Help

```
#include "lmm_heston1d_stdi.h"
#include "math/lmm/lmm_libor.h"
#include "math/lmm/lmm_products.h"
#include "math/lmm/lmm_volatility.h"
#include "math/lmm/lmm_numerical.h"
#include "math/lmm/lmm_zero_bond.h"
#include "pnl/pnl_complex.h"
#include "math/integral.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(AP_WZ)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_WZ)(void *Opt,void *Mod,PricingMethod *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

static Volatility *ptVol;
static Libor *ptLib;
static Swaption* ptSwpt;
//static Caplet* ptCplt;

// stochastic volatility variables
static double epsilon;
static double kappa;
static double theta;
static double V0;
static double rho1_lmm;
static double rho2_lmm;
static double rho3_lmm;

static int* initlog ;
static int* bk ;
static dcomplex dlk;
```

```
static dcomplex* lk_1;

static int* initSqrt;
static int* sk;
//static dcomplex dsk;
static dcomplex* sk_1;


//static double *f1;
//static double *f2;
/* static int indicegene=0; */
/* static int* branch; */

// global variables

static double* ptCor=NULL;
static double *omega=NULL;
static double *alpha=NULL;
static dcomplex* A=NULL ;
static dcomplex* g=NULL ;
static dcomplex* a=NULL ;
static dcomplex* d=NULL ;
static dcomplex* B=NULL ;
static double *lambda=NULL;
static double *zeta=NULL;
static double *beta=NULL;

static int nb_time, int_type;
static double spot, strike , T;


static int compute_a(dcomplex* a, dcomplex z, double* beta
    ,  double* zeta, int nb_time);
static int compute_d(dcomplex*d , dcomplex * a, dcomplex z,
     double * lambda,  int nb_time);
static int compute_B(dcomplex* B , dcomplex* g , dcomplex*
    a, dcomplex* d ,int nb_time);
static int compute_A(dcomplex* A , dcomplex* g , dcomplex*
    a, dcomplex* d ,int nb_time);

static int compute_price(double t, Swaption* ptSwpt,double
```

```
    sigma);

static double func(double u);
static double compute_beta_swaption( double t , Swaption*
    ptSwpt , double* alpha,double sigma );
static double compute_zeta_swaption(double t, Swaption* pt
    Swpt,double sigma );
static double compute_lambda_swaption(double t ,  Swaption*
     ptSwpt , double* omega,double sigma);
static void init_log(int nbpoints);
static void free_log();
static dcomplex characteristic_function( double x, double
    V, dcomplex  z, double T, double* beta, double* zeta,
    double* lambda, dcomplex* A , dcomplex* g, dcomplex* a, dcompl
    ex* d, dcomplex* B , int nb_time );

static int build_correlation(double **ptCr, int number_of_
    corr);
static int check_parameters(double period , int number_of_
    factors ,double swaption_maturity , double swap_maturity ,
    double percent_of_ATM_strike );


static void init_log(int nbpoints)
{
  int i;

  initlog =(int *)malloc(sizeof(int)*nbpoints);
  initSqrt = (int *)malloc(sizeof(int)*nbpoints);

  bk =(int *)malloc(sizeof(int)*nbpoints);
  lk_1=(dcomplex *)malloc(sizeof(dcomplex)*nbpoints);
  sk =(int *)malloc(sizeof(int)*nbpoints);
  sk_1=(dcomplex *)malloc(sizeof(dcomplex)*nbpoints);
  for (i=0;i<nbpoints;i++)
    {
      initlog[i]=0;
      initSqrt[i]=0;
      bk[i]=0;
      sk[i]=0;
    }
```

```
}


static void free_log()
{
  free(initlog);
  initlog=NULL;
  free(initSqrt);
