```
   Help
#include <stdlib.h>
#include  "cir2d_stdi.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_SWAPTION)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_SWAPTION)(void *Opt,void *Mod,PricingMethod *
    Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double *C,*T;
static double *sigma,*theta,*r,*e,*k,*x0,delta;
static int N_coupon,d;

/*Function for ZCB computation*/
static double B_i(int i,double t)
{
  return 2*(exp(r[i]*t) - 1)/((k[i] + r[i])*(exp(r[i]*t) -
    1) + 2*r[i]);
}

static double B_0(double t)
{
  double s = 0.0;
  int i;
  for (i= 1; i<= d; i++)
    s = s + (2*k[i]*theta[i]*t/(r[i] - k[i]) - 2*k[i]*thet
    a[i]/sigma[i]/sigma[i]*log(((k[i] + r[i])*(exp(r[i]*t) - 1)
     + 2*r[i])/2/r[i]));

  return -delta*t + s;
}
```

```
/*Improved Euler scheme approximation*/
int approximate(int n,int j,double t,int generator,double *
    x_value,double *in)
{
  double x_T0,delta_t,s,a,in_value,g;
  int i;

  delta_t=(T[0]-t)/(double)n;
  x_T0 = x0[j];
  a=k[j]*theta[j];
  in_value = 0.0;
  for (i= 1; i<= n; i++)
    {
      g= pnl_rand_normal(generator);
      s = (1-k[j]/2.0*delta_t)*sqrt(x_T0) + sigma[j]*sqrt(
    delta_t)*g/2.0/(1-k[j]/2.0*delta_t);
      x_T0 = s*s + (a - sigma[j]*sigma[j]/4.0)*delta_t;
      in_value = in_value + x_T0*delta_t;
    }

  *in=in_value;
  *x_value=x_T0;

  return OK;

}

/*Zero coupon Bond Prices*/
static double P(double t,double Ti,double *x)
{
  double s = 0.0;
  int j;
  for (j= 1; j<= d; j++)
    s = s + B_i(j,Ti - t)*x[j];

  return exp(B_0(Ti - t) - s);
}

/*Coupon Bond Prices*/
double CB_T0(int M,double t,int generator)
```

```
{
  int i;
  double cb,*g;
  double x_value,in;

  g = malloc((d+1)*sizeof(double));
  cb=0.;

  for(i=1;i<=d;i++)
    {
      approximate(M,i,t,generator,&x_value,&in);
      g[i]=x_value;
      e[i]=in;
    }
  for(i=1;i<=N_coupon;i++)
    cb=cb+C[i]*P(T[0],T[i],g);

  free(g);

  return cb;
}


/*Computation of Swaption with Monte Carlo Method*/
static int price_compute_MC(NumFunc_1  *p,double t,double
    K,int M,long N_MC,int generator,double confidence,double *
    price,double *error_price,double *inf_price,double *sup_price)
{
  double s;
  int n,init_mc;
  double mean_price,var_price,alpha,z_alpha;
  int simulation_dim= 1;

  /* Value to construct the confidence interval */
  alpha= (1.- confidence)/2.;
  z_alpha= pnl_inv_cdfnor(1.- alpha);

  mean_price= 0.0;
  var_price= 0.0;

  /*MC sampling*/
  init_mc= pnl_rand_init(generator,simulation_dim,N_MC);

  /* Test after initialization for the generator */
  if(init_mc == OK)
    {

      for (n= 0; n< N_MC; n++)
        {
          s = CB_T0(M,t,generator);
          p->Par[0].Val.V_DOUBLE=1.;
          mean_price =mean_price + exp(-delta*(T[0]-t)-e[1]
    -e[2])*(p->Compute)(p->Par,s);
          var_price = var_price + exp(-2*delta*(T[0]-t)-2*
    e[1]-2*e[2])*(p->Compute)(p->Par,s)*(p->Compute)(p->Par,s);

        }
    }

  /*Price*/
  *price = mean_price/(double)N_MC;
  *error_price = sqrt(var_price/(double)N_MC-SQR(*price))/
    sqrt((double)N_MC);;
  *inf_price=*price-z_alpha*(*error_price);
  *sup_price=*price+z_alpha*(*error_price);

  return OK;
}

/*Swaption=Option on Coupon-Bearing Bond*/
static int mc_swaption_cir2d(NumFunc_1  *p,double t0,
