Help

```
#include <stdlib.h>
#include "hk1d_stdi.h"
#include "mathsb.h"
#include "currentzcb.h"
#include "hktree.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(TR_BERMUDANSWAPTION)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(TR_BERMUDANSWAPTION)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


static char init[]="initialyield.dat";

/*Swaption=Option on Coupon-Bearing Bond*/
/*All details comments for the functions used here are mai
    nly in "hwtree1dincludes.h" and partially in this file*/
static void HK_iterations( int flat_flag, double r_flat,
    char* init, double a, double sigma_HW,
                        double T0, double per, int m,
    double K0, int xnumber, discrete_fct *N);



////////////////////////////////////////////////////////////
    ///////////////////////////////////
// computes V_0(K), the current Hull-White-price of the
    digital (T,S)-caplet with strike K
////////////////////////////////////////////////////////////
    ///////////////////////////////////
```

```
/*static double HW_DigitalCaplet(double a0, double sigma0,
    double T, double S, double tau0, double P0T, double P0S, double
    K)
  {
  double sigma_P, log_term;

  sigma_P = sigma0 * (exp(-a0*T) - exp(-a0*S))/a0 * sqrt( (
    exp(2*a0*T)-1)/(2*a0) );

  log_term = log( P0T / P0S / (tau0*K+1) );

  return tau0 * P0S * cdf_nor( log_term/sigma_P - sigma_P/2
     );
  }
  */


/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
// solves V_0(K)=x, where V_0(K) is the current Hull-White-
    price of the digital (T,S)-caplet with strike K
/////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
static double Inv_HW_DigitalCaplet(double a0, double sigma0
    , double T, double S, double tau0, double P0T, double P0S,
     double x)
{
  double tau0_inv,sigma_P,exp_term,K;

  tau0_inv = 1/tau0;

  sigma_P = sigma0 * (exp(-a0*T) - exp(-a0*S))/a0 * sqrt( (
    exp(2*a0*T)-1)/(2*a0) );

  exp_term = exp( -SQR(sigma_P)/2 - sigma_P * pnl_inv_cdf
    nor( x / (tau0*P0S) ) );

  K = tau0_inv*P0T/P0S * exp_term - tau0_inv;
  /*if (K<0) printf("Inv_HW_DigitalCaplet returns the negat
    ive strike K = %f{n", K); */
```

```
  return K;
}

/////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////
// solves V_0^{m-1,HK}(K) = V_0^{m-1,HW}(K) in terms of si
    gma_HK, where V_0^{m-1,HK}(K) resp. V_0^{m-1,HK}(K)
// is the current Hunt-Kennedy-price resp. Hull-White
    price of the digital (T_{m-1},T_m)-caplet with strike K
/////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////
static double ComputeSigma_HK(double a, double sigma_HW,
    double Tm_1, double Tm, double tau, double P0Tm_1, double P0Tm,
    double K)
{
  double sigma_HK, R0m_1, p, q, sigma_P;

  R0m_1 = (P0Tm_1/P0Tm - 1.) / tau;
  sigma_P = sigma_HW * (exp(-a*Tm_1) - exp(-a*Tm))/a * sq
    rt( (exp(2.*a*Tm_1)-1)/(2.*a) );
  p   = 2. * log( (R0m_1+tau)/(K+tau) ) / sigma_P  -  sigma
    _P;
  q = -2. * log( R0m_1/K );
  sigma_HK = (sqrt( SQR(p)/4. - q ) - p/2.) / sqrt( (exp(2.
    *a*Tm_1)-1)/(2.*a) );

  return sigma_HK;
}




/////////////////////////////////////////////////////////////
    //////////////////////////////
// functional form of the INVERSE of the numeraire at T[m-1
    ], i.e. of 1/P(T_{m-1}, T_m)
/////////////////////////////////////////////////////////////
    //////////////////////////////
static double N_mminus1(double x, double C_2)
{
```

```
  return 1 + C_2*exp(x);
}

////////////////////////////////////////////////////////////
    ////////////////////////////////