  initSqrt=NULL;
  free(bk);
  bk=NULL;
  free(lk_1);
  lk_1=NULL;
  free(sk);
  sk=NULL;
  free(sk_1);
  sk_1=NULL;
}

static double lmm_swaption_payer_stoVol_pricer(double perio
    d , int number_of_factors ,double swaption_maturity ,
    double swap_maturity ,double strike ,double sigma,double l0)
{

  double tenor=period;
  int number_of_dates;
  double priceVal=0.25;
  double price;
  double percent_of_ATM_strike;

  percent_of_ATM_strike=-20.;
  check_parameters(period , number_of_factors , swaption_
    maturity ,swap_maturity ,percent_of_ATM_strike);
  number_of_dates=(int)(swap_maturity/period)+2;
  mallocLibor(&ptLib , number_of_dates , tenor,l0 );
  mallocVolatility(&ptVol , number_of_factors , sigma);

  //s=(int)(swaption_maturity/tenor);
  //M=(int)(swap_maturity/tenor);
  //atm_Strike=(computeZeroCoupon(ptLib,0,s) - computeZero
    Coupon(ptLib,0,M))/computeZeroCouponSum(ptLib, 0,s+1,M );
```

```
  /*atm_Strike*=(1 +  percent_of_ATM_strike/100.);*/
  /*strike=atm_Strike;*/

  mallocSwaption(&ptSwpt , swaption_maturity , swap_maturit
    y , priceVal , strike , tenor );

  omega=(double *)malloc(sizeof(double)*   ptSwpt->number0
    fDates  );
  alpha=(double *)malloc(sizeof(double)*   ptSwpt->number0
    fDates  );

  // build the correlation structure
  build_correlation(&ptCor,number_of_factors);
  compute_price(0.0 , ptSwpt,sigma);
  price=ptSwpt->price;

  // free memory
  free(ptCor);
  ptCor=NULL;
  free(omega);
  omega=NULL;
  free(alpha);
  alpha=NULL;
  freeSwaption(&ptSwpt);
  ptSwpt=NULL;
  freeLibor(&ptLib);
  ptLib=NULL;
  freeVolatility(&ptVol);
  ptVol=NULL;

  return(price);
}

static int check_parameters(double period , int number_of_
    factors ,double swaption_maturity , double swap_maturity ,
    double percent_of_ATM_strike )
{// this function checks the consistency of the parameters

  double s;
  double M;
```

```
  if(swaption_maturity>=swap_maturity)
    {
      printf(" swaption maturity must be lower than swap
    maturity !{n");
      exit(-1);
    }
  if(number_of_factors>3)
    {
       printf("number of factors must be lower or equal to
    3 {n");
    }
  s=(int)(swaption_maturity/period);
  if (fabs(swaption_maturity-s*period)> 0.0)
    {
      printf(" swaption maturity must be a multiple of pe
    riod{n");
      exit(-1);
    }
  M=(int)(swap_maturity/period);
  if (fabs(swap_maturity-M*period)> 0.0)
    {
      printf(" swap maturity must be a multiple of period{
    n");
      exit(-1);
    }
  if( (int)(number_of_factors>3))
    {
       printf("number of factors must be lower or equal to
    3 {n");
       exit(-1);
    }
  /*if((percent_of_ATM_strike<-40.) || (percent_of_ATM_stri
    ke>40.))
    {
       printf("percent of the ATM strike must be within th
    e range [-40{% ; 40{%] {n");
       exit(-1);
       }*/

  return(1);
}
```

```
static int build_correlation( double **ptCorrel , int numb
    er_of_factors)
{
  if(number_of_factors>3)
    {
      printf("Only three factors allorwed!!!{n");
      exit(-1);
    }
  else
    {
      *ptCorrel=(double *)malloc(sizeof(double)*number_of_
    factors);
      switch( number_of_factors )
  {
  case 1:
    (*ptCorrel)[0]=rho1_lmm;
    break;
  case 2:
    (*ptCorrel)[0]= rho1_lmm ;
    (*ptCorrel)[1]= rho2_lmm;
