```
   Help
#include <stdlib.h>
#include  "cir2d_stdi.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(AP_SWAPTION)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_SWAPTION)(void *Opt,void *Mod,PricingMethod *
    Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double *C,*T;
static double *lambda1,*lambda2,*sigma,*theta,*r,*k,*x0,de
    lta;
static int N_coupon,d,M;

/*Function for ZCB computation*/
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]);
}

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;
}
```

```
/*Coefficient for moments computation*/
static double Ci(int *I,int m)
{
  double s = 1.0;
  int j;
  for (j= 1; j<= m; j++)    s = s*C[I[j]];

  return s;
}

static double F_i(int i,int *I,int m)
{
  double s = 0.0;
  int j;
  for (j= 1; j<= m; j++)    s = s + B_i(i,T[I[j]] - T[0])
    ;

  return s;
}

static double F_0(int *I,int m)
{
  double s = 0.0;
  int j;
  for (j= 1; j<= m; j++)    s = s + B_0(T[I[j]] - T[0]);

  return s;
}

/*Coefficient for Laplace transform computation*/
static double F_etoile(int i,int *I,int m,double W)
{
  return F_i(i,I,m) + B_i(i,W - T[0]);
}

static double F_etoile0(int *I,int m,double W)
{
  return F_0(I,m) + B_0(W - T[0]);
}
```

```
static double N_i(double t,int i,int *I,double W,int m)
{
  double a,b;
  a = F_etoile(i,I,m,W)*(lambda1[i]*exp(r[i]*t) - lambda2[
    i]) + 2/sigma[i]/sigma[i]*(exp(r[i]*t) - 1);
  b = F_etoile(i,I,m,W)*(exp(r[i]*t) - 1) - (lambda2[i]*exp
    (r[i]*t) - lambda1[i]);

  return a/b;
}

static double M_W(double t,double W,int *I,int m)
{
  double s = 0.0,r1;
  int i;

  for (i= 1; i<= d; i++)
    {
      r1 = ((F_etoile(i,I,m,W) - lambda2[i])*exp(r[i]*t) -
    (F_etoile(i,I,m,W) - lambda1[i]))/(2*r[i]/sigma[i]/sigma[
    i]);
      s = s + 2*k[i]*theta[i]*t/(r[i] - k[i]) - 2*k[i]*thet
    a[i]/sigma[i]/sigma[i]*log(r1);
    }

  return F_etoile0(I,m,W) - delta*t + s;
}

/*Zero coupon Bond Prices*/
static double P(double t,double Ti)
{
  double s = 0.0;
  int j;
  for (j= 1; j<= d; j++)
    s = s + B_i(j,Ti - t)*x0[j];

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


/*Laplace Transform*/
```

```
static double L(double t,double W,int *I,int m)
{
  double s = 0.0;
  int i;
  for (i= 1; i<= d; i++)
    s = s + N_i(T[0] - t,i,I,W,m)*x0[i];

  return exp(M_W(T[0] - t,W,I,m) - s)/P(t,W);
}

/*Moments computations*/
static double moment1(double t,double W)
{
  int *I;
  double mu = 0.0;
  I = malloc(2*sizeof(int));
  for (I[1]= 1; I[1]<= N_coupon; I[1]++)
    mu = mu + Ci(I,1)*L(t,W,I,1);

  return mu;
}

static double moment2(double t,double W)
{
  int *I;
  double mu = 0.0;
  I = malloc(3*sizeof(int));
  for (I[1]= 1; I[1]<= N_coupon; I[1]++)
    for (I[2]= 1; I[2]<= N_coupon; I[2]++)
      mu = mu + Ci(I,2)*L(t,W,I,2);

