Help

```
#include  "cir2d_stdi.h"
#include "enums.h"

#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)
static int CHK_OPT(  MC_CIR2D_TEICHMANNBAYER)(void *Opt,
    void *Mod)
{
  return NONACTIVE;
}
  int CALC(MC_CIR2D_TEICHMANNBAYER)(void*Opt,void *Mod,
    PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
/* linear uniform interpolation of [0,T] of size N*/
/* return value = dt*/
static double linspace1(double T0, double T1, int N,
    double* t )
{
  double dt;
  int i;

  dt=(T1 - T0) / (double)( N - 1 );
  t[0] = T0;
  for (i=1; i<N; i++)
    t[i] = t[i-1] + dt;
  return dt;
}
/* linear interpolation using stepsize dt; return T */
static double linspace2( double dt, int N, double* t ){
  double T = dt * (double)( N-1 );
  int i;
  t[0] = 0.0;
  for (i=1; i<N; i++)
    t[i] = t[i-1] + dt;
  return T;
```

```
}

/* extrapolate a CIR-HJM-forward rate curve from a short ra
    te */
/* l is assumed to be the length of x */
static void CIR2HJM( double r0, double kappa, double theta,
     double sigma, double gamma_tb, const double* x, int l,
    double* r ){
  double g0, g1, G1;
  int i;
  for (i=0; i<l; i++ ) {
    G1 = 2.0 * (exp( gamma_tb * x[i] ) - 1.0) / (( gamma_tb
     + kappa ) * (exp( gamma_tb * x[i] ) - 1.0) + 2.0 * gamma_
    tb );
    g0 = kappa * theta * G1;
    g1 = 4.0 * (gamma_tb * gamma_tb) * exp( gamma_tb * x[i]
      ) / (( ( gamma_tb + kappa ) * exp( gamma_tb * x[i] ) +
    gamma_tb - kappa ) * ( ( gamma_tb + kappa ) * exp( gamma_tb
    * x[i] ) + gamma_tb - kappa ));
    r[i] = g0 + r0 * g1;
  }
}

/* very bad random number generator */
static void GenBernoulli2( int** J, int N,int generator )
{
  int i,j;
  for(i=0; i<N; i++)
    for(j=0; j<2; j++)
    if (pnl_rand_uni(generator)< 0.5)
      J[i][j] = 0;
    else  J[i][j] = 1;

}

static  void CopyVect( const double* orig, double* dest,
    int N ){
  int i;
  for (i=0; i<N; i++ )
    dest[i] = orig[i];
}
```

```
static  void omegadot2( int N, double dt, int NCub, int**
    J, int n, double** dB ){
  double tempd1 = sqrt(dt) / sqrt((double)(n));
  int i,k;
  for(i=0; i<(NCub-1); i++ ){
    for (k = 0; k<n; k++ ){
      dB[i*n+k][0] = (J[i][0] == 0) ? tempd1 : (- tempd1);
      dB[i*n+k][1] = (J[i][1] == 0) ? tempd1 : (- tempd1);
    }
  }
  for (k = (n*(NCub-1)); k<N; k++ ){
    dB[k][0] = (J[NCub-1][0] == 0) ? tempd1 : (-tempd1);
    dB[k][1] = (J[NCub-1][1] == 0) ? tempd1 : (-tempd1);
  }
}

static  double Shift( const double* r, const double* x,
    double dx, double dt, int m, int k, int i_shift, double r_shift
    ){
  /*double ret;*/
  if (k < m - i_shift - 1 )
    return (1.0-r_shift) * r[k+i_shift] + r_shift * r[k+i_
    shift+1];
  else
    return r[m-1];
 }

static  double alpha0( const double* r, double kappa,
    double theta, double sigma, double gamma, const double* x, int
    m, int k, double expg){
  double g1, G1;
  G1 = 2.0 * (expg - 1.0) / (( gamma + kappa ) * (expg - 1.