    break;
  case 3:
    (*ptCorrel)[0]= rho1_lmm ;
    (*ptCorrel)[1]= rho2_lmm;
    (*ptCorrel)[2]= rho3_lmm;
    break;
  default:
    exit(-1);
    break;
  }
    }
  return(1);
}


/************************************************************
    ************************************************************
    ***********************************
 *
```

```
 *
 *
 *
 *   functions needed for the computation of the characteri
    stic function of the process
 *
 *
 ************************************************************
    ************************************************************
    ************************************/

static int compute_a(dcomplex * a, dcomplex z, double * bet
    a ,  double * zeta, int nb_time)
{
  int j;
  dcomplex b0;

  for(j=0;j<nb_time;j++)
    {
      a[j]=RCmul( -zeta[j]*epsilon,z);
      b0=Complex( kappa*beta[j] ,0.0);
      a[j]=Cadd(a[j],b0);
    }
  return(1);

}

static int compute_d(dcomplex*d , dcomplex * a, dcomplex z,
     double * lambda,  int nb_time)
{
  int j;
  dcomplex b0,b1,b2,slk;
  b0=Cmul(z,z);
  b0=Csub(b0,z);

  for(j=0;j<nb_time;j++)
    {
      b1=RCmul( pow(lambda[j]*epsilon,2) , b0);
      b2=Cmul( a[j] , a[j] );
      d[j]=Csub( b2 , b1 );
      d[j]=Clog(d[j]);
```

```
      // Test for  Sqrt
      if(initSqrt[j]>0)
  {
    slk = Csub(d[j],sk_1[j]);
    if(slk.i < -M_PI)
      {

        sk[j] = sk[j] + 1;

      }
    else if(slk.i > M_PI)
      {
        sk[j] = sk[j] - 1;
      }

    sk_1[j] = d[j];
  }
      else
  {
    initSqrt[j]++;
    sk_1[j] = d[j];
  }
      //
      d[j] = Cadd( d[j], Complex(0.,2*M_PI*sk[j]));
      d[j] = RCmul(0.5,d[j]);
      d[j] = Cexp(d[j]);

    }
  return(1);
}

static int compute_B(dcomplex* B , dcomplex* g , dcomplex*
    a, dcomplex* d ,int nb_time)
{
  dcomplex b1,b0,b2,b4,b3;
  double tau=ptLib->tenor;
  int j;

  B[ nb_time -1 ] = Complex( 0.0 , 0.0 );
  g[ nb_time -1 ] = Cdiv( Cadd(a[ nb_time -1 ],d[ nb_time -
```

```
  1 ]) , Csub(a[ nb_time -1 ],d[ nb_time -1 ]) );

for(j = nb_time - 1 ; j >= 0 ;j--)
  {


    if ( j==(nb_time-1) )
{
  g[j]=Cdiv( Cadd(a[j],d[j])   ,  Csub(a[j],d[j])    );
  b4=Complex(0.0,0.0);
}
    else
{
  g[j]=Cdiv( Csub( Cadd(a[j],d[j]),RCmul(epsilon*epsilon
  ,B[j+1]))  , Csub( Csub(a[j],d[j]) , RCmul(epsilon*epsilon
  ,B[j+1]) )  );
  b4=RCmul(epsilon*epsilon,B[j+1]);
}

    b1=RCmul(tau , d[j]);
    b1=Cexp(b1);

    b0=Complex(1.0,0.0);
    b0=Csub(b0,b1);

    b1=Cmul(g[j],b1);
    b3=Complex(1.0,0.0);
    b3=Csub(b3,b1);
    b3=RCmul(epsilon*epsilon, b3);

    b2=Cadd(a[j],d[j]);
    b2=Csub(b2,b4);
    b2=Cmul(b2,b0);

    b2=Cdiv(b2,b3);

    if ( j==(nb_time-1) )
{
  B[j]=b2;
}
    else
```

```
  {
    B[j]=Cadd(B[j+1],b2);

  }
    }
  return(1);
}


static int compute_A(dcomplex* A , dcomplex* g , dcomplex*
    a, dcomplex* d ,int nb_time)
{
  dcomplex b0,b1,b2,b3,b4;
  int j;
  double tau=ptLib->tenor;


  for( j=nb_time-1 ; j>=0 ; j-- )
    {
      b1=RCmul(tau , d[j]);
      b1=Cexp(b1);
      b3=Cmul(g[j],b1);

      b0=Complex(1.0,0.0);
      b0=Csub(b0,b3);

      b2=Complex(1.0,0.0);
      b2=Csub(b2,g[j]);

      b2=Cdiv(b0,b2);
      b2=Clog(b2);

      // Test for the log function
      if(initlog[j]>0)
  {
    dlk = Csub(b2,lk_1[j]);
