF DOWN-OUT
  {
    j=kmax+1;

    if(ifCall==2) /* DIGITAL */
{
  do
    {   y->array[j]=(j-NO1-0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];

  vv1->array[j-1]=-rebate*eom->array[j];
      j++;
    }while(j<=Nmax);
}
    else if(ifCall==1)  /*  CALL */
{
  do
    {   y->array[j]=(j-NO1-0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]>=str)
  {vv1->array[j-1]=(-Strike1+bar*exp(y->array[j])-reb
  ate)*eom->array[j];}
      else
  {vv1->array[j-1]=-rebate*eom->array[j];}
      j++;
    }while(j<=Nmax);
}
    else              /* PUT */
{
  do
    {   y->array[j]=(j-NO1-0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]<=str)
  {vv1->array[j-1]=(Strike1-bar*exp(y->array[j])-rebate
  )*eom->array[j];}
      else
  {vv1->array[j-1]=-rebate*eom->array[j];}
```

```
      j++;
    }while(j<=Nmax);
}
  }// END IF DOWN-OUT
else // IF UP-OUT
  {
    j=kmax-1;

    if(ifCall==2)      /* DIGITAL */
{
  do
    {   y->array[j]=(j-N01+0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];

  {vv1->array[j-1]=-rebate*eom->array[j];}
      j--;
    }while(j>0);
}
    else if(ifCall==1)    /* CALL */
{
  do
    {   y->array[j]=(j-N01+0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]>=str)
  {vv1->array[j-1]=(-Strike1+bar*exp(y->array[j])-reb
  ate)*eom->array[j];}
      else
  {vv1->array[j-1]=-rebate*eom->array[j];}
      j--;
    }while(j>0);
}
    else                  /* PUT */
{
  do
    {   y->array[j]=(j-N01+0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]<=str)
  {vv1->array[j-1]=(Strike1-bar*exp(y->array[j])-rebate
```

```
    )*eom->array[j];}
        else
    {vv1->array[j-1]=-rebate*eom->array[j];}
        j--;
      }while(j>0);
  }

    }// END IF UP
  //======= EXCHANGE tp AND tm ARRAYS FOR UP-OUT
  if(upordown==1)
    {
      switcharrow = tp1;
      tp1 = tm1;
      tm1 = switcharrow;
    }
  //=========================================

  for(i=0;i<Nmax;i++)
    {
      tp1->array[i]=tp1->array[i]/a2_1;
      alin1->array[i]=alin1->array[i]/a2_1;
    }

  fft_real(vv1, 2*kmax, -1);

  vv1->array[0]= vv1->array[0]*tm1->array[0];
  vv1->array[1]= vv1->array[1]*tm1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tm1->array[2*i]-t2*tm1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tm1->array[2*i]+t1*tm1->array[2
    *i+1];
    }
  fft_real(vv1, 2*kmax, 1);

 /*Main Loop on time grid*/
```

```
for(L=2; L<=N1; L++)
  {
    t->array[L]=t->array[L-1]+dt;
    mut=t->array[L]*mu1;

    if(upordown==0)// IF DOWN-OUT
         !!
{
  i=0;
  do
    {
      vv1->array[i]=0;
      i++;
    }while(y->array[i+1]<=mut);

  do
    {
      vv1->array[i]=vv1->array[i]-reb->array[i+1];
      i++;
    }while(i<Nmax);
}
    else  //IF UP-OUT
{
  i=Nmax;//-1;
  do
    {
      vv1->array[i]=0;
      i--;
    }while(y->array[i+1]>=mut);

  do
    {
      vv1->array[i]=vv1->array[i]-reb->array[i+1];
      i--;
    }while(i>=0);
}

    fft_real(vv1, 2*kmax, -1);
    vv1->array[0]= vv1->array[0]*alin1->array[0];
    vv1->array[1]= vv1->array[1]*alin1->array[1];
    for(i = 1; i <= kmax-1; i++)
```

```
{
  t1 = vv1->array[2*i];
  t2 = vv1->array[2*i+1];
  vv1->array[2*i]= t1*alin1->array[2*i]-t2*alin1->array[
  2*i+1];
  vv1->array[2*i+1]= t2*alin1->array[2*i]+t1*alin1->arra
  y[2*i+1];
}
    fft_real(vv1, 2*kmax, 1);
  }// END MAIN LOOP

t->array[N1+1]=t->array[N1]+dt;
mut=mu1*t->array[N1+1];

if(upordown==0)// IF DOWN-OUT                          !!
  {
    i=0;
    do
  {
  vv1->array[i]=0;
  i++;
  }while(y->array[i+1]<=mut);

    do
{
  vv1->array[i]=vv1->array[i]-reb->array[i+1];
  i++;
}while(i<Nmax);
  }
else  //IF UP-OUT
  {
    i=Nmax;//-1;
    do
  {
  vv1->array[i]=0;
  i--;
  }while(y->array[i+1]>=mut);

    do
{
  vv1->array[i]=vv1->array[i]-reb->array[i+1];
```

```
  i--;
}while(i>=0);
  }
fft_real(vv1, 2*kmax, -1);
vv1->array[0]= vv1->array[0]*tp1->array[0];
vv1->array[1]= vv1->array[1]*tp1->array[1];
for(i = 1; i <= kmax-1; i++)
  {
    t1 = vv1->array[2*i];
    t2 = vv1->array[2*i+1];
    vv1->array[2*i]= t1*tp1->array[2*i]-t2*tp1->array[2*
  i+1];
    vv1->array[2*i+1]= t2*tp1->array[2*i]+t1*tp1->array[2
  *i+1];
  }
fft_real(vv1,2*kmax, 1);
for(i=1;i<Nmax+1;i++)
          // !!
{
     v1->array[i]=vv1->array[i-1]/eom->array[i]+rebate;
y->array[i]=y->array[i]-mut;
}

pp=log(Spot/bar);
i=1;
while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
i=i-1;
//Price, quadratic interpolation
Sl = bar*exp(y->array[i-1]);
Sm = bar*exp(y->array[i]);
Sr = bar*exp(y->array[i+1]);
// S0 is between Sm and Sr
pricel = v1->array[i-1];
pricem = v1->array[i];
pricer = v1->array[i+1];
//quadratic interpolation
A = pricel;
B = (pricem-pricel)/(Sm-Sl);
C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
//Price
*ptprice = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
```

```
  if(*ptprice<0) {*ptprice=0.0;}
  //Delta
  *ptdelta = B + C*(2*Spot-Sl-Sm);

  //=====================================PREMIA_NULLTYPE===
    ===
  /*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&v1);
  pnl_vect_free(&vv1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tbp1);
  pnl_vect_free(&ltp1);
  pnl_vect_free(&ltm1);
  pnl_vect_free(&tb1);
  pnl_vect_free(&tbm1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&reb);
  pnl_vect_free(&t);
  return OK;
}

// ////////////////////// AMERICAN ////////////////////////
int fastwienerhopfamerican(int model, double mu, double qu,
     double om,
int ifCall, double Spot, double lm1, double lp1,
    double num,double nup, double cnum,double cnup,
    double r, double divid,
    double T, double h, double Strike1,
    double er, long int step,
    double *ptprice, double *ptdelta)
{
 PnlVect *t, *y, * v1, *vv1,*tp1,*tm1,
    *tb1, *tbp1,*tbm1,*ltp1,*ltm1, *alin1,*eom, *reb, *swit
    charrow,*al1, *dividterm;
 long int N1=0, Nx, kmax, j, L;

 double pp;
```

```
 double xi,xip,xim,anp,anm,sxi,nup1=1.,num1=1., abp, abm;
 double t1, t2;
 double angle;

 double a2_1=0.;
double mut, emut;
double mod;
long int i;
 long int k;
 long int Nmax, NO1;
 double dt;
 double q;
 double mu1=0., kx;
double Sl, Sm, Sr;
double pricel, pricem, pricer;
double A, B, C;
double sigma;

 //divid=0.0;

 if(ifCall==0) //IF PUT
 {
  if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
   {
     *ptprice = 0.;
     *ptdelta = 0.;
     printf("Spot is out of range. Increase scale para
    meter {n");
     return OK;
   }
  if( Spot<=1.2*Strike1*exp( -er*log(2.) ) )
   {
     *ptprice = Strike1 - Spot;
     *ptdelta = -1.0;
     printf("Spot is out of range. Increase scale para
    meter {n");
     return OK;
   }
 }
 else
 {
```

```
 if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
  {
    *ptprice = - Strike1 + Spot;
    *ptdelta = 1.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
 if( Spot<=Strike1*exp( -er*log(2.) )*1.2 )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}

k=64;

kx=er*log(2)/h;
while(k<kx)
{
     k=k*2;
}
kmax=k;

N1=step;
Nmax=2*kmax;
NO1=kmax;

//Memory allocation for space grid
y=pnl_vect_create_from_zero(Nmax+1);//space grid points
v1=pnl_vect_create_from_zero(Nmax+1);//prices at previous
   time step
vv1=pnl_vect_create_from_zero(2*kmax+2);
tp1=pnl_vect_create_from_zero(2*kmax+2);
alin1=pnl_vect_create_from_zero(2*kmax+2);
tbp1=pnl_vect_create_from_zero(2*kmax+2);
ltp1=pnl_vect_create_from_zero(2*kmax+2);
ltm1=pnl_vect_create_from_zero(2*kmax+2);
```