// functional form of the INVERSE of the numeraire at T[m-2
    ], i.e. of 1/P(T_{m-2}, T_m)
////////////////////////////////////////////////////////////
    ////////////////////////////////
/*static double N_mminus2(double a_HW, double sigma_HW,
    double T_mminus2, double T_mminus1, double tau_mminus2, double
    P0T_mminus2,
  double P0T_mminus1, double P0T_m, double C_0, double C_1,
     double Sig, double x)
  {
  double result, J_term, P_term, V0_market_inv;

  P_term = 1 + C_0 * exp(x);
  J_term = P0T_m * tau_mminus2 * ( cdf_nor(-x/Sig) + C_1*
    cdf_nor(-x/Sig+Sig) );
  V0_market_inv =  Inv_HW_DigitalCaplet( a_HW, sigma_HW, T_
    mminus2, T_mminus1, tau_mminus2, P0T_mminus2, P0T_mminus1,
    J_term);

  result = P_term * ( 1 + tau_mminus2 * V0_market_inv);

  return result;
  }*/



////////////////////////////////////////////////////////////
    ///
// returns the variance of the HK-process x_t given x_s
////////////////////////////////////////////////////////////
    ///
static double SigmaSqr( double t, double s, double sigma,
    double a)
{
  return SQR(sigma) * ( exp(2.*a*t) - exp(2.*a*s) )/(2*a);
```

```
}



///////////////////////////////////////////////////////////
    /////
// returns (U_{t,s}f)(x), where U is the semigroup of operators
// corresponding to the HK-process
///////////////////////////////////////////////////////////
    ///////
static double U(double t, double s, discrete_fct *f,
    double x, double sigma, double a)
{
  return NormalTab( x, SigmaSqr(t,s,sigma,a), f);
}



static void SetUf(discrete_fct *g, double t, double s, dis
    crete_fct *f, double sigma, double a)
     // Sets g = U_{t,s}f in a reasonable way
{
  SetNf( g, SigmaSqr(t,s,sigma,a), f);
}



static double UfUpBound (discrete_fct *f, double t, double
    s, double vmax, double sigma, double a)
     // returns the minimum of all x>=f.xleft such that U_{
    t,s}(f*1_{(x,infty)})(0) < vmax
{
  return NfUpBound ( f,  SigmaSqr(t,s,sigma,a), vmax);
}


static double UfLoBound (discrete_fct *f, double t, double
    s, double vmin, double sigma, double a)
     // returns the maximum of all x<f.right such that U_{
    t,s}(f*1_{(x,infty)})(0) > vmin
```

```
{
  return NfLoBound ( f,  SigmaSqr(t,s,sigma,a), vmin);
}

static void HK_iterations( int flat_flag, double r_flat,
    char* init, double a, double sigma_HW,
                           double T0, double per, int m,
    double K0, int xnumber, discrete_fct *N)
// flat_flag           : flag to decide wether initial yi
    eld curve is flat at r_flat (0) or read from the file init (
    1)
// a, sigma_HW         : parameters of the HW-model repres
    enting the market ("a" is common with the HK-process)
// T0                  : first HK-date
// per                 : HK-periodicity
// m                   : number of HK-dates
// K0                  : calibration strike (for the compu
    tation of sigma_HK)
// N                   : functional forms of the INVERSE of
     the numeraire at T[i], i.e. of 1/P(T_i,T_m), for i=0,...,
    m-1
// xnumber             : parameter for the discretization
    of the functional forms
{

  double *T;                        /* HK-dates */
  double *tau;                      /* tau[i] = year fraction
    from T[i] to T[i+1] */
  double *P0;                       /* P0[i] = P(0,T[i]) (ini
    tial zcb prices) */
  double **Sigma;                   /* corresponds to Sigma_{
    T[i],T[j-1]}, where T[-1] denotes 0 */
  /* here SQR(Sigma_{t,s}) is the variance of x_t given x_
    s */
  double C_2;                       /* corresponds to the cons