    0) + 2.0 * gamma );
  g1 = 4.0 * (gamma * gamma) * expg / (( ( gamma + kappa )
    * expg + gamma - kappa ) * ( ( gamma + kappa ) * expg +
    gamma - kappa ));
  return (sigma * sigma) * (g1) * ( r[0] * (G1) - 0.25 );
}

static  double HJMSigma( const double* r, double kappa,
```

```
    double theta, double sigma, double gamma, const double* x, int
    m, int k, double expg, double sqrtr ){
  return sigma * sqrtr * ( 4.0 * (gamma * gamma) * expg / (
    ( ( gamma + kappa ) * expg + gamma - kappa ) * ( ( gamma +
     kappa ) * expg + gamma - kappa )));
}


/* value P(0,T) of a zero coupon bond */
static double ZeroCB( const double* r, const double* x,
    double dx, double T ){
  int Tx = ceil( T / dx ); /* index of T in the x-grid */
  double integ = 0.0;
  int i;
  for (i=0; i<Tx; i++ )
    integ += 0.5 * ( r[i] + r[i+1] ) * dx;
  return exp( - integ );
}

/* compute the empirical mean value of a vector */
static double mean( const double* X, int M ){
  double ret = 0.0;
  int i;
  for (i=0; i<M; i++ )
    ret += X[i];
  ret = ret / (double)(M);
  return ret;
}

/* compute the empirical standard deviation of a vector */
static double stdev( const double* X, int M ){
  double mu = mean( X, M );
  double ret = 0.0;
  int i;
  for(i=0; i<M; i++ )
    ret += X[i]*X[i];
  ret = ret / (double)(M);
  ret = sqrt( ret - mu * mu );
  return ret;
}
```

```
/* n number of time intervals on each cubature interval*/
/* N number of time intervals*/
/* m number of space intervals*/
/* M number of paths for Monte-Carlo simulation*/
static int mc_cir2d_teichmannbayer(double x01,double x02,
    double k1,double k2,double sigma1,double sigma2,double theta1,
    double theta2,double shift,double t0,  double T_bond, double T_
    option,NumFunc_1 *p,int generator,int n,int L,int k, int M,
    double *price,double *error)
{

  double delta;
  double kappa[2],theta[2],sigma[2],r0[2];
  double gamma[2];
  int NCub;
  double *t, dt, *x, dx, r_shift;
  int mAct, i_shift;
  int i;
  int **J;
  double *r1,*r2;
  double *rp,*rm,*rp1,*rm1,*rpc1,*rmc1,*rp2,*rm2,*rpc2,*rm
    c2;
  double *res;
  double Bp, Bm; /* savings account */
  double expg1,  expg2, sqrtrm1, sqrtrp1, sqrtrm2, sqrtrp2;
     /* auxiliary variables */
  int j;
  int k_bis;
  double zerop, zerom;
  double **dB;

  pnl_rand_init(generator,1,M);

  kappa[0]=k1;
  kappa[1]=k2;
  theta[0]=theta1;
  theta[1]=theta2;
  sigma[0]=sigma1;
  sigma[1]=sigma2;
```

```
r0[0]=x01;
r0[1]=x02;
delta=shift;
gamma[0] = sqrt( (kappa[0]*kappa[0]) + 2.0 * (sigma[0] *
  sigma[0]) );
gamma[1] = sqrt( (kappa[1]*kappa[1]) + 2.0 * (sigma[1] *
  sigma[1]) );



if (L % n == 0)
  NCub = L / n;
else
  NCub = L / n + 1;

/* generate the time grid */
    t=malloc((L+1)*sizeof(double));
    dt = linspace1( t0, T_option, L+1, t );
    /* generate the spacegrid */
    dx = ( T_bond - T_option ) / (double)( k );
    mAct = (int)( T_option / dx ) + k + (int)(L * (dt/dx)
  ) + 1;
    x=malloc((mAct+1)*sizeof(double));
    dt = linspace1( t0, T_option, L+1, t );
    linspace2( dx, mAct + 1, x );
    /* for the shift semigroup, express dt in temrs of dx
  :
     dt = i_shift * dx + r_shift * dx */
    r_shift = dt / dx;
    i_shift = (int)( r_shift );