    if(dlk.i < -M_PI)
      {
        bk[j] = bk[j] + 1;

      }
```

```
    else if(dlk.i > M_PI)
      {
        bk[j] = bk[j] - 1;
      }

    lk_1[j] = b2;
  } else {
    initlog[j]++;
    lk_1[j] = b2;
  }

      b2 = Cadd(b2, Complex(0.,2*M_PI*bk[j]));
      b2=RCmul(2.,b2);

      b4=Cadd(a[j],d[j]);
      b4=RCmul(tau, b4);
      b4=Csub(b4,b2);
      b4=RCmul(kappa*theta/pow(epsilon,2),b4);

      if ( j==(nb_time-1) )
  {
    A[j] = b4 ;
  }
      else
  {
    A[j] = Cadd(A[j+1],b4);
  }
    }

  return(1);
}



/*************************************************
 *
 *  swaption
 *
 *************************************************/

static int build_beta_swaption(double* beta, Swaption* pt
```

```
    Swpt,double sigma)
{
  double t;
  int i;
  int nb_time;

  nb_time=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);

  for (i=0;i<nb_time;i++)
    {
      t=i*ptLib->tenor;
      beta[i]=compute_beta_swaption( t , ptSwpt , alpha,si
    gma);
    }
  return(1);
}

static int build_lambda_swaption(double* lambda , Swaption*
     ptSwpt , Libor *ptLib , double* omega,double sigma)
{
  double t;
  int i,nb_time;

  nb_time=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);

  for (i=0;i<nb_time;i++)
    {
      t=i*ptLib->tenor;
      lambda[i]=compute_lambda_swaption(t , ptSwpt , omega,
    sigma );
    }
  return(1);
}

static int build_zeta_swaption( double* zeta , Swaption* pt
    Swpt , Libor *ptLib,double sigma )
{
  double t;
  int i,nb_time;

  nb_time=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);
```

```
  for (i=0;i<nb_time;i++)
    {
      t=i*ptLib->tenor;
      zeta[i]=compute_zeta_swaption( t, ptSwpt,sigma );
    }
  return(1);
}


static double compute_beta_swaption( double t , Swaption*
    ptSwpt , double* alpha,double sigma)
{
  int i,j,l;
  float v;
  float sum=0.0;
  int M,s;
  double Ti;
  double epsj;

  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);

  sum=0.0;
  for(j=s;j<M;j++)
    {

      epsj=0.0;
      for(i=1;i<=j;i++)
  {
    Ti=GET(ptLib->maturity,i);
    v=0.0;
    for(l=0;l<ptVol->numberOfFactors;l++)
      {
        v+=evalVolatility(ptVol,l,t,Ti)*ptCor[l];
      }

    v*=(ptLib->tenor*GET(ptLib->libor,i))/(1+ptLib->tenor*
    GET(ptLib->libor,i));
    epsj+=v;
  }
```

```
      epsj*=alpha[j]*epsilon/kappa;
      sum+=epsj;
    }

  sum+=1.;
  return(sum);
}


static double compute_zeta_swaption(double t, Swaption* pt
    Swpt,double sigma)
{
  int j,l;
  double sum,Tj;
  int M,s;
  double v;

  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);

  sum=0.0;

  for (j=s;j<M;j++)
    {
      Tj=GET(ptLib->maturity,j);
      v=0.0;
      for(l=0;l<ptVol->numberOfFactors;l++)
  {
    v+=evalVolatility(ptVol,l,t,Tj)*ptCor[l];
  }
      sum+= (v*omega[j]);
    }
  return(sum);
}

static double compute_lambda_swaption(double t ,  Swaption*
     ptSwpt , double* omega,double sigma)
{

  int l,i1,i2;
  double sum=0.0;
```

```
  int M,s;
  double Ti1,Ti2;


  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);


  for(l=0;l<ptVol->numberOfFactors;l++)
    {
      for(i1=s;i1<M;i1++)
  {
    for(i2=s;i2<M;i2++)
      {

        Ti1=GET(ptLib->maturity,i1);
        Ti2=GET(ptLib->maturity,i2);
        sum+= omega[i1]*omega[i2]*evalVolatility(ptVol,l,