```
 tb1=pnl_vect_create_from_zero(2*kmax+2);
 tbm1=pnl_vect_create_from_zero(2*kmax+2);
 tm1=pnl_vect_create_from_zero(2*kmax+2);
 al1=pnl_vect_create_from_zero(2*kmax+2);
 eom=pnl_vect_create_from_zero(Nmax+1);
 reb=pnl_vect_create_from_zero(Nmax+1);
 dividterm=pnl_vect_create_from_zero(Nmax+1);
 t=pnl_vect_create_from_zero(N1+2); //time points

lp1+=om;
 lm1+=om;

 /*Time step*/
dt=T/N1;
 t->array[1]=0;

q=qu+1/dt;

//============================ compute coefficients
if(model == 1){/*TSL*/
 mu1=mu;
 sigma=0.;
}
if(model == 2){/*NIG*/
 mu1=mu;
 sigma=0.;
}
if(model == 3){/*VG*/
 mu1=0.0;
 sigma=0.;
 if((mu>0.0)&&(mu<0.1)) {mu1=mu-0.1;}
 if((mu<=0.0)&&(mu>-0.1)) {mu1=mu+0.1;}
 q+= mu1*om;
}
if(model == 4){/*KOU*/
 mu1=0.0;//mu

 sigma=nup;
 mu+=sigma*sigma*om;
 nup=num;
 cnum*=(lm1-om)/lm1;
```

```
 cnup*=(lp1-om)/lp1;
}
findcoef(model, mu-mu1, sigma, lm1, lp1, num, nup, cnum, cn
    up, q, r,T, h, kmax, er, N1, al1);
 //=========================================
a2_1=q*dt;

 Nx=Nmax; /*number of space points*/

 i=0;
 do
 {
  alin1->array[i]=al1->array[i+1];
  i++;
 }while(i<2*kmax);



if (nup==num)
   {
     nup1=nup/2.0;
     num1=num/2.0;
   }
 if (nup>num)
   {
     nup1=nup;// /2;
  num1=0.0;
   }
 if (nup<num)
   {
     nup1=0.0;// /2;
  num1=num;
   }
     abp=pow(lp1,nup1);
     abm=pow(-lm1,num1);

     ltp1->array[0]=1;
     ltm1->array[0]=1;
     xi=-M_PI/h;

     xip=pow(xi*xi+lp1*lp1,nup1/2)/abp;
```

```
    xim=pow(xi*xi+lm1*lm1,num1/2)/abm;
    anp=nup1*atan(xi/lp1);
    anm=num1*atan(xi/lm1);
    ltp1->array[1]=xip*cos(anp);
    ltm1->array[1]=xim*cos(anm);
    ltp1->array[2*kmax]=xip*sin(anp);
    ltm1->array[2*kmax]=xim*sin(anm);
    sxi=xi/kmax;
    xi=sxi;

   i=1;
do
{
 xip=pow(xi*xi+lp1*lp1,nup1/2)/abp;
 xim=pow(xi*xi+lm1*lm1,num1/2)/abm;
 anp=nup1*atan(xi/lp1);
 anm=num1*atan(xi/lm1);
 ltp1->array[2*i]=xip*cos(anp);
 ltp1->array[2*i+1]=xip*sin(anp);
 ltm1->array[2*i]=xim*cos(anm);
 ltm1->array[2*i+1]=xim*sin(anm);
 i++;
 xi=xi+sxi;
}while(i<kmax);

alin1->array[1]=alin1->array[1]*(ltp1->array[1]*ltm1->arra
   y[1]-ltp1->array[2*kmax]*ltm1->array[2*kmax]);
for(i = 1; i <= kmax-1; i++)
  {
    t1 = alin1->array[2*i];
    t2 = alin1->array[2*i+1];
    alin1->array[2*i]= t1*ltm1->array[2*i]-t2*ltm1->array[
   2*i+1];
    alin1->array[2*i+1]= t2*ltm1->array[2*i]+t1*ltm1->arra
   y[2*i+1];
  }
for(i = 1; i <= kmax-1; i++)
  {
       t1 = alin1->array[2*i];
       t2 = alin1->array[2*i+1];
       alin1->array[2*i]= t1*ltp1->array[2*i]-t2*ltp1->ar
```

```
    ray[2*i+1];
        alin1->array[2*i+1]= t2*ltp1->array[2*i]+t1*ltp1->
    array[2*i+1];
   }

 ltp1->array[0]=1/ltp1->array[0];
 ltm1->array[0]=1/ltm1->array[0];
 ltp1->array[1]=ltp1->array[1]/(ltp1->array[1]*ltp1->array[
    1]+ltp1->array[2*kmax]*ltp1->array[2*kmax]);
 ltm1->array[1]=ltm1->array[1]/(ltm1->array[1]*ltm1->array[
    1]+ltm1->array[2*kmax]*ltm1->array[2*kmax]);

i=1;
 do
   {   mod=ltp1->array[2*i+1]*ltp1->array[2*i+1]+ltp1->arra
    y[2*i]*ltp1->array[2*i];
     ltp1->array[2*i+1]=-ltp1->array[2*i+1]/mod;
     ltp1->array[2*i]=ltp1->array[2*i]/mod;

     mod=ltm1->array[2*i+1]*ltm1->array[2*i+1]+ltm1->array[
    2*i]*ltm1->array[2*i];
     ltm1->array[2*i+1]=-ltm1->array[2*i+1]/mod;
     ltm1->array[2*i]=ltm1->array[2*i]/mod;
     i++;
   }while(i<kmax);

 tb1->array[0]=log(alin1->array[0]);
 tb1->array[1]=log(alin1->array[1]);

 i=1;
 do
   {   mod=alin1->array[2*i+1]*alin1->array[2*i+1]+alin1->
    array[2*i]*alin1->array[2*i];
     tb1->array[2*i]=0.5*log(mod);
     if (alin1->array[2*i]==0)
       {
   if (alin1->array[2*i+1]>0)
     {tb1->array[2*i+1]=M_PI/2.0;}
   else
     {tb1->array[2*i+1]=-M_PI/2.0;}
       }
```

```
    else
      { angle=atan(alin1->array[2*i+1]/alin1->array[2*i]);
  if (alin1->array[2*i]>0) {tb1->array[2*i+1]=angle;}
  if (alin1->array[2*i]<0) {if (alin1->array[2*i+1]<0) {
   tb1->array[2*i+1]=angle-M_PI;}
    else {tb1->array[2*i+1]=angle+M_PI;}
  }
      }
    i++;
  }while(i<kmax);

i=0;
do
{
 alin1->array[i]=al1->array[i+1];
 i++;
}while(i<2*kmax);

fft_real(tb1, 2*kmax, 1);

 i=1;
tbp1->array[0]=0;
tbm1->array[0]=0;
do
{ tbp1->array[0]=tbp1->array[0]-tb1->array[i];
  tbp1->array[i]=tb1->array[i];
  tbm1->array[i]=0;
  i++;
}while(i<kmax);
do
  { tbm1->array[i]=tb1->array[i];
    tbp1->array[i]=0;
    tbm1->array[0]=tbm1->array[0]-tb1->array[i];
    i++;
  }while(i<2*kmax);

fft_real(tbp1, 2*kmax, -1);
fft_real(tbm1, 2*kmax, -1);

tp1->array[0]=exp(tbp1->array[0]);
tp1->array[1]=exp(tbp1->array[1]);
```

```
tm1->array[0]=exp(tbm1->array[0]);
tm1->array[1]=exp(tbm1->array[1]);


i=1;
do
  {   mod=exp(tbp1->array[2*i]);
    tp1->array[2*i]=mod*cos(tbp1->array[2*i+1]);
    tp1->array[2*i+1]=mod*sin(tbp1->array[2*i+1]);
    mod=exp(tbm1->array[2*i]);
    tm1->array[2*i]=mod*cos(tbm1->array[2*i+1]);
    tm1->array[2*i+1]=mod*sin(tbm1->array[2*i+1]);
    i++;
  }while(i<kmax);

tp1->array[0]= tp1->array[0]*ltp1->array[0];
tp1->array[1]= tp1->array[1]*ltp1->array[1];

tm1->array[0]= tm1->array[0]*ltm1->array[0];
tm1->array[1]= tm1->array[1]*ltm1->array[1];

 for(i = 1; i <= kmax-1; i++)
   {
     t1 = tm1->array[2*i];
     t2 = tm1->array[2*i+1];
     tm1->array[2*i]= t1*ltm1->array[2*i]-t2*ltm1->array[2
   *i+1];
     tm1->array[2*i+1]= t2*ltm1->array[2*i]+t1*ltm1->arra
   y[2*i+1];
   }
 for(i = 1; i <= kmax-1; i++)
   {
     t1 = tp1->array[2*i];
     t2 = tp1->array[2*i+1];
     tp1->array[2*i]= t1*ltp1->array[2*i]-t2*ltp1->array[2
   *i+1];
     tp1->array[2*i+1]= t2*ltp1->array[2*i]+t1*ltp1->arra
   y[2*i+1];
   }
```