    r_shift = r_shift - i_shift;

    /* J describes one cubature path */
    J=(int **)calloc(NCub,sizeof(int *));
    for (i=0;i<NCub;i++)
    J[i]=(int *)calloc(2,sizeof(int));

/* generate the initial forward rate curve */
r1=malloc((mAct+1)*sizeof(double));/* saves initial
                                    * forward rate curve
*/
```

```
 r2=malloc((mAct+1)*sizeof(double));/* saves initial forw
  ard rate curve */

CIR2HJM( r0[0], kappa[0], theta[0], sigma[0], gamma[0], x
  , mAct+1, r1 );
CIR2HJM( r0[1], kappa[1], theta[1], sigma[1], gamma[1], x
  , mAct+1, r2 );


    rp=malloc((mAct+1)*sizeof(double));
    rm=malloc((mAct+1)*sizeof(double));
    rp1=malloc((mAct+1)*sizeof(double));
    rm1=malloc((mAct+1)*sizeof(double));
    rpc1=malloc((mAct+1)*sizeof(double));
    rmc1=malloc((mAct+1)*sizeof(double));
    rp2=malloc((mAct+1)*sizeof(double));
    rm2=malloc((mAct+1)*sizeof(double));
    rpc2=malloc((mAct+1)*sizeof(double));
    rmc2=malloc((mAct+1)*sizeof(double));

    /* the path-wise discounted payoff */
    res =malloc((M)*sizeof(double));

    /* the "brownian" increments (i.e. the cubature deriv
  atives) */
    dB=(double **)calloc(L,sizeof(double *));
    for (i=0;i<L;i++)
    dB[i]=(double *)calloc(2,sizeof(double));

/* now iterate through all paths for the MC-simulation*/
for(j=0; j<M; j++ ){
  /* re-initialize r and B */
  GenBernoulli2( J, NCub,generator ); /* generate J */
  CopyVect( r1, rp1, mAct+1 );
  CopyVect( r1, rm1, mAct+1 );
  CopyVect( r2, rp2, mAct+1 );
  CopyVect( r2, rm2, mAct+1 );
  Bp = 1.0;
  Bm = 1.0;

  /* generate dB */
```

```
  omegadot2( L, dt, NCub, J, n, dB );

  /* iterate through the time grid */

  for(i=0; i<L; i++ ){
    sqrtrp1 = (rp1[0] > 0.0) ? sqrt(rp1[0]) : 0.0;
    sqrtrm1 = (rm1[0] > 0.0) ? sqrt(rm1[0]) : 0.0;
    sqrtrp2 = (rp2[0] > 0.0) ? sqrt(rp2[0]) : 0.0;
    sqrtrm2 = (rm2[0] > 0.0) ? sqrt(rm2[0]) : 0.0;
    Bp += Bp * (delta + rp1[0] + rp2[0]) * dt;
    Bm += Bm * (delta + rm1[0] + rm2[0]) * dt;
    CopyVect( rp1, rpc1, mAct+1 );
    CopyVect( rm1, rmc1, mAct+1 );
    CopyVect( rp2, rpc2, mAct+1 );
    CopyVect( rm2, rmc2, mAct+1 );
    /* iterate through the space grid */
    for (k_bis=0; k_bis<=mAct; k_bis++){
expg1 = exp( gamma[0] * x[k_bis] );
expg2 = exp( gamma[1] * x[k_bis] );
rp1[k_bis] = Shift( rpc1, x, dx, dt, mAct+1, k_bis, i_sh
  ift, r_shift ) + alpha0( rpc1, kappa[0], theta[0], sigma[0]
  , gamma[0], x, mAct+1, k_bis, expg1 ) * dt;
rp1[k_bis] += HJMSigma( rpc1, kappa[0], theta[0], sigma[
  0], gamma[0], x, mAct+1, k_bis, expg1, sqrtrp1 ) * dB[i][0
  ];
rm1[k_bis] = Shift( rmc1, x, dx, dt, mAct+1, k_bis, i_sh
  ift, r_shift ) + alpha0( rmc1, kappa[0], theta[0], sigma[0]
  , gamma[0], x, mAct+1, k_bis, expg1 ) * dt;
rm1[k_bis] -= HJMSigma( rmc1, kappa[0], theta[0], sigma[
  0], gamma[0], x, mAct+1, k_bis, expg1, sqrtrm1 ) * dB[i][0
  ];
rp2[k_bis] = Shift( rpc2, x, dx, dt, mAct+1, k_bis, i_sh
  ift, r_shift ) + alpha0( rpc2, kappa[1], theta[1], sigma[1]
  , gamma[1], x, mAct+1, k_bis, expg2 ) * dt;
rp2[k_bis] += HJMSigma( rpc2, kappa[1], theta[1], sigma[
  1], gamma[1], x, mAct+1, k_bis, expg2, sqrtrp2 ) * dB[i][1
  ];
rm2[k_bis] = Shift( rmc2, x, dx, dt, mAct+1, k_bis, i_sh