    double x01,double x02,double k1,double k2,double sigma11,double
    sigma22,double theta1,double theta2,double shift,double t_
    op,double swap_maturity,double Nominal,double K,double perio
    dicity,long N_MC,int M, int generator,double confidence,
    double *price, double *error_price,double *inf_price, double *su
    p_price)
{
  int i;
  double first_payement;

  /*dimension*/
```

```
d=2;

/*Parameters of the model*/
theta = malloc((d + 1)*sizeof(double));
sigma = malloc((d + 1)*sizeof(double));
k = malloc((d + 1)*sizeof(double));
x0 = malloc((d + 1)*sizeof(double));

theta[1]=theta1;
theta[2]=theta2;
sigma[1]=sigma11;
sigma[2]=sigma22;
k[1]=k1;
k[2]=k2;
x0[1]=x01;
x0[2]=x02;
delta=shift;

/*Auxiliary Parameters*/
r = malloc((d + 1)*sizeof(double));
e = malloc((d+1)*sizeof(double));
r[1] = sqrt(k[1]*k[1] + 2.0*sigma[1]*sigma[1]);
r[2] = sqrt(k[2]*k[2] + 2.0*sigma[2]*sigma[2]);

/*Compute Coupon Bearing*/
first_payement=t_op+periodicity;
N_coupon=(int)((swap_maturity-first_payement)/periodicity
  )+1;
T = malloc((N_coupon + 1)*sizeof(double));
C = malloc((N_coupon + 1)*sizeof(double));

/*Payement dates*/
T[0]=t_op;
for (i=1; i<= N_coupon; i++)
  T[i] = T[i-1]+ periodicity;

/*Coupon*/
for (i= 1; i< N_coupon; i++)
  C[i] = Nominal*K*periodicity;
C[N_coupon]=Nominal*(1.+K*periodicity);
```

```
  /*Price Computation*/
  price_compute_MC(p,t0,K,M,N_MC,generator,confidence,
    price,error_price,inf_price,sup_price);

  free(theta);
  free(sigma);
  free(k);
  free(x0);
  free(r);
  free(e);
  free(T);
  free(C);

  return OK;
}

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

  return mc_swaption_cir2d(ptOpt->PayOff.Val.V_NUMFUNC_1,pt
    Mod->T.Val.V_DATE,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->sh
    ift.Val.V_PDOUBLE,ptOpt->OMaturity.Val.V_DATE,ptOpt->BMatu
    rity.Val.V_DATE,ptOpt->Nominal.Val.V_PDOUBLE,ptOpt->FixedRa
    te.Val.V_PDOUBLE,ptOpt->ResetPeriod.Val.V_DATE,Met->Par[0].
    Val.V_LONG,Met->Par[1].Val.V_INT, Met->Par[2].Val.V_ENUM.val
    ue,Met->Par[3].Val.V_PDOUBLE,&(Met->Res[0].Val.V_DOUBLE),&(
    Met->Res[1].Val.V_DOUBLE),&(Met->Res[2].Val.V_DOUBLE),&(Met->
    Res[3].Val.V_DOUBLE));
}

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

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

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


      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_INT=500;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumRNGs;

      Met->Par[3].Val.V_DOUBLE= 0.95;


    }

  type_generator= Met->Par[2].Val.V_ENUM.value;



  if(pnl_rand_or_quasi(type_generator)==PNL_QMC)
    {
      Met->Res[1].Viter=IRRELEVANT;
      Met->Res[2].Viter=IRRELEVANT;
      Met->Res[3].Viter=IRRELEVANT;

    }
  else
    {
      Met->Res[1].Viter=ALLOW;
      Met->Res[2].Viter=ALLOW;
      Met->Res[3].Viter=ALLOW;
    }
  return OK;
```

```
}

PricingMethod MET(MC_SWAPTION)=
{
  "MC_Cir2d_Swaption",
  {{"N iterations",LONG,{100},ALLOW},{"TimeStepNumber",INT2
    ,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_SWAPTION),
  {{"Price",DOUBLE,{100},FORBID},{"Error Price",DOUBLE,{100
    },FORBID},
   {"Inf Price",DOUBLE,{100},FORBID},
   {"Sup Price",DOUBLE,{100},FORBID} ,{" ",PREMIA_NULLTYPE,
    {0},FORBID}},
  CHK_OPT(MC_SWAPTION),
  CHK_ok,
  MET(Init)
} ;
```

References