```
if(ifCall==0)
{
for(j=1;j<=Nmax;j++)
 {
  y->array[j]=(j-N01-0.5)*h;
  vv1->array[j-1]=0;
  eom->array[j]=exp(-om*y->array[j]);
  reb->array[j]=Strike1*dt*r*eom->array[j];
  dividterm->array[j]=-Strike1*divid*dt*eom->array[j]*exp(
    y->array[j]);
 }
}
else
{
for(j=1;j<=Nmax;j++)
 {
  y->array[j]=(j-N01+0.5)*h;
  vv1->array[j-1]=0;
  eom->array[j]=exp(-om*y->array[j]);
  reb->array[j]=-Strike1*dt*r*eom->array[j];
  dividterm->array[j]=Strike1*divid*dt*eom->array[j]*exp(y-
    >array[j]);
 }
}

if(ifCall==0)
{   j=kmax+1;
    do
   {   vv1->array[j-1]=(-Strike1+Strike1*exp(y->array[j]))
    *eom->array[j];
    j++;
   }while(j<=Nmax);
}
else
{   j=kmax-1;
  do
   {   vv1->array[j-1]=(Strike1-Strike1*exp(y->array[j]))*
    eom->array[j];
    j--;
   }while(j>0);
}
```

```
 //======= EXCHANGE tp AND tm ARRAYS FOR UP-OUT
 if(ifCall==1)
 {
 switcharrow = tp1;
 tp1 = tm1;
 tm1 = switcharrow;
 }
 //=========================================


 for(i=0;i<Nmax;i++)
 {
     tp1->array[i]=tp1->array[i]/a2_1;
     alin1->array[i]=alin1->array[i]/a2_1;
   }

 fft_real(vv1, 2*kmax, -1);

 vv1->array[0]= vv1->array[0]*tm1->array[0];
 vv1->array[1]= vv1->array[1]*tm1->array[1];

 for(i = 1; i <= kmax-1; i++)
   {
     t1 = vv1->array[2*i];
     t2 = vv1->array[2*i+1];
     vv1->array[2*i]= t1*tm1->array[2*i]-t2*tm1->array[2*
   i+1];
     vv1->array[2*i+1]= t2*tm1->array[2*i]+t1*tm1->array[2
   *i+1];
   }
 fft_real(vv1, 2*kmax, 1);

//======================*Main Loop on time grid*=========
   ===============/

for(L=2; L<=N1; L++)
{
 t->array[L]=t->array[L-1]+dt;

 mut=t->array[L]*mu1;
```

```
  emut=exp(-mut);

  i=Nmax-1;
    do
    {  vv1->array[i]=vv1->array[i]-reb->array[i+1]-divid
    term->array[i+1]*emut;
      if (vv1->array[i]<0)
      {vv1->array[i]=0;}
      i--;
    }while(i>=0);


  fft_real(vv1, 2*kmax, -1);

  vv1->array[0]= vv1->array[0]*alin1->array[0];
  vv1->array[1]= vv1->array[1]*alin1->array[1];

  for(i = 1; i <= kmax-1; i++)
  {
    t1 = vv1->array[2*i];
    t2 = vv1->array[2*i+1];
    vv1->array[2*i]= t1*alin1->array[2*i]-t2*alin1->array[
    2*i+1];
    vv1->array[2*i+1]= t2*alin1->array[2*i]+t1*alin1->arra
    y[2*i+1];
  }

      fft_real(vv1, 2*kmax, 1);
 }// ======================================= PREMIA_NULLT
    YPE MAIN LOOP ==================

  t->array[N1+1]=t->array[N1]+dt;
  mut=mu1*t->array[N1+1];
  emut=exp(-mut);

i=Nmax-1;
    do
    {  vv1->array[i]=(vv1->array[i]-reb->array[i+1])-div
    idterm->array[i+1]*emut;
      if (vv1->array[i]<0)
      {vv1->array[i]=0;}
```

```
      i--;
    }while(i>=0);

  fft_real(vv1, 2*kmax, -1);
  vv1->array[0]= vv1->array[0]*tp1->array[0];
  vv1->array[1]= vv1->array[1]*tp1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tp1->array[2*i]-t2*tp1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tp1->array[2*i]+t1*tp1->array[2
    *i+1];
    }
  fft_real(vv1,2*kmax, 1);

if(ifCall==0)
{
  for(i=1;i<Nmax+1;i++)
  {    y->array[i]=y->array[i]-mut;
       v1->array[i]=vv1->array[i-1]/eom->array[i]+Strike1-
    Strike1*exp(y->array[i]);
  }
}
else
{
  for(i=1;i<Nmax+1;i++)
  {    y->array[i]=y->array[i]-mut;
       v1->array[i]=vv1->array[i-1]/eom->array[i]-Strike1+
    Strike1*exp(y->array[i]);
  }
}

 pp=log(Spot/Strike1);

 i=1;

 while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
i=i-1;
```

```
  //Price, quadratic interpolation
  Sl = Strike1*exp(y->array[i-1]);
  Sm = Strike1*exp(y->array[i]);
  Sr = Strike1*exp(y->array[i+1]);

  // S0 is between Sm and Sr
  pricel = v1->array[i-1];
  pricem = v1->array[i];
  pricer = v1->array[i+1];

  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);

    //Price
  *ptprice = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  if(*ptprice<0) {*ptprice=0.0;}
    //Delta
  *ptdelta = B + C*(2*Spot-Sl-Sm);

  //====================================PREMIA_NULLTYPE===
    ===
/*Memory desallocation*/
pnl_vect_free(&y);
  pnl_vect_free(&v1);
  pnl_vect_free(&vv1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tbp1);
  pnl_vect_free(&ltp1);
  pnl_vect_free(&ltm1);
  pnl_vect_free(&tb1);
  pnl_vect_free(&tbm1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&reb);
  pnl_vect_free(&t);
  pnl_vect_free(&dividterm);
```

```
return OK;
}


//Code for Backward Fourier
//=========================================================
    ========
static int findcoef_bi(int model, double mu, double sigma,
    double lm1, double lp1,
        double num, double nup,
        double cnum,double cnup, double q, double r1,
        double T, double h, long int kmax,
        long int Nt,
        PnlVect * bl1)

{
  //PnlVect *bl1;
  long int i;
  double mod;
  //PnlVect *alin1;
  double xi,xip,xim,anp,anm,sxi,nup1,num1;
  double lpm1;

  double del1;
  double sg2;
  double cpl;     //  !!!!!!
  double cml;

  /*Memory allocation for space grid*/

 // bl1=pnl_vect_create(2*kmax+1);

 del1=-T/Nt;
  ////////////////////////// this part depends on model
if(model==1/*TSL*/)
{

  lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  nup1=nup/2;
  num1=num/2;
```

```
  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=pow(xi*xi+lp1*lp1,nup1);
  xim=pow(xi*xi+lm1*lm1,num1);
  anp=nup*atan(xi/lp1);
  anm=num*atan(xi/lm1);
  bl1->array[1]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm);
  bl1->array[2*kmax]=-cnup*xip*sin(anp)-cnum*xim*sin(anm)-
    mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
    {
      xip=pow(xi*xi+lp1*lp1,nup1);
      xim=pow(xi*xi+lm1*lm1,num1);
      anp=nup*atan(xi/lp1);
      anm=num*atan(xi/lm1);
      bl1->array[2*i]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(
    anm);
      bl1->array[2*i+1]=-cnup*xip*sin(anp)-cnum*xim*sin(an
    m)-mu*xi;
      i++;
      xi=xi+sxi;
 }while(i<kmax);
}// END TSL
if(model==2/*NIG*/)
{
       lpm1=q-sqrt(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=0.5*atan(xim/xip);
  bl1->array[1]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos(
    anp);
  bl1->array[2*kmax]=cnup*pow(xip*xip+xim*xim, 0.25)*sin(
    anp);
  sxi=xi/kmax;
```

```
  xi=sxi;

       i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=0.5*atan(xim/xip);
    bl1->array[2*i]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*
    cos(anp);
    bl1->array[2*i+1]=cnup*pow(xip*xip+xim*xim, 0.25)*si
    n(anp);
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END NIG
if(model == 3/*VGP*/)
{
       lpm1=q-log(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=atan(xim/xip);
  bl1->array[1]=lpm1+cnup*log(xip*xip+xim*xim)/2;
  bl1->array[2*kmax]=cnup*anp-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

    i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=atan(xim/xip);
    bl1->array[2*i]=lpm1+cnup*log(xip*xip+xim*xim)/2;
    bl1->array[2*i+1]=cnup*anp-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);
```

```
}// END VGP
if(model == 4/*KOU*/)
{
  // nup==sigma
     sg2=sigma*sigma/2;
  cpl=cnup*lp1;      //  !!!!!!
  cml=cnum*lm1;

  //lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi+lp1*lp1;
  xim=xi*xi+lm1*lm1;
  bl1->array[1]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
  bl1->array[2*kmax]=xi*(cpl/xip+cml/xim)-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi+lp1*lp1;
    xim=xi*xi+lm1*lm1;
    bl1->array[2*i]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
    bl1->array[2*i+1]=xi*(cpl/xip+cml/xim)-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END KOU
  ////////////////////////////////////////////////////
  bl1->array[0]=exp(del1*bl1->array[0]);
  bl1->array[1]=exp(del1*bl1->array[1]);
  i=1;
  do
    { bl1->array[2*i]=exp(del1*bl1->array[2*i]);
    mod=del1*bl1->array[2*i+1];
    bl1->array[2*i+1]=bl1->array[2*i]*sin(mod);
    bl1->array[2*i]=bl1->array[2*i]*cos(mod);
    i++;
```

```
    }while(i<kmax);

  return 1;
}

////////////////////////// Code for VAR coef ///////////////
    ////////////
//=========================================================
    ========
static int findcoef_var(int model, double mu, double sigma,
     double lm1, double lp1,
        double num, double nup,
        double cnum,double cnup, double q,
        double T, double h, long int kmax,
        long int Nt,
        PnlVect * bl1)