    tant C_2 in the formula for N[m-1] */


  double x, s, result, J_term, P_term, V0_market_inv;
  double xle, xste, xri, eps;
  double sigma_HK;        /* parameter of the HK-proces */
```

```
double xleft,xstep;      /* parameters for the discretiz
  ation of the functional forms */
int i,j,k;
discrete_fct Ptilde_i, Ptilde_ix;


//////////////////////////////////////////
// initialisation of the main variables //


T = malloc((m+1)*sizeof(double));
for (i=0; i<=m; i++)  T[i] = i * per + T0;
// T[0]=T0           :  first resetting date of the swap
// T[1],...,T[m]     :  payment dates of the swap
// T[0],...,T[n-1]   :  exercise dates of the swaption
  ---> We suppose m>=n !!


tau = malloc(m*sizeof(double));
for(i=0; i<m; i++)  tau[i]=per;


P0 = malloc((m+1)*sizeof(double));
for(i=0; i<=m; i++)  P0[i] = CurrentZCB(T[i], flat_flag,
  r_flat, init);


// computation of sigma_HK
sigma_HK = ComputeSigma_HK(a, sigma_HW, T[m-1], T[m], ta
  u[m-1], P0[m-1], P0[m], K0);


Sigma = malloc(m*sizeof(double*));
for(i=0; i<m; i++)
  {
    Sigma[i] = malloc((i+1)*sizeof(double));
    for(j=0; j<=i; j++)
      {
        if (j==0) s=0; else s=T[j-1];
        Sigma[i][j] = sigma_HK * sqrt( (exp(2*a*T[i])-exp
```

```
  (2*a*s)) / (2*a) );
      }
  }


// constant in the formula for N[m-1]
C_2 = (P0[m-1]/P0[m] - 1) * exp( -SQR(Sigma[m-1][0])/2 );



/////////////////////////////////////////////////////////
  ///////////////
// initialization of N[m-1] (for which we have an explic
  it formula !) //
/////////////////////////////////////////////////////////
  ///////////////
xleft = -SQR(Sigma[m-1][0]) - Sigma[m-1][0]*sqrt(40.);
xstep = 2*fabs(xleft)/(double)xnumber;
Set_discrete_fct( &N[m-1], xleft, xstep, xnumber);
for(j=0; j<N[m-1].xnumber; j++)
  {
    x = N[m-1].xleft + j*N[m-1].xstep;
    N[m-1].val[j] =  N_mminus1( x , C_2 );
  }
/*
   printf("N[%d]{n",m-1); ShowDiscreteFct( &N[m-1] );
   sprintf(filename, "N[%d].txt", m-1);
   SaveDiscreteFctToFile( &N[m-1], filename);
   printf("{ninitial discounted ZCB price for maturity T[
  %d]=%f :{n", m-1, T[m-1]);
   printf("HK: %f{n", U(T[m-1], 0., &N[m-1], 0., sigma_
  HK, a) );
   printf("HW: %f{n{n", P0[m-1]/P0[m]);
   */



//////////////////////////////////////////////////
// iterative computation of the N[m-2],...,N[0] //
//////////////////////////////////////////////////
for(i=m-2; i>=0; i--)
```

```
{
  // printf("beginning for i=%d{n", i); // scanf("%d",
&j);

  ////////////////////////////////////////////////////
  // setting of P^tilde_i := U_{T[i+1],T[i]} N[i+1] //
  ////////////////////////////////////////////////////
  SetUf( &Ptilde_i, T[i+1], T[i], &N[i+1], sigma_HK, a)
;
  // sets Ptilde_i such that domain( Ptilde_i ) = [ U_{
T[i+1],T[i]} N[i+1] > 0 ]

  /*
     printf("Ptilde_i{n"); ShowDiscreteFct( &Ptilde_i )
;
     sprintf(filename, "Ptilde[%d].txt", i);
     SaveDiscreteFctToFile( &Ptilde_i, filename);
     printf("{ninitial discounted ZCB price for maturit
y T[%d]=%f :{n", i+1, T[i+1]);
     printf("HK: %f{n", U(T[i], 0., &Ptilde_i, 0., si
gma_HK, a) );
     printf("HW: %f{n{n", P0[i+1]/P0[m]);
     */


  /////////////////////
  // setting of N[i] //
  /////////////////////
  eps = 0.000001;
  xle = UfUpBound( &Ptilde_i, T[i], 0., P0[i+1]/P0[m]-
eps, sigma_HK, a);
  xri = UfLoBound( &Ptilde_i, T[i], 0., eps, sigma_HK,
a);
  xste=(xri-xle)/(double)(xnumber-1);
  Set_discrete_fct( &N[i], xle, xste, xnumber);

  // printf("N[%d]{n",i); ShowDiscreteFct( &N[i] );



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