  ift, r_shift ) + alpha0( rmc2, kappa[1], theta[1], sigma[1]
  , gamma[1], x, mAct+1, k_bis, expg2 ) * dt;
rm2[k_bis] -= HJMSigma( rmc2, kappa[1], theta[1], sigma[
```

```
    1], gamma[1], x, mAct+1, k_bis, expg2, sqrtrm2 ) * dB[i][1
    ];
      }
    }

    /* Now combine the three components, delta, r1, r2 into
     the forward rate */
    /* curve r. */
    for (k_bis=0; k_bis<=mAct; k_bis++){
      rp[k_bis] = delta + rp1[k_bis] + rp2[k_bis];
      rm[k_bis] = delta + rm1[k_bis] + rm2[k_bis];
    }
    /* compute the discounted payoff for this particular
    path */
     /* compute the discounted payoff for this particular
    path */
        zerop = ZeroCB(rp, x, dx, T_bond - T_option );
        zerom = ZeroCB(rm, x, dx, T_bond - T_option );

        res[j]=0.5*((p->Compute)(p->Par,zerop)/Bp+(p->Compu
    te)(p->Par,zerom)/ Bm);
  }
  *price = mean( res, M );
  *error = 1.65 * stdev( res, M ) / sqrt( (double)(M) );

  /* free memory */
        free(t);
        free(x);
        free(r1);
        free(r2);
        free(rp);
        free(rm);
        free(rp1);
        free(rpc1);
        free(rm1);
        free(rmc1);
        free(rp2);
        free(rpc2);
        free(rm2);
        free(rmc2);
        free(res);
```

```
        for (i=0;i<L;i++)
          free(dB[i]);
        free(dB);

        for (i=0;i<NCub;i++)
          free(J[i]);
        free(J);


  return  OK;
}

int CALC(MC_CIR2D_TEICHMANNBAYER)(void *Opt,void *Mod,Prici
    ngMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return mc_cir2d_teichmannbayer(ptMod->x01.Val.V_PDOUBLE,
    ptMod->x02.Val.V_PDOUBLE,ptMod->k1.Val.V_DOUBLE,ptMod->k2.
    Val.V_DOUBLE,ptMod->Sigma1.Val.V_PDOUBLE,ptMod->Sigma2.Val.
    V_PDOUBLE,ptMod->theta1.Val.V_PDOUBLE,ptMod->theta2.Val.V_
    PDOUBLE,ptMod->shift.Val.V_PDOUBLE,ptMod->T.Val.V_DATE,pt
    Opt->BMaturity.Val.V_DATE,ptOpt->OMaturity.Val.V_DATE,ptOpt->
    PayOff.Val.V_NUMFUNC_1, Met->Par[0].Val.V_ENUM.value,Met->
    Par[1].Val.V_PINT,Met->Par[2].Val.V_PINT,Met->Par[3].Val.V_PI
    NT,Met->Par[4].Val.V_PINT,&(Met->Res[0].Val.V_DOUBLE),&(
    Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(MC_CIR2D_TEICHMANNBAYER)(void *Opt, voi
    d *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"ZeroCouponCallBondEuro"
    )==0)|| (strcmp(((Option*)Opt)->Name,"ZeroCouponPutBondEu
    ro")==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_ENUM.value=0;
      Met->Par[0].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[1].Val.V_PINT=20;
      Met->Par[2].Val.V_PINT=400;
      Met->Par[3].Val.V_PINT=10;
      Met->Par[4].Val.V_PINT=20;
    }

  return OK;
}

PricingMethod MET(MC_CIR2D_TEICHMANNBAYER)=
{
  "MC_CIR2D_TEICHMANNBAYER",

  {{"RandomGenerator",ENUM,{100},ALLOW},
   {"Number of time intervals on each cubature interval",
    INT,{100},ALLOW},
   {"Number of time intervals",INT,{100},ALLOW},
   {"Number of space intervals*",INT,{100},ALLOW},
   {"Number of paths for Monte-Carlo simulation",PINT,{100}
    ,ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_CIR2D_TEICHMANNBAYER),
  {{"Price",DOUBLE,{100},FORBID},{"MC Error",DOUBLE,{100},
    FORBID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_CIR2D_TEICHMANNBAYER),
  CHK_ok,
  MET(Init)
} ;
```

References