{
  long int i;
  double mod;
  double xi,xip,xim,anp,anm,sxi,nup1,num1;
  double lpm1;

  double del1;
  double sg2;
  double cpl;
  double cml;

  del1=-T/Nt;
  ////////////////////////// this part depends on model
if(model==1/*TSL*/)
{

  lpm1=q+pow(lp1,nup)*cnup+pow(-lm1,num)*cnum;

  nup1=nup/2;
  num1=num/2;
  bl1->array[0]=q;
  xi=M_PI/h;
  xip=pow(xi*xi+lp1*lp1,nup1);
  xim=pow(xi*xi+lm1*lm1,num1);
```

```
  anp=nup*atan(xi/lp1);
  anm=num*atan(xi/lm1);
  bl1->array[1]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(anm);
  bl1->array[2*kmax]=-cnup*xip*sin(anp)-cnum*xim*sin(anm)-
    mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
    {
      xip=pow(xi*xi+lp1*lp1,nup1);
      xim=pow(xi*xi+lm1*lm1,num1);
      anp=nup*atan(xi/lp1);
      anm=num*atan(xi/lm1);
      bl1->array[2*i]=lpm1-cnup*xip*cos(anp)-cnum*xim*cos(
    anm);
      bl1->array[2*i+1]=-cnup*xip*sin(anp)-cnum*xim*sin(an
    m)-mu*xi;
      i++;
      xi=xi+sxi;
 }while(i<kmax);
}// END TSL
if(model==2/*NIG*/)
{
       lpm1=q-sqrt(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=0.5*atan(xim/xip);
  bl1->array[1]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*cos(
    anp);
  bl1->array[2*kmax]=cnup*pow(xip*xip+xim*xim, 0.25)*sin(
    anp);
  sxi=xi/kmax;
  xi=sxi;

       i=1;
  do
```

```
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=0.5*atan(xim/xip);
    bl1->array[2*i]=lpm1+cnum*pow(xip*xip+xim*xim, 0.25)*
    cos(anp);
    bl1->array[2*i+1]=cnup*pow(xip*xip+xim*xim, 0.25)*si
    n(anp);
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END NIG
if(model == 3/*VGP*/)
{
       lpm1=q-log(-lp1*lm1)*cnup;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi-lp1*lm1;
  xim=-(lp1+lm1)*xi;
  anp=atan(xim/xip);
  bl1->array[1]=lpm1+cnup*log(xip*xip+xim*xim)/2;
  bl1->array[2*kmax]=cnup*anp-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

    i=1;
  do
  {
    xip=xi*xi-lp1*lm1;
    xim=-(lp1+lm1)*xi;
    anp=atan(xim/xip);
    bl1->array[2*i]=lpm1+cnup*log(xip*xip+xim*xim)/2;
    bl1->array[2*i+1]=cnup*anp-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);
}// END VGP
if(model == 4/*KOU*/)
{
```

```
    sg2=sigma*sigma/2;
  cpl=cnup*lp1;
  cml=cnum*lm1;

  bl1->array[0]=q;
  xi=-M_PI/h;
  xip=xi*xi+lp1*lp1;
  xim=xi*xi+lm1*lm1;
  bl1->array[1]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
  bl1->array[2*kmax]=xi*(cpl/xip+cml/xim)-mu*xi;
  sxi=xi/kmax;
  xi=sxi;

  i=1;
  do
  {
    xip=xi*xi+lp1*lp1;
    xim=xi*xi+lm1*lm1;
    bl1->array[2*i]=q+(sg2+cnup/xip+cnum/xim)*xi*xi;
    bl1->array[2*i+1]=xi*(cpl/xip+cml/xim)-mu*xi;
    i++;
    xi=xi+sxi;
  }while(i<kmax);

}//   END KOU
  ////////////////////////////////////////////////////
  bl1->array[0]=exp(del1*bl1->array[0]);
  bl1->array[1]=exp(del1*bl1->array[1]);
  i=1;
  do
    { bl1->array[2*i]=exp(del1*bl1->array[2*i]);
    mod=del1*bl1->array[2*i+1];
    bl1->array[2*i+1]=bl1->array[2*i]*sin(mod);
    bl1->array[2*i]=bl1->array[2*i]*cos(mod);
    i++;
    }while(i<kmax);

  return 1;
}
static int expectation(long int kmax,PnlVect * vv1, PnlVec
    t * bl1)
```

```
{
  long int i;
  double t1,t2;


  vv1->array[0]= vv1->array[0]*bl1->array[0];
  vv1->array[1]= vv1->array[1]*bl1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*bl1->array[2*i]-t2*bl1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*bl1->array[2*i]+t1*bl1->array[2
    *i+1];
    }
  fft_real(vv1, 2*kmax, 1);
  return 1;
}



/*////////////////////// BARRIER AND DIGITAL Backward induc
    tion////////////////////////////////*/

 int bi_barr(double mu, double qu, double om, int upordown,
     int ifCall, double Spot, double lm1, double lp1,
            double num,double nup, double cnum,double cn
    up,
            double r, double divid,
            double T, double h, double Strike1,
            double bar,double rebate,
            double er, long int step,
            double *ptprice, double *ptdelta)
{

  PnlVect *y, * v1, *vv1,*eom, *reb, *al1;
  long int N1=0, Nx, kmax, j, L;
```

```
 double pp;
 long int i;
 long int k;
 long int Nmax, N01;
 double dt;

 double q;
 double str, kx;
 double Sl, Sm, Sr;
 double pricel, pricem, pricer;
 double A, B, C;


if(upordown==0)
{
 if(Spot<=bar)
  {
    *ptprice = rebate;
    *ptdelta = 0.;
   return OK;
 }
 if( Spot>=0.8*bar*exp( er*log(2.) ) )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}
else
{
 if(Spot>=bar)
  {
    *ptprice = rebate;
    *ptdelta = 0.;
   return OK;
  }
 if( Spot<=bar*exp( -er*log(2.) )*1.2 )
  {
    *ptprice = 0.;
```

```
     *ptdelta = 0.;
     printf("Spot is out of range. Increase scale para
    meter {n");
     return OK;
   }
 }

 str=log(Strike1/bar);

 k=64;

 kx=er*log(2)/h;
 while(k<kx)
 {
      k=k*2;
 }
 kmax=k;

 N1=step;
 Nmax=2*kmax;
 NO1=kmax;          //  !!!kmax

 //Memory allocation for space grid
 y=pnl_vect_create_from_zero(Nmax+1);//space grid points
 v1=pnl_vect_create_from_zero(Nmax+1);//prices at previous
    time step
 vv1=pnl_vect_create_from_zero(2*kmax+2);
 al1=pnl_vect_create_from_zero(2*kmax+2);
 eom=pnl_vect_create_from_zero(Nmax+1);
 reb=pnl_vect_create_from_zero(Nmax+1);
 //t=pnl_vect_create_from_zero(N1+2); //time points

 lp1+=om;
 lm1+=om;

 /*Time step*/
 dt=T/N1;
// t->array[1]=0;

  q=qu;
```

```
//============================ compute coefficients
findcoef_bi(3, mu, 0., lm1, lp1, num, nup, cnum, cnup, q,
   r,T, h, kmax, N1, al1);
//=========================================

Nx=Nmax; /*number of space points*/



 /*Put Pay-off functions*/
 if(upordown==0)
   for(j=1;j<=kmax;j++)
     {
 y->array[j]=(j-N01-0.5)*h;
 vv1->array[j-1]=0;
 eom->array[j]=exp(-om*y->array[j]);
 reb->array[j]=rebate*dt*r*eom->array[j];
     }
 else
   for(j=kmax;j<=Nmax;j++)
     {
 y->array[j]=(j-N01+0.5)*h;
 vv1->array[j-1]=0;
 eom->array[j]=exp(-om*y->array[j]);
 reb->array[j]=rebate*dt*r*eom->array[j];
     }

 if(upordown==0) // IF DOWN-OUT
   {
     j=kmax+1;

     if(ifCall==2) /* DIGITAL */
 {
   do
     {   y->array[j]=(j-N01-0.5)*h;
       eom->array[j]=exp(-om*y->array[j]);
       reb->array[j]=rebate*dt*r*eom->array[j];

   vv1->array[j-1]=-rebate*eom->array[j];
       j++;
     }while(j<=Nmax);
```

```
}
    else if(ifCall==1)  /*  CALL */
{
  do
    {   y->array[j]=(j-N01-0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]>=str)
  {vv1->array[j-1]=(-Strike1+bar*exp(y->array[j])-reb
  ate)*eom->array[j];}
      else
  {vv1->array[j-1]=-rebate*eom->array[j];}
      j++;
    }while(j<=Nmax);
}
    else              /* PUT */
{
  do
    {   y->array[j]=(j-N01-0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
      if(y->array[j]<=str)
  {vv1->array[j-1]=(Strike1-bar*exp(y->array[j])-rebate
  )*eom->array[j];}
      else
  {vv1->array[j-1]=-rebate*eom->array[j];}
      j++;
    }while(j<=Nmax);
}
  }// END IF DOWN-OUT
else // IF UP-OUT
  {
    j=kmax-1;

    if(ifCall==2)      /* DIGITAL */
{
  do
    {   y->array[j]=(j-N01+0.5)*h;
      eom->array[j]=exp(-om*y->array[j]);
      reb->array[j]=rebate*dt*r*eom->array[j];
```