```
////////////////////
  // initialization of P^tilde_{i,x} as a (restricted)
copy of P^tilde_i //
  ////////////////////////////////////////////////////
////////////////////
  Set_discrete_fct( &Ptilde_ix, N[i].xleft, N[i].xstep,
 N[i].xnumber+1);
  for(j=0; j<Ptilde_ix.xnumber; j++)
    {
      x = Ptilde_ix.xleft + j*Ptilde_ix.xstep;
      Ptilde_ix.val[j] =  U( T[i+1], T[i], &N[i+1], x,
sigma_HK, a);
    }



  ////////////////////////////////////////////////////
///////////////////////
  // evaluation of N_i in its discretizing points N[i].
xleft + j*N[i].xstep //
  ////////////////////////////////////////////////////
///////////////////////
  for(j=0; j<N[i].xnumber; j++)
    {
      x = N[i].xleft + j*N[i].xstep;   // observe: x =
Ptilde_ix.xleft + j*Ptilde_ix.xstep
      P_term =  Ptilde_ix.val[j];      // hence: P_ter
m = P_i^tilde(x) !!!

      Ptilde_ix.val[j] = 0;
      // VERY IMPORTANT: now Ptilde_ix corresponds rea
lly to P^tilde_{i,x} := P^tilde_i * 1_{(x,infty)} !!!

      J_term = P0[m] * tau[i] * U( T[i], 0., &Ptilde_ix
, 0., sigma_HK, a);

      if (J_term==0) {printf("At j=%d: J_term=0 !{n",
j); scanf("%d", &k); ShowDiscreteFctVal( &Ptilde_ix ); }
      if (J_term/(tau[i]*P0[i+1])>=1)
        {printf("At j=%d: J_term too large !{n", j);
scanf("%d", &k);}
```

```
        V0_market_inv =  Inv_HW_DigitalCaplet( a, sigma_
    HW, T[i], T[i+1], tau[i], P0[i], P0[i+1], J_term);

        result = P_term * ( 1 + tau[i] * V0_market_inv);
        // now we have:  result = N_i(x)

        N[i].val[j] = result;
      }

    /*
       printf("eval. of N[%d] in its discret. points fini
    shed{n", i);
       sprintf(filename, "N[%d].txt", i);
       SaveDiscreteFctToFile( &N[i], filename);
       printf("end for i=%d{n{n",i);
       */

    Delete_discrete_fct(&Ptilde_i);
    Delete_discrete_fct(&Ptilde_ix);

  } // end of i-loop



  ////////////////////////////////
  // free the variables         //

  free(T);
  free(tau);
  free(P0);
  for(i=0; i<m; i++) free(Sigma[i]);
  free(Sigma);

  // end of: free the variables //
  ////////////////////////////////

}

static int bermudanswaption_hk1d(int flat_flag,double a,
```

```
    double t0, double sigma_HW,double r_flat,double T_final,double
    T0,NumFunc_1 *p,int am,double Nominal,double K,double per/*
    ,int n*/,long N_step,int xnumber,double *price/*,double *
    delta*/)
{
  // flat_flag           : flag to decide wether initial yi
    eld curve is flat at r_flat (0) or read from the file init (
    1)
  // a, sigma_HW         : parameters of the HW-model rep
    resenting the market ("a" is common with the HK-process)
  // T0                  : first reset date of the swap (=
    first HK-date)
  // per                 : reset period of the swap (= HK-
    period)
  // n                   : number of exercise dates of the
    swaption
  // m                   : number of payment dates of the
    swap (= number of HK-dates)
  // t0                  : time for which the swaption
    price is computed
  // payer               : payer swaption (1) or receiver
    swaption (0)
  // K                   : strike of the swaption
  // Nominal             : nominal value of the swap
  // N_step              : number of time steps in the tre
    e for the HK-process
  // xnumber             : parameter for the discretization
     of the functional forms

  double *T;                      /* reset/payment dates of
    the swap; exercise dates of the Bermudan swaption; HK-da
    tes */
  double *tau;                    /* tau[i] = year fraction
    from T[i] to T[i+1] */
  double **Sigma;                 /* corresponds to Sigma_{
    T[i],T[j-1]}, where T[-1] denotes 0 */
  /* here SQR(Sigma_{t,s}) is the variance of x_t given x_
    s */
  discrete_fct *N;                /* functional forms of th
    e INVERSE of the numeraire at T[i], */
  /* i.e. of 1/P(T_i,T_m), for i=0,...,m-1 */
```