    t,Ti1)*evalVolatility(ptVol,l,t,Ti2);
      }
  }
    }
  return(sqrt(sum));
}



static int compute_Omega(Libor *ptLib, Swaption* ptSwpt ,
    double *omega)
{
  int j;
  double BT_M,BT_s,sumZc;
  int s,M;

  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);


  BT_M=computeZeroCoupon(ptLib,0,M);
  BT_s=computeZeroCoupon(ptLib,0,s);
  sumZc=computeZeroCouponSum(ptLib, 0,s+1,M );
```

```
  for(j=0;j<M;j++)
    {
      omega[j]=0.0;
    }

  for(j=s;j<M;j++)
    {
      omega[j]=(ptLib->tenor*GET(ptLib->libor,j)/(1+ptLib->
    tenor*GET(ptLib->libor,j)))*( BT_M/(BT_s-BT_M) + computeZero
    CouponSum(ptLib, 0,j+1,M )/sumZc) ;

    }
  return(1);


}

static int compute_alpha(Libor *ptLib, Swaption* ptSwpt ,
    double* alpha)
{
  int j;
  int M,s;

  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);

  for(j=0;j<M;j++)
    {
      alpha[j]=0.0;
    }

  for(j=s;j<M;j++)
    {
      alpha[j]=(ptLib->tenor*computeZeroCoupon(ptLib,0,j+1)
    )/computeZeroCouponSum(ptLib, 0,s+1,M ) ;
    }

  return(1);
```

```
}


/***************************/

static dcomplex characteristic_function( double x, double
    V, dcomplex  z, double T, double* beta, double* zeta,
    double* lambda, dcomplex* A , dcomplex* g, dcomplex* a, dcompl
    ex* d, dcomplex* B , int nb_time )
{

  dcomplex resu;

  compute_a(a, z,  beta ,   zeta,  nb_time);
  compute_d(d ,  a,  z, lambda,   nb_time);
  compute_B( B , g , a, d , nb_time);
  compute_A( A ,  g ,  a,  d , nb_time);


  resu=Cexp( Cadd(A[0],RCmul(V,B[0])) );

  return(resu);

}



static double func(double u)
{

  dcomplex uc1,uc2,uc3;

  if (int_type==1)
    {
      uc1 = Complex(0.0,u);
    }
  else
    {
      uc1 = Complex(1. , u );
    }
  uc3 = characteristic_function( 0.0 ,  V0 , uc1 , T , bet
```

```
    a , zeta , lambda , A , g , a , d , B , nb_time );
  uc2=Cexp( Complex(0.0, -u *log( strike / spot )));
  uc3=Cmul( uc2 , uc3 );
  return( uc3.i / u )  ;
}


int compute_price(double t, Swaption *ptSwpt,double sigma )
{

  int i,M,s;
  double BT_M, BT_s,sumZc;
  double  w;
  double price;
  int ngauss=10. ,N=1000;
  double ai,b,h=1.,tol=1.e-12,tpi,tp3,tp1,tp2;

  nb_time=(int)(ptSwpt->swaptionMaturity / ptSwpt->tenor );

  beta = (double *)malloc(nb_time*sizeof(double));
  zeta = (double *)malloc(nb_time*sizeof(double));
  lambda = (double *)malloc(nb_time*sizeof(double));

  A=(dcomplex*)malloc(nb_time*sizeof(dcomplex)) ;
  g=(dcomplex*)malloc(nb_time*sizeof(dcomplex)) ;
  a=(dcomplex*)malloc(nb_time*sizeof(dcomplex)) ;
  d=(dcomplex*)malloc(nb_time*sizeof(dcomplex)) ;
  B=(dcomplex*)malloc(nb_time*sizeof(dcomplex)) ;

  M=(int)( ptSwpt->swapMaturity/ptSwpt->tenor);
  s=(int)( ptSwpt->swaptionMaturity/ptSwpt->tenor);
  BT_M=computeZeroCoupon(ptLib,0,M);
  BT_s=computeZeroCoupon(ptLib,0,s);
  sumZc=computeZeroCouponSum(ptLib, 0,s+1,M );
  spot=(BT_s - BT_M)/sumZc;
  w=BT_s - BT_M;
  strike=ptSwpt->strike;
  compute_Omega(ptLib, ptSwpt ,omega);
  compute_alpha(ptLib, ptSwpt ,  alpha);
  build_beta_swaption( beta, ptSwpt,sigma);
  build_lambda_swaption( lambda , ptSwpt ,ptLib ,omega,si
```

```
  gma);
build_zeta_swaption(zeta, ptSwpt , ptLib,sigma );

int_type=1;
init_gauss(ngauss);
init_log(nb_time);
//
//================