```
    {vv1->array[j-1]=-rebate*eom->array[j];}
        j--;
      }while(j>0);
  }
      else if(ifCall==1)    /* CALL */
  {
    do
      {   y->array[j]=(j-N01+0.5)*h;
        eom->array[j]=exp(-om*y->array[j]);
        reb->array[j]=rebate*dt*r*eom->array[j];
        if(y->array[j]>=str)
    {vv1->array[j-1]=(-Strike1+bar*exp(y->array[j])-reb
    ate)*eom->array[j];}
        else
    {vv1->array[j-1]=-rebate*eom->array[j];}
        j--;
      }while(j>0);
  }
      else                  /* PUT */
  {
    do
      {   y->array[j]=(j-N01+0.5)*h;
        eom->array[j]=exp(-om*y->array[j]);
        reb->array[j]=rebate*dt*r*eom->array[j];
        if(y->array[j]<=str)
    {vv1->array[j-1]=(Strike1-bar*exp(y->array[j])-rebate
    )*eom->array[j];}
        else
    {vv1->array[j-1]=-rebate*eom->array[j];}
        j--;
      }while(j>0);
  }

    }// END IF UP


 /*Main Loop on time grid*/

  for(L=1; L<=N1; L++)
    { fft_real(vv1, 2*kmax, -1);
      expectation(kmax,vv1,al1);
```

```
    if(upordown==0)// IF DOWN-OUT
         !!
{
  i=0;
  do
    {
      vv1->array[i]=0;
      i++;
    }while(y->array[i+1]<=0);

  do
    {
      vv1->array[i]=vv1->array[i]-reb->array[i+1];
      i++;
    }while(i<Nmax);
}
    else  //IF UP-OUT
{
  i=Nmax;//-1;
  do
    {
      vv1->array[i]=0;
      i--;
    }while(y->array[i+1]>=0);

  do
    {
      vv1->array[i]=vv1->array[i]-reb->array[i+1];
      i--;
    }while(i>=0);
}


  }// END MAIN LOOP

for(i=1;i<Nmax+1;i++)
          // !!
{
  v1->array[i]=vv1->array[i-1]/eom->array[i]+rebate;
}
```

```
  pp=log(Spot/bar);
  i=1;
  while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
  //Price, quadratic interpolation
  Sl = bar*exp(y->array[i-1]);
  Sm = bar*exp(y->array[i]);
  Sr = bar*exp(y->array[i+1]);
  // S0 is between Sm and Sr
  pricel = v1->array[i-1];
  pricem = v1->array[i];
  pricer = v1->array[i+1];
  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
  //Price
  *ptprice = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  if(*ptprice<0) {*ptprice=0.0;}
  //Delta
  *ptdelta = B + C*(2*Spot-Sl-Sm);

  //=====================================END======
  /*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&v1);
  pnl_vect_free(&vv1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&reb);
  return OK;
}

// ////////////////////// AMERICAN Backward induction///////
    ///////////////////

int bi_american(double mu, double qu, double om,
int ifCall, double Spot, double lm1, double lp1,
    double num,double nup, double cnum,double cnup,
    double r, double divid,
    double T, double h, double Strike1,
    double er, long int step,
```

```
    double *ptprice, double *ptdelta)
{
 PnlVect  *y, * v1, *vv1,*eom, *reb, *al1, *dividterm;
 long int N1=0, Nx, kmax, j, L;

 double pp;

 //double mut, emut;
 long int i;
 long int k;
 long int Nmax, NO1;
 double dt;
 double q;
 double kx;
double Sl, Sm, Sr;
double pricel, pricem, pricer;
double A, B, C;


 //divid=0.0;

 if(ifCall==0) //IF PUT
 {
  if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
   {
     *ptprice = 0.;
     *ptdelta = 0.;
     printf("Spot is out of range. Increase scale para
    meter {n");
     return OK;
   }
  if( Spot<=1.2*Strike1*exp( -er*log(2.) ) )
   {
     *ptprice = Strike1 - Spot;
     *ptdelta = -1.0;
     printf("Spot is out of range. Increase scale para
    meter {n");
     return OK;
   }
 }
 else
```

```
{
 if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
  {
    *ptprice = - Strike1 + Spot;
    *ptdelta = 1.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
 if( Spot<=Strike1*exp( -er*log(2.) )*1.2 )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}

k=64;

kx=er*log(2)/h;
while(k<kx)
{
     k=k*2;
}
kmax=k;

N1=step;
Nmax=2*kmax;
NO1=kmax;

//Memory allocation for space grid
y=pnl_vect_create_from_zero(Nmax+1);//space grid points
v1=pnl_vect_create_from_zero(Nmax+1);//prices at previous
   time step
vv1=pnl_vect_create_from_zero(2*kmax+2);
al1=pnl_vect_create_from_zero(2*kmax+2);
eom=pnl_vect_create_from_zero(Nmax+1);
reb=pnl_vect_create_from_zero(Nmax+1);
dividterm=pnl_vect_create_from_zero(Nmax+1);
```

```
 //t=pnl_vect_create_from_zero(N1+2); //time points

 lp1+=om;
 lm1+=om;

 /*Time step*/
dt=T/N1;
 //t->array[1]=0;

q=qu;

//============================ compute coefficients
 findcoef_bi(3, mu, 0., lm1, lp1, num, nup, cnum, cnup, q,
    r,T, h, kmax, N1, al1);
 //=========================================


 Nx=Nmax; /*number of space points*/

 /*Pay-off functions*/

if(ifCall==0)
{
for(j=1;j<=Nmax;j++)
 {
  y->array[j]=(j-N01-0.5)*h;
  vv1->array[j-1]=0;
  eom->array[j]=exp(-om*y->array[j]);
  reb->array[j]=Strike1*dt*r*eom->array[j];
  dividterm->array[j]=-Strike1*divid*dt*eom->array[j]*exp(
    y->array[j]);
 }
}
else
{
for(j=1;j<=Nmax;j++)
 {
  y->array[j]=(j-N01+0.5)*h;
  vv1->array[j-1]=0;
  eom->array[j]=exp(-om*y->array[j]);
  reb->array[j]=-Strike1*dt*r*eom->array[j];
```

```
  dividterm->array[j]=Strike1*divid*dt*eom->array[j]*exp(y-
    >array[j]);
 }
}

if(ifCall==0)
{   j=kmax+1;
    do
   {   vv1->array[j-1]=(-Strike1+Strike1*exp(y->array[j]))
    *eom->array[j];
    j++;
   }while(j<=Nmax);
}
else
{   j=kmax-1;
  do
   {   vv1->array[j-1]=(Strike1-Strike1*exp(y->array[j]))*
    eom->array[j];
    j--;
   }while(j>0);
}

 //======================*Main Loop on time grid*=========
    ===============/

 for(L=1; L<=N1; L++)
 {
  fft_real(vv1, 2*kmax, -1);
  expectation(kmax,vv1,al1);

  i=Nmax-1;
    do
    {  vv1->array[i]=vv1->array[i]-reb->array[i+1]-divid
    term->array[i+1];
      if (vv1->array[i]<0)
      {vv1->array[i]=0;}
      i--;
    }while(i>=0);
 }// ======================================== END MAIN LOOP
     ==================
```

```
 if(ifCall==0)
{
  for(i=1;i<Nmax+1;i++)
  {    v1->array[i]=vv1->array[i-1]/eom->array[i]+Strike1-
    Strike1*exp(y->array[i]);
  }
}
else
{
  for(i=1;i<Nmax+1;i++)
  {    v1->array[i]=vv1->array[i-1]/eom->array[i]-Strike1+
    Strike1*exp(y->array[i]);
  }
}

 pp=log(Spot/Strike1);

 i=1;

 while ((y->array[i]<=pp)&&(i<Nx)) {i++;}

  //Price, quadratic interpolation
  Sl = Strike1*exp(y->array[i-1]);
  Sm = Strike1*exp(y->array[i]);
  Sr = Strike1*exp(y->array[i+1]);

  // S0 is between Sm and Sr
  pricel = v1->array[i-1];
  pricem = v1->array[i];
  pricer = v1->array[i+1];

  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);

    //Price
  *ptprice = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  if(*ptprice<0) {*ptprice=0.0;}
    //Delta
  *ptdelta = B + C*(2*Spot-Sl-Sm);
```

```
  //====================================END======
/*Memory desallocation*/
pnl_vect_free(&y);
  pnl_vect_free(&v1);
  pnl_vect_free(&vv1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&reb);
  pnl_vect_free(&dividterm);

return OK;
}

int lookback_fls(int model, double mu, double qu, double om
    , int ifCall, double Spot, double minmax, double lm1,
    double lp1,
            double num,double nup, double cnum,double cn
    up,
            double r, double divid,
            double T, double h,
            double er, double *ptprice, double *ptdelta)
{

  PnlVect  *y,*vv0,*vv1,*tp1,*tm1,*coef,
  *alin1,*al1;
  long int N1=0,Nx, kmax, j;

  double pp;
  double a2_1=0.;
  int nn,n; //Laplace gs
  long int i;
  long int k;
  long int Nmax;
  double dt;

  double q;
  double mu1,kx;
  double Sl, Sm, Sr;
  double pricel, pricem, pricer;
  double price,delta;
```