```
struct Tree Tr;                 /* tree for the HK-proces
  s */


double x, s, **disc_payoff, PtildeTiTk, calib_strike, dis
  c_price;
double sigma_HK;                            /* parameter of
  the HK-proces */
int i,j,k,*ind,*Size,payer_sign;
int m,n,payer;

m=(int)((T_final-T0)/per);
n=m;

if ((p->Compute)==&Put)
  payer=1;
else
  /*if ((p->Compute)==&Call)*/
  payer=0;
//////////////////////////////////////////
// initialisation of the main variables //


T = malloc((m+1)*sizeof(double));
for (i=0; i<=m; i++)  T[i] = i * per + T0;
// T[0]=T0           :  first resetting date of the swap
// T[1],...,T[m]     :  payment dates of the swap
// T[0],...,T[n-1]   :  exercise dates of the swaption
  ---> We suppose m>=n !!


tau = malloc(m*sizeof(double));
for(i=0; i<m; i++)  tau[i]=per;


// comput. of sigma_HK via calibr. to the digital (T[m-1]
  ,T[m])-caplet
// calib_strike = (P0[m-1]/P0[m]-1)/tau[m-1]; // at the
  money case
calib_strike = K;
sigma_HK = ComputeSigma_HK(a, sigma_HW, T[m-1], T[m], ta
```

```
  u[m-1],
                             CurrentZCB(T[m-1], flat_flag,
  r_flat, init),
                             CurrentZCB(T[m]  , flat_flag,
  r_flat, init), calib_strike);


Sigma = malloc(m*sizeof(double*));
for(i=0; i<m; i++)
  {
    Sigma[i] = malloc((i+1)*sizeof(double));
    for(j=0; j<=i; j++)
      {
        if (j==0) s=0; else s=T[j-1];
        Sigma[i][j] = sigma_HK * sqrt( (exp(2*a*T[i])-exp
  (2*a*s)) / (2*a) );
      }
  }


// functional forms of the INVERSE of the numeraire at T[
  i], i.e. of 1/P(T_i,T_m), for i=0,...,m-1
N = malloc(m*sizeof(discrete_fct));


// end of: initialisation of the main variables //
//////////////////////////////////////////////////


/////////////////////////////////////////////////////////
  ////////
// construction of a trinomial tree for the HK-process
        //


// the last exercise date T[n-1] is the final time of th
  e tree, N_step is the number of time steps
SetTimegrid( &Tr, T[n-1], N_step );


// add (if necessary) the exercise dates T[0],...T[n-2]
```

```
  to the time grid of the tree
for (i=0; i<n-1; i++)  AddTime( &Tr, T[i] );


// construct a tree for the HK-process (x_t) given by: dx
  _t = sigma*exp(a*t) dW_t , x_0=0
SetHKtree( &Tr, a, sigma_HK );


// end of: construction of a trinomial tree for the HK-
  process //
/////////////////////////////////////////////////////////
  ////////



ind = malloc(n*sizeof(int));
for (i=0; i<n; i++)
  ind[i] = indiceTime( &Tr, T[i] );
// we have: Tr.t[ ind[i] ]  = T[i]


Size = malloc(n*sizeof(int));
for (i=0; i<n; i++)
  Size[i] = Tr.TSize[ind[i]];
// at T[i], the tree has Size[i] nodes


disc_payoff = malloc(n*sizeof(double*));
// disc_payoff[i] will represent the discounted payoff of
   the payer swaption at T[i] !!
for (i=0; i<n; i++)
  {
    disc_payoff[i] = malloc(Size[i]*sizeof(double));

    for (j=0; j<Size[i]; j++)
      disc_payoff[i][j] = -1 - K*tau[m-1];
  }
// for the moment, disc_payoff[i] represents the constant
   payoff -1-K*tau[m-1]
```

```
// Construct the functional forms N[0],...,N[m-1]
HK_iterations( flat_flag, r_flat, init, a, sigma_HW,
               T0, per, m, K, xnumber, N);