tp3 = 0.;
tpi = 1.;
i   = 0;
while( (fabs(tpi)>tol) && (i<N) )
{
  ai    = i*h;
  b    = ai + h;
  tpi  = integrale_gauss(func,ai,b);

  tp3 += tpi;
  i++;
}
if (fabs(tpi)>tol) printf("tpi %f= {n",tpi);
tp1 = tp3/M_PI;
tp1 += 0.5;
//================
//
free_gauss();
free_log();


int_type=2;
init_gauss(ngauss);
init_log(nb_time);
//
//================
tp3 = 0.;
tpi = 1.;
i   = 0;
while( (fabs(tpi)>tol) && (i<N) )
{
  ai    = i*h;
  b    = ai + h;
```

```
    tpi  = integrale_gauss(func,ai,b);
    tp3 += tpi;
    i++;
  }
  if (fabs(tpi)>tol) printf("tpi %f= {n",tpi);

  tp2 = tp3/M_PI;
  tp2 += 0.5;
  //================
  //
  free_gauss();
  free_log();

  price = w*( tp2 - strike/spot * tp1 ) ;
  ptSwpt->price=price;



  // free
  free(beta);
  beta=NULL;
  free(zeta);
  zeta=NULL;
  free(lambda);
  lambda=NULL;
  free(A);
  A=NULL;
  free(g);
  g=NULL;
  free(a);
  a=NULL;
  free(d);
  d=NULL;
  free(B);
  B=NULL;

  return(1);
}


static int ap_wuzhang_lmm_heston1d(double epsilon_lmm,
```

```
    double kappa_lmm,double theta_lmm,double V0_lmm,double rho1,
    double rho2,NumFunc_1 *p,double l0,double t0, double sigma,int
    nb_factors,double swap_maturity,double swaption_maturity,
    int am,double Nominal,double strike,double tenor,double *
    price)
{
  int numMat;
  double* maturities;
  double swaption_price;


  epsilon=epsilon_lmm;

  kappa=kappa_lmm;
  theta=theta_lmm;
  V0=V0_lmm;
  rho1_lmm=rho1;
  rho2_lmm=rho2;
  rho2_lmm=0.;

  swap_maturity=swap_maturity-t0;
  swap_maturity=swap_maturity-t0;

  numMat=(int)(swap_maturity/tenor);
  maturities=(double*)malloc(numMat*sizeof(double));

  /* if ((p->Compute)==&Put) */
  /*   payer_or_receiver=1; */
  /* else */
  /*   if ((p->Compute)==&Call) */
  /*     payer_or_receiver=0; */

   swaption_price=lmm_swaption_payer_stoVol_pricer(tenor,nb
    _factors,swaption_maturity,swap_maturity,strike,sigma,l0);

  *price=Nominal*swaption_price;

   free(maturities);

  return OK;
}
```

```
int CALC(AP_WZ)(void *Opt,void *Mod,PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return ap_wuzhang_lmm_heston1d(ptMod->Sigma2.Val.V_PDOUB
    LE,
         ptMod->MeanReversion.hal.V_PDOUBLE,
         ptMod->LongRunVariance.Val.V_PDOUBLE,
         ptMod->Sigma0.Val.V_PDOUBLE,
         ptMod->Rho1.Val.V_PDOUBLE,
         ptMod->Rho2.Val.V_PDOUBLE,
         ptOpt->PayOff.Val.V_NUMFUNC_1,ptMod->l0.Val.V_
    PDOUBLE,
         ptMod->T.Val.V_DATE,
         ptMod->Sigma.Val.V_PDOUBLE,
         ptMod->NbFactors.Val.V_ENUM.value,
         ptOpt->BMaturity.Val.V_DATE,
         ptOpt->OMaturity.Val.V_DATE,
         ptOpt->EuOrAm.Val.V_BOOL,
         ptOpt->Nominal.Val.V_PDOUBLE,
         ptOpt->FixedRate.Val.V_PDOUBLE,
         ptOpt->ResetPeriod.Val.V_DATE,
         &(Met->Res[0].Val.V_DOUBLE));

}

static int CHK_OPT(AP_WZ)(void *Opt, void *Mod)
{

  if ((strcmp(((Option*)Opt)->Name,"PayerSwaption")==0))
    return OK;
  else
    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
```

```
    {
      Met->init=1;
    }

  return OK;
}

PricingMethod MET(AP_WZ)=
{
  "AP_WuZhang_LMMHeston",
  {
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(AP_WZ),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(AP_WZ),
  CHK_ok,
  MET(Init)
} ;
```

References