```
  double A, B, C;
  double sigma;



 k=64;

 kx=er*log(2.0)/h;
 while(k<kx)
 {
      k=k*2;
 }
 kmax=k;

 nn=14;       //gs
// N1=step;
 Nmax=2*kmax;

 //Memory allocation for space grid
 y=pnl_vect_create_from_zero(Nmax+1);//space grid points
 vv0=pnl_vect_create_from_zero(Nmax+1);
 vv1=pnl_vect_create_from_zero(2*kmax+2);
 al1=pnl_vect_create_from_zero(2*kmax+2);
 alin1=pnl_vect_create_from_zero(2*kmax+2);
 tp1=pnl_vect_create_from_zero(2*kmax+2);
 tm1=pnl_vect_create_from_zero(2*kmax+2);
  coef=pnl_vect_create_from_zero(nn+1); // gs
 //eom=pnl_vect_create_from_zero(Nmax+1);




 lp1+=om;
 lm1+=om;
 q=qu;

 //============================ compute coefficients
if(model == 1){
 mu1=0;
 sigma=0.;
```

```
}
/*if(model == 2){
 mu1=mu;
 sigma=0.;
}
if(model == 3){
 mu1=0.0;
 sigma=0.;
 if((mu>0.0)&&(mu<0.1)) {mu1=mu-0.1;}
 if((mu<=0.0)&&(mu>-0.1)) {mu1=mu+0.1;}
 q+= mu1*om;
}*/
if(model == 4){/*KOU*/
 mu1=0.0;//mu

 sigma=nup;
 mu+=sigma*sigma*om;
 nup=num;
 cnum*=(lm1-om)/lm1;
 cnup*=(lp1-om)/lp1;
}
if(ifCall==1)
{
    for(j=0;j<Nmax;j++)
    {
    y->array[j]=(j-kmax)*h;
    if(y->array[j]>0)
    {vv0->array[j]=0;}
        else
    {vv0->array[j]=1.0-exp(y->array[j]);}
  }
}
if(ifCall==0)
{
    for(j=0;j<Nmax;j++)
    {
    y->array[j]=(j-kmax)*h;
    if(y->array[j]<0)
    {vv0->array[j]=0;}
        else
    {vv0->array[j]=exp(y->array[j])-1.0;}
```

```
  }
}



fft_real(vv0, 2*kmax, -1);

//-----------------------------------------------weghts-----
    ------------------------------- gs
coef->array[1]=0.00277778;
coef->array[2]=-6.40277778;
coef->array[3]=924.05000000;
coef->array[4]=-34597.92777778;
coef->array[5]=540321.11111111;
coef->array[6]=-4398346.36666666;
coef->array[7]=21087591.77777770;
coef->array[8]=-63944913.04444440;
coef->array[9]=127597579.55000000;
coef->array[10]=-170137188.08333300;
coef->array[11]=150327467.03333300;
coef->array[12]=-84592161.50000000;
coef->array[13]=27478884.76666660;
coef->array[14]=-3925554.96666666;
//-------------------------------------------------------
    ------------------------------- gs
*ptprice=0;
*ptdelta=0;

//-------------------------------------------------------
    -----loop in weighted terms-----------------------
for(n=1; n<=nn;) // for(n=1; n<=nn;)
{
 /*Time step*/
 dt=T/(log(2.0)*n); //gs
// cout<<dt<<endl;
 q=qu+1/dt;

 findcoefnew(model, mu-mu1, sigma, lm1, lp1, num, nup, cnum
    , cnup, q, r,T, h, kmax, er, N1, al1);
 //=========================================
// a2_1=q*dt;
```

```
a2_1=q;

Nx=Nmax; /*number of space points*/

for(i=0;i<2*kmax;i++) alin1->array[i]=al1->array[i+1];

if(model == 1){/*TSL*/ findfactor(0.0, 0.0, h,  kmax, lp1,
    lm1, alin1, tp1, tm1);}
if(model == 4){/*KOU*/ findfactor(2.0, 2.0, h,  kmax, lp1,
    lm1, alin1, tp1, tm1);}

//if(model == 1){/*TSL*/ factorslb(lm1, lp1, h, 0.0, 0.0,
   kmax, alin1, tp1);}
//if(model == 4){/*KOU*/ factorslb(lm1, lp1, h, 2.0, 2.0,
   kmax, alin1, tp1);}
 //=======================================
if(ifCall==1)
{
 for(i=0;i<Nmax;i++)
   {
     tp1->array[i]=tp1->array[i]/a2_1;
     vv1->array[i]=vv0->array[i];
   }
 expectation(kmax, vv1, tp1);
/* vv1->array[0]= vv1->array[0]*tp1->array[0];
 vv1->array[1]= vv1->array[1]*tp1->array[1];

 for(i = 1; i <= kmax-1; i++)
   {
     t1 = vv1->array[2*i];
     t2 = vv1->array[2*i+1];
     vv1->array[2*i]= t1*tp1->array[2*i]-t2*tp1->array[2*
   i+1];
     vv1->array[2*i+1]= t2*tp1->array[2*i]+t1*tp1->array[2
   *i+1];
   }*/
}
if(ifCall==0)
{
 for(i=0;i<Nmax;i++)
   {
```

```
      tm1->array[i]=tm1->array[i]/a2_1;
      vv1->array[i]=vv0->array[i];
    }
  expectation(kmax, vv1, tm1);
/*  vv1->array[0]= vv1->array[0]*tm1->array[0];
  vv1->array[1]= vv1->array[1]*tm1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tm1->array[2*i]-t2*tm1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tm1->array[2*i]+t1*tm1->array[2
    *i+1];
    }*/
 }
  //fft_real(vv1, 2*kmax, 1);
  pp=log(Spot/minmax);
  i=1;
  while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
  i=i-1;
  //Price, quadratic interpolation
    Sl = minmax*exp(y->array[i-1]);
  Sm = minmax*exp(y->array[i]);
  Sr = minmax*exp(y->array[i+1]);

  // S0 is between Sm and Sr
  pricel = minmax*vv1->array[i-1];
  pricem = minmax*vv1->array[i];
  pricer = minmax*vv1->array[i+1];
  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
  //Price
  price = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  delta = B + C*(2*Spot-Sl-Sm);

  *ptprice = *ptprice+coef->array[n]*price;
  *ptdelta = *ptdelta+coef->array[n]*delta;
```

```
   n++;
}//====================================END LOOP in n
 //   if(*ptprice<0) {*ptprice=0.0;}
if(ifCall==0)
{*ptprice=*ptprice*log(2.0)/T-Spot/exp(divid*T)+minmax/exp(
    r*T);
    *ptdelta=*ptdelta*log(2.0)/T-1/exp(divid*T);
}
else
{*ptprice=Spot/exp(divid*T)-minmax/exp(r*T)+*ptprice*log(2.
    0)/T;
 *ptdelta=1.0/exp(divid*T)+*ptdelta*log(2.0)/T;
}

  //====================================END======
  /*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&vv1);
  pnl_vect_free(&vv0);
  pnl_vect_free(&tp1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
 pnl_vect_free(&coef);

  return OK;
}
int lookback_fxs(int model, double mu, double qu, double om
    , int ifCall, double Spot, double minmax, double lm1,
    double lp1,
            double num,double nup, double cnum,double cn
    up,
            double r, double divid, double Strike,
            double T, double h,
            double er, double *ptprice, double *ptdelta)
{

  PnlVect *y,*vv0,*vv1,*tp1,*tm1,*coef,*alin1,*al1;
  long int N1=0, Nx, kmax, j;
  double pp;
  double a2_1=0.;
```

```
  int nn,n; //Laplace gs
  long int i;
  long int k;
  long int Nmax;
  double dt;

  double q;
  double mu1, str=0., kx;
  double Sl, Sm, Sr;
  double pricel, pricem, pricer;
  double price,delta;
  double A, B, C;
  double sigma;

 k=64;

 kx=er*log(2.0)/h;
 while(k<kx)
 {
      k=k*2;
 }
 kmax=k;

 nn=14;      //gs
// N1=step;
 Nmax=2*kmax;

 //Memory allocation for space grid
 y=pnl_vect_create_from_zero(Nmax+1);//space grid points
 vv0=pnl_vect_create_from_zero(Nmax+1);
 vv1=pnl_vect_create_from_zero(2*kmax+2);
 al1=pnl_vect_create_from_zero(2*kmax+2);
 alin1=pnl_vect_create_from_zero(2*kmax+2);
 tp1=pnl_vect_create_from_zero(2*kmax+2);
 tm1=pnl_vect_create_from_zero(2*kmax+2);
 coef=pnl_vect_create_from_zero(nn+1); // gs

 lp1+=om;
 lm1+=om;
 q=qu;
```

```
 //============================ compute coefficients
if(model == 1){
 mu1=0;
 sigma=0.;
}
/*if(model == 2){
 mu1=mu;
 sigma=0.;
}
if(model == 3){
 mu1=0.0;
 sigma=0.;
 if((mu>0.0)&&(mu<0.1)) {mu1=mu-0.1;}
 if((mu<=0.0)&&(mu>-0.1)) {mu1=mu+0.1;}
 q+= mu1*om;
}*/
if(model == 4){/*KOU*/
 mu1=0.0;//mu

 sigma=nup;
 mu+=sigma*sigma*om;
 nup=num;
 cnum*=(lm1-om)/lm1;
 cnup*=(lp1-om)/lp1;
}
Strike=Strike/minmax;



if(ifCall==0)
{  str=Strike-1;
  if (str<0){str=0.0;}
    for(j=0;j<Nmax;j++)
    {
    y->array[j]=(j-kmax)*h;
    if(y->array[j]>0)
    {vv0->array[j]=0;}
        else
    {vv0->array[j]=Strike-exp(y->array[j]);
    if (vv0->array[j]<0) {vv0->array[j]=0.0;}
    vv0->array[j]=vv0->array[j]-str;
```