  return mu;
}

static double moment3(double t,double W)
{
  int *I;
  double mu = 0.0;
  I = malloc(4*sizeof(int));
  for (I[1]= 1; I[1]<= N_coupon; I[1]++)
    for (I[2]= 1; I[2]<= N_coupon; I[2]++)
```

```
      for (I[3]= 1; I[3]<= N_coupon; I[3]++)
  mu = mu + Ci(I,3)*L(t,W,I,3);

  return mu;
}

static double moment4(double t,double W)
{
  int *I;
  double mu = 0.0;
  I = malloc(5*sizeof(int));
  for (I[1]= 1; I[1]<= N_coupon; I[1]++)
    for (I[2]= 1; I[2]<= N_coupon; I[2]++)
      for (I[3]= 1; I[3]<= N_coupon; I[3]++)
  for (I[4]= 1; I[4]<= N_coupon; I[4]++)
    mu = mu + Ci(I,4)*L(t,W,I,4);

  return mu;
}

static double moment5(double t,double W)
{
  int *I;
  double mu = 0.0;

  I = malloc(6*sizeof(int));
  for (I[1]= 1; I[1]<= N_coupon; I[1]++)
    for (I[2]= 1; I[2]<= N_coupon; I[2]++)
      for (I[3]= 1; I[3]<= N_coupon; I[3]++)
  for (I[4]= 1; I[4]<= N_coupon; I[4]++)
    for (I[5]= 1; I[5]<= N_coupon; I[5]++)
      mu = mu + Ci(I,5)*L(t,W,I,5);

  return mu;
}

static double coeff(double t,double W,double K)
{
  double *c,r1,*temp,*Gamma,*Lambda,*mu;
  int m;
```

```
/*Memory allocation*/
mu = malloc((M + 1)*sizeof(double));
c = malloc((M + 1)*sizeof(double));
temp = malloc((M + 1)*sizeof(double));
Gamma = malloc((M + 1)*sizeof(double));
Lambda = malloc((M + 1)*sizeof(double));

/*Moments parameters*/
mu[1] = moment1(t,W);
mu[2] = moment2(t,W);
mu[3] = moment3(t,W);
mu[4] = moment4(t,W);
mu[5] = moment5(t,W);

/*Cumulatants*/
c[1] = mu[1];
c[2] = mu[2]-pow(mu[1],2);
c[3] = mu[3]-3.0*mu[1]*mu[2]+2.0*pow(mu[1],3);
c[4] = mu[4]-4.0*mu[1]*mu[3]-3.0*pow(mu[2],2)+12.0*pow(mu
  [1],2)*mu[2]-6.0*pow(mu[1],4);
c[5] = mu[5]-5.0*mu[1]*mu[4]-10.0*mu[2]*mu[3]+20.0*pow(mu
  [1],2)*mu[3]+30.0*mu[1]*pow(mu[2],2)-60.0*pow(mu[1],3)*mu[
  2]+24.0*pow(mu[1],5);
c[6]=0.0;
c[7]=0.0;

/*Lambda Terms of Expansion*/
temp[1] = cdf_nor((c[1]-K)/sqrt(c[2]));
temp[2] = 1.0/sqrt(2*M_PI*c[2])*exp(-pow((K-c[1]),2)/(2*
  c[2]));
Lambda[0] = temp[1];
Lambda[1] = temp[2]*c[2];
Lambda[2] = c[2]*temp[1]+temp[2]*(c[2]*(K-c[1]));
Lambda[3] = temp[2]*(c[2]*pow((K-c[1]),2)+2*pow(c[2],2));
Lambda[4] = 3.0*pow(c[2],2)*temp[1]+temp[2]*(c[2]*pow((K-
  c[1]),3)+3*pow(c[2],2)*(K-c[1]));
Lambda[5] = temp[2]*(c[2]*pow((K-c[1]),4)+4.0*pow(c[2],2)
  *pow((K-c[1]),2)+8.0*pow(c[2],3));
Lambda[6] = 15.0*pow(c[2],3)*temp[1]+temp[2]*(c[2]*pow((
  K-c[1]),5)+5*pow(c[2],2)*pow(K-c[1],3)+15.0*pow(c[2],3)*(K-
  c[1]));
```

```
  Lambda[7] = temp[2]*(c[2]*pow((K-c[1]),6)+6.0*pow(c[2],2)
    *pow((K-c[1]),4)+24.0*pow(c[2],3)*pow((K-c[1]),2)+48.0*po
    w(c[2],4));

  /*Gamma Terms of expansion*/
  temp[3] = c[3]/(3*2*1);
  temp[4] = c[4]/(4*3*2*1);
  temp[5] = c[5]/(5*4*3*2*1);
  temp[6] = c[6]/(6*5*4*3*2*1);
  temp[7] = c[7]/(7*6*5*4*3*2*1);
