```
   Help
#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)
#else

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pnl/pnl_mathtools.h"
#include"lmm_numerical.h"

double BSFormula(Swaption *ptSwpt, Libor* ptLib, double ev
    alTime, double blackVol)
{ /*pricing a swaption in the black model*/

  double Sigma;
  double borne=7.;
  double d1;
  double d2;
  double sum;
  int o,s,m;
  double underlying;


  s=(int)(ptSwpt->swaptionMaturity/ptLib->tenor);
  m=(int)(ptSwpt->swapMaturity/ptLib->tenor);
  o=(int)(evalTime/ptLib->tenor);

  sum=computeZeroCouponSum(ptLib, o , s+1 , m );
  underlying=computeSwapRate(ptLib, o ,  s, m );

  Sigma=blackVol*sqrt(ptSwpt->swaptionMaturity);
  d1= (log(underlying/ptSwpt->strike)+ 0.5*pow(Sigma,2))/Si
    gma;
  d2=( log(underlying/ptSwpt->strike)- 0.5*pow(Sigma,2))/Si
    gma ;

  if ( (d1<borne) && (d1>-borne))
    {
      return(sum*(underlying*cdf_nor(d1) -ptSwpt->strike*
    cdf_nor(d2) ));

    }
  else
    {
      printf(" can not compute swaption price{n");
      return(0.);
    }

}

double ps_lmm(int n, double *u, double *v)
{
  int l;
  double s=0;

  for(l=0;l<n;l++){s+=u[l]*v[l];}

  return s;
}


double maxi(double a,double b)
{
  if (a>b)
    return(a);
  else
    return(b);
}

double ppos(double x){
  return(maxi(x,0));
}

int Set_to_Zero(double *ptr,int dim){
  int l;
  for(l=0;l<dim;l++) ptr[l]=0.0;
  return(1);
}
/*******************   Evolution Routines      ************
    ******************/
```

```
static int evolutionUnderSpotMeasure(const PnlVect *ptRand,
     Libor* ptLibOld , Libor* ptLibNew, Volatility *ptVol,
    double dt , double t,double sigma_cost)
{
  // computes the evolution of libor rates under the spot
    measure
  int i,k,l;
  double val=0.0;
  double drift=0.0;
  double vol=0.0;
  double normVolatility=0.0;
  double scalarProductVol=0.0;
  double T_i,T_k;
  double v_i;


  for(i=1;i<ptLibNew->numberOfMaturities;i++)
    {
      if (GET(ptLibOld->libor,i)==0.0)
  {
    LET(ptLibNew->libor,i)=0.0;
  }
      else
  {
    // compute the drift
    drift=0.0;
    for(k=1;k<=i;k++)
      {
        scalarProductVol=0.0;
        for(l=0;l<ptVol->numberOfFactors;l++)
    {
      //scalarProductVol+= ptVol->vol[i-1][l]*ptVol->vol[
    k-1][l];
      T_i=GET(ptLibOld->maturity,i);
      T_k=GET(ptLibOld->maturity,k);
      scalarProductVol+= evalVolatility(ptVol,l,t,T_i)*ev
    alVolatility(ptVol,l,t,T_k);

    }

        drift+= scalarProductVol * GET(ptLibOld->libor,k)*
```

```
   ptLibOld->tenor ;
    drift/=(1.+ ptLibOld->tenor * GET(ptLibOld->libor,
  k)) ;
   }

  // compute de square of the volatility and the random
  choc
  normVolatility=0.0;
  vol=0.0;
  for(l=0;l<ptVol->numberOfFactors;l++)
   {
    T_i=GET(ptLibOld->maturity,i);
    v_i=evalVolatility(ptVol,l,t,T_i);
    normVolatility += pow(v_i ,2);
    vol+= v_i* GET (ptRand, l);
   }

  drift+=  (-0.5*normVolatility);
  drift*=dt;
  val=(drift + sqrt(dt)*vol);

  //update
  LET(ptLibNew->libor,i)=GET(ptLibOld->libor,i)*exp(val)
  ;
 }
  }
 return(1);
}

static int evolutionUnderForwardMeasure(const PnlVect *ptRa
  nd, Libor* ptLibOld , Libor* ptLibNew, Volatility *ptVol,
  double dt , double t,double sigma_cost){
// computes the evolution of libor rates under the forward
  measure
// corresponding to the swap Maturity: numeraire is B(t,T_
  e)
// Forward rate L(t;T_{e-1},T_e)=ptLib->Libor(e-1) is a mar
  tingale
 return(1);
}
```

```
/***************           Numeraire&Evolution routine
     *****************************/

//Computation of Numeraire on the k-th MC path
//NumeraireSpot(T_j)=RollOverBond=Prod_{i=0}^{j-1}[1+tenor*
    L(T_i;i,i+1)]
//Numeraire(T_j)=ZeroCoupBond(T_j,T_numberofmaturities)

void computeNumeraire(char*MeasureName,Libor* ptLib,Swaptio
    n* ptSwpt,double *Numeraire,int j,int k,double auxspot){
  int l,s;
  double aux=1.0;
  s=(int)(ptSwpt->swaptionMaturity/ptLib->tenor);

  if(strcmp(MeasureName,"Spot")==0){
    if(((s-2)<=j)&&(j<=(ptLib->numberOfMaturities-3)))
    Numeraire[k*(ptLib->numberOfMaturities-s)+(j+2-s)]=auxspot;
      }
  else{
    if ((s-1)<=j){
      for(l=j+1;l<ptLib->numberOfMaturities;l++) aux*=(1./(
    1.+ptLib->tenor*GET(ptLib->libor,l)));
      Numeraire[k*(ptLib->numberOfMaturities-s)+(j+1-s)]=au
    x;
    }
  }
  return;
}

void
Name_To_Measure(char *ErrorMessage, char *name,
                int (**computeEvol)(const PnlVect* ptRand,
    Libor* ptLibOld, Libor* ptLibNew,
                                    Volatility* ptVol,
    double dt,double t, double sigma_cost))
{
  /*initialization of evolution */
  if (strcmp("Spot",name)==0){
    *computeEvol=evolutionUnderSpotMeasure;
  } else if (strcmp("Fwd",name)==0){
    *computeEvol=evolutionUnderForwardMeasure;
  }  else {
    strcat(ErrorMessage,"Measure Error:");
    strcat(ErrorMessage,"(");
    strcat(ErrorMessage,name);
    strcat(ErrorMessage,") is not good. Please, try with a
    valid Measure Name");
  }
  return;
}

#endif //PremiaCurrentVersion
```

References