```
    }
  }
}
if(ifCall==1)
{  str=1-Strike;
  if (str<0){str=0.0;}
    for(j=0;j<Nmax;j++)
    {
    y->array[j]=(j-kmax)*h;
    if(y->array[j]<0)
    {vv0->array[j]=0;}
        else
    {vv0->array[j]=exp(y->array[j])-Strike;
    if (vv0->array[j]<0) {vv0->array[j]=0.0;}
    vv0->array[j]=vv0->array[j]-str;
    }
  }
}


fft_real(vv0, 2*kmax, -1);

//-----------------------------------------------weghts-----
    -------------------------------- gs
coef->array[1]=0.00277778;
coef->array[2]=-6.40277778;
coef->array[3]=924.05000000;
coef->array[4]=-34597.92777778;
coef->array[5]=540321.11111111;
coef->array[6]=-4398346.36666666;
coef->array[7]=21087591.77777770;
coef->array[8]=-63944913.04444440;
coef->array[9]=127597579.55000000;
coef->array[10]=-170137188.08333300;
coef->array[11]=150327467.03333300;
coef->array[12]=-84592161.50000000;
coef->array[13]=27478884.76666660;
coef->array[14]=-3925554.96666666;
//-------------------------------------------------------------
    -------------------------------- gs
*ptprice=0;
```

```
*ptdelta=0;

//-------------------------------------------------------
    -----loop in weighted terms-----------------------
for(n=1; n<=nn;) // for(n=1; n<=nn;)
{
 /*Time step*/
 dt=T/(log(2.0)*n); //gs
// cout<<dt<<endl;
 q=qu+1/dt;

 findcoefnew(model, mu-mu1, sigma, lm1, lp1, num, nup, cnum
    , cnup, q, r,T, h, kmax, er, N1, al1);
 //=========================================
// a2_1=q*dt;
 a2_1=q;

 Nx=Nmax; /*number of space points*/

 for(i=0;i<2*kmax;i++) alin1->array[i]=al1->array[i+1];

 if(model == 1){/*TSL*/ findfactor(0.0, 0.0, h,  kmax, lp1,
     lm1, alin1, tp1, tm1);}
 if(model == 4){/*KOU*/ findfactor(2.0, 2.0, h,  kmax, lp1,
     lm1, alin1, tp1, tm1);}

 //if(model == 1){/*TSL*/ factorslb(lm1, lp1, h, 0.0, 0.0,
    kmax, alin1, tp1);}
 //if(model == 4){/*KOU*/ factorslb(lm1, lp1, h, 2.0, 2.0,
    kmax, alin1, tp1);}
  //=======================================
 if(ifCall==0)
 {
  for(i=0;i<Nmax;i++)
    {
      tp1->array[i]=tp1->array[i]/a2_1;
      vv1->array[i]=vv0->array[i];
    }
  expectation(kmax, vv1, tp1);
 /* vv1->array[0]= vv1->array[0]*tp1->array[0];
  vv1->array[1]= vv1->array[1]*tp1->array[1];
```

```
  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tp1->array[2*i]-t2*tp1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tp1->array[2*i]+t1*tp1->array[2
    *i+1];
    }*/
 }
 if(ifCall==1)
 {
  for(i=0;i<Nmax;i++)
    {
      tm1->array[i]=tm1->array[i]/a2_1;
      vv1->array[i]=vv0->array[i];
    }
  expectation(kmax, vv1, tm1);
  /*vv1->array[0]= vv1->array[0]*tm1->array[0];
  vv1->array[1]= vv1->array[1]*tm1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tm1->array[2*i]-t2*tm1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tm1->array[2*i]+t1*tm1->array[2
    *i+1];
    }*/
 }
//  fft_real(vv1, 2*kmax, 1);
  pp=log(Spot/minmax);
  i=1;
  while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
  i=i-1;
  //Price, quadratic interpolation
  Sl = minmax*exp(y->array[i-1]);
  Sm = minmax*exp(y->array[i]);
  Sr = minmax*exp(y->array[i+1]);
```

```
  // S0 is between Sm and Sr
  pricel = minmax*vv1->array[i-1];
  pricem = minmax*vv1->array[i];
  pricer = minmax*vv1->array[i+1];
  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
  //Price
  price = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  delta = B + C*(2*Spot-Sl-Sm);

  *ptprice = *ptprice+coef->array[n]*price;
  *ptdelta = *ptdelta+coef->array[n]*delta;
   n++;
}//====================================END LOOP in n
 //   if(*ptprice<0) {*ptprice=0.0;}
//if(ifCall==1)
//{*ptprice=*ptprice*log(2.0)/T+minmax/exp(r*T)*str;
//    *ptdelta=*ptdelta*log(2.0)/T;
//}
//else
//{*ptprice=str*minmax/exp(r*T)+*ptprice*log(2.0)/T;
// *ptdelta=*ptdelta*log(2.0)/T;
//}
  *ptprice=str*minmax/exp(r*T)+*ptprice*log(2.0)/T;
  *ptdelta=*ptdelta*log(2.0)/T;
  //====================================END======
  /*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&vv1);
  pnl_vect_free(&vv0);
  pnl_vect_free(&tp1);
  pnl_vect_free(&alin1);
 // pnl_vect_free( tbp1);
 // pnl_vect_free( ltp1);
 // pnl_vect_free( ltm1);
 // pnl_vect_free( tb1);
 // pnl_vect_free( tbm1);
  pnl_vect_free(&tm1);
```

```
  pnl_vect_free(&al1);
  pnl_vect_free(&coef);
  // pnl_vect_free( t);
  return OK;
}
int swing(int model, double mu, double qu, double om,
int ifCall, double Spot, double lm1, double lp1,
    double num,double nup, double cnum,double cnup,
    double r, double divid,
    double T, double h, double Strike1, double del, int Nd,
    double er, long int step,
    double *ptprice, double *ptdelta)
{
 PnlVect *t, *y, * v1, *gg, *G, *vv1,*tp1,*tm1,*eomcut,
    *alin1,*eom, *reb, *switcharrow,*al1, *bl1;
 int dims[3], tab[3];
 PnlHmat *Price;
 long int N1=0, Nx, kmax, j, L, n, r_index,k_index;

 double pp;
 double t1, t2;

 double a2_1=0.;
long int i;
 long int k;
 long int Nmax, N01;
 double dt;
 double q;
 double kx;
double Sl, Sm, Sr;
double pricel, pricem, pricer;
double A, B, C;
double sigma;
double flagCall;
 flagCall = ifCall ? 1.0 : -1.0; //  1.0 for Call, -1.0 fo
    r Put


 if(ifCall==0) //IF PUT
 {
  if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
```

```
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
 if( Spot<=1.2*Strike1*exp( -er*log(2.) ) )
  {
    *ptprice = Strike1 - Spot;
    *ptdelta = -1.0;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}
else
{
 if( Spot>=0.8*Strike1*exp( er*log(2.) ) )
  {
    *ptprice = - Strike1 + Spot;
    *ptdelta = 1.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
 if( Spot<=Strike1*exp( -er*log(2.) )*1.2 )
  {
    *ptprice = 0.;
    *ptdelta = 0.;
    printf("Spot is out of range. Increase scale para
   meter {n");
    return OK;
  }
}

k=64;

kx=er*log(2.0)/h;
while(k<kx)
{
```

```
     k=k*2;
}
kmax=k;

N1=step;
Nmax=2*kmax;
NO1=kmax;

//Memory allocation for space grid
y=pnl_vect_create(Nmax+1);//space grid points
v1=pnl_vect_create(Nmax+1);//prices at previous time step
gg=pnl_vect_create(2*kmax+2);//payoff
vv1=pnl_vect_create(2*kmax+2);
tp1=pnl_vect_create(2*kmax+2);
alin1=pnl_vect_create(2*kmax+2);
tm1=pnl_vect_create(2*kmax+2);
al1=pnl_vect_create(2*kmax+2);
bl1=pnl_vect_create(2*kmax+2);
eom=pnl_vect_create(Nmax+1);
eomcut=pnl_vect_create(Nmax+1);
reb=pnl_vect_create(Nmax+1);
G=pnl_vect_create(Nmax+1);
t=pnl_vect_create(N1+2); //time points



 dims[0]=N1+1;
 dims[1]=Nd+1;
   dims[2]=Nmax+1;
 Price=pnl_hmat_create(3,dims);



/*Time step*/
 dt=T/N1;
 /*First index where i*t<t-r_period*/
 for(n=0; n<=N1; n++) {t->array[n]= n*dt;}
// int M=Ntime;

// mat_index=M;
 i=N1;
```

```
  do{
    i--;
  }while(t->array[i]>(T-del)+0.00000001);

r_index=i;
k_index=N1-r_index;


 lp1+=om;
 lm1+=om;


  q=qu+1/dt;