// Complete the discounted payoff of the payer swaption
  in disc_payoff
for(i=0; i<n; i++)
  for(j=0; j<Size[i]; j++)
    {
      x = Tr.pLRij[ ind[i] ][j];
      disc_payoff[i][j]+=InterpolDiscreteFct( &N[i], x );
      for(k=i+1; k<m; k++)
        {
          PtildeTiTk = NormalTab( x, SQR(Sigma[k][i+1]),
  &N[k] );
          disc_payoff[i][j]-=K * tau[k-1] * PtildeTiTk;
        }
    }
// now disc_payoff[i] represents 1/P(T[i],T[m]) - (1+K*ta
  u[m-1]) - K*sum_{k=i+1}^{m-1} tau[k-1] *
// P^tilde(T[i],T[k]) which is the correct discounted
  payoff at T[i] of the payer swaption !!


if (payer) payer_sign=1; else payer_sign=-1;


initPayoff1(&Tr, T[n-1]);
for (i=0; i<n; i++)
  {
    for (j=0; j<Size[i]; j++)
      {
        Tr.Payoffunc[ind[i]][j] = MAX( payer_sign * disc_
  payoff[i][j] , 0 );
      }
  }


// Compute the swaption from the last exercise date T[n-1
```

```
  ] to 0 in Tr.plQij
Computepayoff1(&Tr, T[n-1]);


// return  plQij[0][1]  as discounted price of the swapt
  ion
if (t0==0)
  {
    disc_price = Nominal * Tr.pLQij[0][1];
    //      printf("disc. price = %e{n", disc_price);
    //      *price = P0[m] * disc_price;
    *price = CurrentZCB(T[m], flat_flag, r_flat, init) *
  disc_price;
    // printf("price = %e{n", *price);
  }
else printf("Evaluation in t>0 is not implemented.{n");



////////////////////////////////
// free the variables         //

free(T);
free(tau);
//  free(P0);
for(i=0; i<m; i++)  free(Sigma[i]);
free(Sigma);
for(i=0; i<m; i++)  Delete_discrete_fct(&N[i]);
free(N);

DeletePayoff1(&Tr, T[n-1]);
DeleteTree(&Tr);

for(i=0; i<n; i++)  free(disc_payoff[i]);
free(disc_payoff);
free(ind);
free(Size);

// end of: free the variables //
////////////////////////////////
```

```
  return OK;
}

int CALC(TR_BERMUDANSWAPTION)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return bermudanswaption_hk1d(ptMod->flat_flag.Val.V_INT,
                               ptMod->a.Val.V_DOUBLE,
                               ptMod->T.Val.V_DATE,
                               ptMod->Sigma.Val.V_PDOUBLE,
                               MOD(GetYield)(ptMod),
                               ptOpt->BMaturity.Val.V_DATE,
                               ptOpt->OMaturity.Val.V_DATE,
                               ptOpt->PayOff.Val.V_NUMFUNC_
    1,
                               ptOpt->EuOrAm.Val.V_BOOL,
                               ptOpt->Nominal.Val.V_PDOUB
    LE,
                               ptOpt->FixedRate.Val.V_PDOUB
    LE,
                               ptOpt->ResetPeriod.Val.V_DA
    TE,
                               Met->Par[0].Val.V_LONG,
                               Met->Par[1].Val.V_INT,
                               &(Met->Res[0].Val.V_DOUBLE))
    ;
}


static int CHK_OPT(TR_BERMUDANSWAPTION)(void *Opt, void *
    Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"PayerBermudanSwaption")
    ==0) || (strcmp(((Option*)Opt)->Name,"
    ReceiverBermudanSwaption")==0))
    return OK;
  else
```

```
    return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_LONG=140;
      Met->Par[1].Val.V_INT=1000;


    }
  return OK;
}

PricingMethod MET(TR_BERMUDANSWAPTION)=
{
  "TR_HK1d_BERMUDANSWAPTION",
  {{"TimeStepNumber",LONG,{100},ALLOW},
      {"Parameter for the discretization of the functional
    forms",INT,{100},ALLOW},
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_BERMUDANSWAPTION),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_BERMUDANSWAPTION),
  CHK_ok,
  MET(Init)
} ;
```

References