  Gamma[0] = 1.0+3.0/pow(c[2],2)*temp[4]-15.0/pow(c[2],3)*(
    temp[6]+0.5*pow(temp[3],2));
  Gamma[1] = -3.0/pow(c[2],2)*temp[3]+15.0/pow(c[2],3)*
    temp[5]-105.0/pow(c[2],4)*(temp[7]+temp[3]*temp[4]);
  Gamma[2] = -6.0/pow(c[2],3)*temp[4]+45.0/pow(c[2],4)*(
    temp[6]+0.5*pow(temp[3],2));
  Gamma[3] = 1.0/pow(c[2],3)*temp[3]-10.0/pow(c[2],4)*temp[
    5]+105.0/pow(c[2],5)*(temp[7]+temp[3]*temp[4]);
  Gamma[4] = 1.0/pow(c[2],4)*temp[4]-15.0/pow(c[2],5)*(
    temp[6]+0.5*pow(temp[3],2));
  Gamma[5] = 1.0/pow(c[2],5)*temp[5]-21.0/pow(c[2],6)*(
    temp[7]+temp[3]*temp[4]);
  Gamma[6] = 1.0/pow(c[2],6)*(temp[6]+0.5*pow(temp[3],2));
  Gamma[7] = 1.0/pow(c[2],7)*(temp[7]+temp[3]*temp[4]);

  /*Probabilty of exercise*/
  r1 = 0;
  for (m= 0; m<= M; m++)
    r1 = r1 + Gamma[m]*Lambda[m];

  return r1;
}


/*Computation of Swaption with Approximation using Laplace
    Transform*/
static double price_compute(double t,double K)
{
  double swap = 0.0,r1,r2;
  int i;
```

```
  /*Ordre of expansion*/
  M=7;

  for (i= 1; i<= N_coupon; i++)
    {
      r1 = coeff(t,T[i],K);
      swap = swap + C[i]*P(t,T[i])*r1;
    }

  r2 = coeff(t,T[0],K);
  swap = swap - K*P(t,T[0])*r2;

  return swap;
}

/*Swaption=Option on Coupon-Bearing Bond*/
static int ap_swaption_cir2d(double t0,double x01,double x0
    2,double k1,double k2,double sigma11,double sigma22,
    double theta1,double theta2,double shift,double t_op,double swa
    p_maturity,double Nominal,double K,double periodicity,
    double *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));
  lambda1 = malloc((d + 1)*sizeof(double));
  lambda2 = 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]);
  lambda1[1] = (-k[1] + r[1])/sigma[1]/sigma[1];
  lambda1[2] = (-k[2] + r[2])/sigma[2]/sigma[2];
  lambda2[1] = (-k[1] - r[1])/sigma[1]/sigma[1];
  lambda2[2] = (-k[2] - r[2])/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=price_compute(t0,1.);

  return OK;
}

int CALC(AP_SWAPTION)(void *Opt,void *Mod,PricingMethod *
    Met)
```

```
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;


  return ap_swaption_cir2d(ptMod->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->shift.Val.V_PDOUBLE,ptOpt->OM
    aturity.Val.V_DATE,
         ptOpt->BMaturity.Val.V_DATE,ptOpt->Nominal.Val.
    V_PDOUBLE,ptOpt->FixedRate.Val.V_PDOUBLE,ptOpt->ResetPerio
    d.Val.V_DATE,&(Met->Res[0].Val.V_DOUBLE));
}


static int CHK_OPT(AP_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)
{
  if ( Met->init == 0)
    {
      Met->init=1;
    }

  return OK;
}

PricingMethod MET(AP_SWAPTION)=
{
  "AP_Cir2d_Swaption",
  {{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(AP_SWAPTION),
```

```
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(AP_SWAPTION),
  CHK_ok,
  MET(Init)
} ;
```

References