//============================ compute coefficients
  if(model == 1){/*TSL*/
  q=q-mu*om;
    sigma=0.;
}
if(model == 4){/*KOU*/
 sigma=nup;
 mu+=sigma*sigma*om;
 nup=num;
 cnum*=(lm1-om)/lm1;
 cnup*=(lp1-om)/lp1;
}
if(model == 5){/*Merton*/
 sigma=num; //nup=p0; cnum=lambda;cnup=gam0;
 mu+=sigma*sigma*om;// change for non-zero om!!!
 //cnum*=(lm1-om)/lm1;
 //cnup*=(lp1-om)/lp1;
 num=2.0;
}
findcoef_sw(model, mu, sigma, lm1, lp1, num, nup, cnum, cn
    up, q, r,T, h, kmax, del, N1, al1, bl1);
 //=========================================

a2_1=q*dt;
if(model == 1){/*TSL*/
 nup=0.0;
 num=0.0;
```

```
}
if(model == 5){/*Merton*/
 nup=num;
}
 Nx=Nmax; /*number of space points*/
 ////////////////////////////////////begin factors
 for(i=0;i<2*kmax;i++) alin1->array[i]=al1->array[i+1];
 findfactor(nup, num, h,  kmax, lp1, lm1, alin1, tp1, tm1);


if(ifCall==0)
{
for(j=0;j<Nmax;j++)
 {
  y->array[j]=(j-N01-0.5)*h;
  gg->array[j]=0;
  eomcut->array[j]=0;
  eom->array[j]=exp(om*y->array[j]);
  reb->array[j]=Strike1*dt*r*eom->array[j];
 }
}
else
{
for(j=0;j<Nmax;j++)
 {
  y->array[j]=(j-N01+0.5)*h;
  gg->array[j]=0;
  eomcut->array[j]=0;
  eom->array[j]=exp(om*y->array[j]);
  reb->array[j]=-Strike1*dt*r*eom->array[j];
 }
}

if(ifCall==0)
{   j=kmax;
    do
   {   eomcut->array[j]=exp(y->array[j])*eom->array[j];
    gg->array[j]=Strike1*(eom->array[j]-eomcut->array[j])
    ;//ok
    j--;
   }while(j>=0);
```

```
}
else
{   j=kmax;
  do
   {   eomcut->array[j]=exp(y->array[j])*eom->array[j];
    gg->array[j]=-Strike1*(eom->array[j]-eomcut->array[j]
    );//ok
    j++;
   }while(j<Nmax);
}

  //======= EXCHANGE tp AND tm ARRAYS FOR UP-OUT
  if(ifCall==1)
  {
  switcharrow = tp1;
  tp1 = tm1;
  tm1 = switcharrow;
  }

        for(i=0;i<2*kmax;i++)
    {G->array[i]=Strike1*flagCall*eom->array[i]*(exp(y->
    array[i])-1);}
        fft_real(G, 2*kmax, -1);
      expectation(kmax, G, tm1);

//=========================================================
    =============
  for(i=0;i<Nmax;i++)
  {   tp1->array[i]=tp1->array[i]/a2_1;
  }
//============================Loop in exercise possibiliti
    es==========
  dims[0]=0;
  tab[1]=0;
for (j=0;j<Nd;j++)
{   dims[1]=j;
if (j>0){tab[1]=j-1;}
  for(i=0;i<2*kmax;i++)
  {
    vv1->array[i]=gg->array[i];
    reb->array[i]=G->array[i];
```

```
  }
  fft_real(vv1, 2*kmax, -1);
  for(i=0;i<Nmax;i++) {dims[2]=i; pnl_hmat_set(Price, dim
    s, vv1->array[i]);}//Price[0][j][i]

  //======================*Main Loop on time grid*========
    ================/

 for(L=1; L<N1+1; L++)
{ dims[0]=L;
  //mut=t->array[L]*mu1;
  //emut=exp(-mut);


  vv1->array[0]= vv1->array[0]*tm1->array[0];
  vv1->array[1]= vv1->array[1]*tm1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tm1->array[2*i]-t2*tm1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tm1->array[2*i]+t1*tm1->array[2
    *i+1];
    }
  fft_real(vv1, 2*kmax, 1);
  tab[0]=L-k_index;
  if((L>k_index-1)&&(j>0))
  {
      for(i=0;i<2*kmax;i++)
      {  tab[2]=i;
        reb->array[i]=pnl_hmat_get(Price,tab); } //pnl_
    hmat_get(Price,L-k_index,j-1,i);
        expectation(kmax, reb, bl1);
        fft_real(reb, 2*kmax, -1);
        expectation(kmax, reb, tm1);
      for(i=0;i<2*kmax;i++)
      {  reb->array[i]=reb->array[i]+G->array[i];
      }
```

```
  }

  i=Nmax-1;
    do
    {  if (vv1->array[i]<reb->array[i])
      {vv1->array[i]=reb->array[i];}
      i--;
    }while(i>=0);



  fft_real(vv1, 2*kmax, -1);


  vv1->array[0]= vv1->array[0]*tp1->array[0];
  vv1->array[1]= vv1->array[1]*tp1->array[1];

  for(i = 1; i <= kmax-1; i++)
    {
      t1 = vv1->array[2*i];
      t2 = vv1->array[2*i+1];
      vv1->array[2*i]= t1*tp1->array[2*i]-t2*tp1->array[2*
    i+1];
      vv1->array[2*i+1]= t2*tp1->array[2*i]+t1*tp1->array[2
    *i+1];
    }
  for(i=0;i<Nmax;i++) {dims[2]=i; pnl_hmat_set(Price, dims,
     vv1->array[i]);}  //pnl_hmat_set(Price, L, j, i, vv1->ar
    ray[i]);
 }// ======================================== END MAIN LOOP
     ==================
}//end of exercise loop
  fft_real(vv1,2*kmax, 1);

if(ifCall==0)
{
  for(i=0;i<Nmax;i++)
  {    y->array[i]=y->array[i];
       v1->array[i]=vv1->array[i]/eom->array[i];
  }
}
```

```
else
{
  for(i=0;i<Nmax;i++)
  {    y->array[i]=y->array[i];
       v1->array[i]=vv1->array[i]/eom->array[i];
  }
}

 pp=log(Spot/Strike1);

 i=1;

 while ((y->array[i]<=pp)&&(i<Nx)) {i++;}
  i=i-1;
  //Price, quadratic interpolation
  Sl = Strike1*exp(y->array[i-1]);
  Sm = Strike1*exp(y->array[i]);
  Sr = Strike1*exp(y->array[i+1]);

  // S0 is between Sm and Sr
  pricel = v1->array[i-1];
  pricem = v1->array[i];
  pricer = v1->array[i+1];

  //quadratic interpolation
  A = pricel;
  B = (pricem-pricel)/(Sm-Sl);
  C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);

    //Price
  *ptprice = A+B*(Spot-Sl)+C*(Spot-Sl)*(Spot-Sm);
  if(*ptprice<0) {*ptprice=0.0;}
    //Delta
  *ptdelta = B + C*(2*Spot-Sl-Sm);

  //====================================END======
/*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&v1);
  pnl_vect_free(&vv1);
  pnl_vect_free(&tp1);
```

```
  pnl_vect_free(&alin1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&reb);
  pnl_vect_free(&t);

  pnl_hmat_free(&Price);
return OK;
}
////////////////////////////VAR and CTE//////////////////
    ///////

int var_fft(int model, double mu,
    double Spot, double lm1, double lp1,
    double num,double nup, double cnum,double cnup,
    double T, double h, double Strike1, double er, double
    alpha,
    double *ptprice, double *ptdelta)
{

 PnlVect  *y, *vv1, *al1;
 long int kmax, j;

 double kx;
 long int i;
 long int k;
 long int Nmax;

 k=64;

 kx=er*log(4)/h;
 while(k<kx)
 {
      k=k*2;
 }
 kmax=k;

 Nmax=2*kmax;
```

```
 //Memory allocation for space grid
 y=pnl_vect_create_from_zero(Nmax+1);//space grid points
 vv1=pnl_vect_create_from_zero(2*kmax+2);
 al1=pnl_vect_create_from_zero(2*kmax+2);


////////////////space grid//////////////////////
for(j=0;j<Nmax;j++){y->array[j]=(j-kmax)*h;}

//=========================== compute coefficients for chf
 findcoef_var(1, mu, 0., lm1, lp1, num, nup, cnum, cnup, 0.
    , T, h, kmax, 1, al1);
 //=========================================


////////////////////////recover probabilities///////////
  vv1->array[0]=1.;
  vv1->array[1]=1.;

  i=1;
  j=1;
  do
    { j=-j;
      vv1->array[2*i]=j;
      vv1->array[2*i+1]=0.;
      i++;
 }while(i<kmax);

  expectation(kmax,vv1,al1);



  ////////////////////////////////compute VaR/////////////

 kx=0.;
 i=0;
 while ((kx<alpha)&&(i<Nmax-1))
{kx=kx+vv1->array[i]; i=i+1;
}
```

```
i=i-1;
    //VaR
   *ptprice = Spot*exp(y->array[i])-Strike1;

  ////////////////////////compute CTE////////////////
  *ptdelta = 0;
  while (i<Nmax)
  {
    *ptdelta=*ptdelta+vv1->array[i]*exp(y->array[i]);
    i=i+1;
  }
     //CTE
  *ptdelta = *ptdelta*Spot/(1-alpha)-Strike1;

  //====================================END======
/*Memory desallocation*/
  pnl_vect_free(&y);
  pnl_vect_free(&vv1);
  pnl_vect_free(&al1);
  return OK;
}

#endif //PremiaCurrentVersion
```

References