Help

```
extern "C" {
#include  "inflation_lmm_heston1d_stdf.h"
#include "enums.h"
}
#include <vector>
#include <stdio.h>
#include <iostream>
#include <cstdlib>
#include <math.h>
#include <complex>
#include "math/fft.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else

using namespace std;
using std::string;

extern "C" {
  extern char premia_data_dir[MAX_PATH_LEN];

  extern char *path_sep;
}
static const complex<double> I(0, 1);
static complex<double> AY, BY, AX, BX, CC;
static double *tm1, *PN, *PR, *ZCSR, *ZCSRT, *tm_zcsr, *PNT, *F;
static int Nvalue, Nvalue1;                    /*Number of value read for PN*/
static char init[] = "nominal_zcb_prices.dat";
static char init1[] = "zcii_swap_rates.dat";
static FILE *Entrees;                  /*File variable of the code*/
static FILE *Entrees1;

/*Read Nominal Zero Coupon Bond*/
static int lecture_PN()
{

  int i;
  char ligne[20];
  char *pligne;
```

```
  double p, tt;
  char data[MAX_PATH_LEN];

  sprintf(data, "%s%s%s", premia_data_dir, path_sep, init);
  Entrees = fopen(data, "r");

  if (Entrees == NULL)
    {
      printf("Le FICHIER %s N'A PU ETRE OUVERT. VERIFIER LE CHEMIN\ n", data);
    }

  i = 0;
  pligne = ligne;

  PN = new double[100];
  PNT = new double[100];
  tm1 = new double[100];
  PR = new double[100];
  F = new double[100];

  while (1)
    {
      pligne = fgets(ligne, sizeof(ligne), Entrees);
      if (pligne == NULL) break;
      else
        {
          sscanf(ligne, "%lf t=%lf", &p, &tt);

          PN[i] = p;
          tm1[i] = tt;
          i++;
        }
    }
  Nvalue = i;
  fclose(Entrees);

  return i;
}

/*Read Zero Coupon Swap Rates*/
static int lecture_ZCSR()
```

```
{

  int i;
  char ligne[20];
  char *pligne;
  double p, tt;
  char data[MAX_PATH_LEN];

  sprintf(data, "%s%s%s", premia_data_dir, path_sep, init1);
  Entrees1 = fopen(data, "r");

  ZCSR = new double[100];
  ZCSRT = new double[100];
  tm_zcsr = new double[100];
  if (Entrees1 == NULL)
    {
      printf("Le FICHIER %s N'A PU ETRE OUVERT. VERIFIER LE CHEMIN\ n", data);
    }

  i = 0;
  pligne = ligne;

  while (1)
    {
      pligne = fgets(ligne, sizeof(ligne), Entrees1);
      if (pligne == NULL) break;
      else
        {
          sscanf(ligne, "%lf t=%lf", &p, &tt);
          ZCSR[i] = p;
          tm_zcsr[i] = tt;

          i++;
        }
    }
  Nvalue1 = i;
  fclose(Entrees1);

  return i;
}
```

```
static double bond_zcn(double T)
{
  double  POT;
  int i = 0;


  if (T > 0)
    {
      while (tm1[i] < T && i < Nvalue)
        {
          i = i + 1;
        }

      if (i == 0)
        {
          POT = 1 * (1 - T / tm1[0]) + PN[0] * (T / tm1[0]);
        }
      else
        {
          if (i < Nvalue)
            {

              POT = PN[i - 1] * (tm1[i] - T) / (tm1[i] - tm1[i - 1]) + PN[i] * (
              /*printf("values %d %f %f %f %f\ n",i,PN[i-1],PN[i],tm1[i-1],tm1[i
            }
          else
            {
              POT = PN[i - 1] + (T - tm1[i - 1]) * (PN[i - 1] - PN[i - 2]) / (tm
            }
        }
    }
  else
    {
      POT = 1;
    }

  return POT;
}
static double bond_zcsr(double T)
{
  double  POT;
```

```
  int i = 0;

  if (T > 0)
    {
      while (tm_zcsr[i] < T && i < Nvalue1)
        {
          i = i + 1;
        }

      if (i == 0)
        {
          POT = 1 * (1 - T / tm_zcsr[0]) + ZCSR[0] * (T / tm_zcsr[0]);
        }
      else
        {
          if (i < Nvalue)
            {

              POT = ZCSR[i - 1] * (tm_zcsr[i] - T) / (tm_zcsr[i] - tm_zcsr[i - 1
            }
          else
            {
              POT = ZCSR[i - 1] + (T - tm_zcsr[i - 1]) * (ZCSR[i - 1] - ZCSR[i -
            }
        }
    }
  else
    {
      POT = 0;
    }
  return POT;
}

/*Compute ZeroCoupon Bond Price in Creal Economy*/
static void CalculatePR(int tenor_order, double tenor, double swap_mat)
{
  int i, j;

  i = lecture_PN();
  j = lecture_ZCSR();
  i = MIN(i, j);
```

```
  if (swap_mat > tm1[i - 1])
    {
      printf("\ nError : time bigger than the last time value entered in market_
      exit(EXIT_FAILURE);
    }
  else
    {
      PNT[0] = 1.;
      for (int j = 1; j < tenor_order + 1; j++)
        {
          PNT[j] = bond_zcn((double)j * tenor);
          ZCSRT[j] = bond_zcsr((double)j * tenor);
          /*printf("%f %f\ n",PNT[j],ZCSRT[j]);*/
        }

      PR[0] = 1.0;
      for (int j = 1; j < tenor_order + 1; j++)
        {
          PR[j] = PNT[j] * pow((1.0 + ZCSRT[j]), (double)j * tenor);
          F[j] = (PNT[j - 1] / PNT[j] - 1.) / (tm1[j] - tm1[j - 1]);
          /*printf("%f\ n",PR[j]);*/
        }
    }
}

/*calculate parameter in funcion PHI as in Page 8 in Paper of Mercurio and Moren
void CalculateParameters(double epsilon, double alpha, double theta, double V0,
{
  complex<double> a1, b1, c1, gamma1, a2, b2, c2, gamma2;
  int J = caplet_number;
  double sum = 0.0;
  for (int i = 1; i < J + 1; i++)
    {
      sum = sum + sqrt(V0) * F[i] / (1 + (tm1[i] - tm1[i - 1]) * F[i]) * (tm1[i]
    }
  if (flag_freezing_type == 0)
    {
      a1 = epsilon * epsilon / 2.0;
      b1 = I * u * sigmaI * epsilon * rhoIV - alpha;
      c1 = -I * u * sigmaI * sigmaI / 2.0 - sigmaI * sigmaI * u * u / 2.0;
      gamma1 = sqrt(b1 * b1 - 4.0 * a1 * c1);
```

```
      theta = theta - epsilon / alpha * sum;
      AY = alpha * theta * (gamma1 - b1) / (2.0 * a1) * (tm1[J] - tm1[J - 1]) -
      BY = (gamma1 - b1) / (2.0 * a1) * ((1.0 - exp(gamma1 * (tm1[J] - tm1[J - 1
      a2 = a1;
      b2 = I * u * epsilon * (sigmaI * rhoIV - sigmaI * rhoIV) - alpha;
      c2 = I * u * (sigmaI * sigmaI - sigmaI * sigmaI) / 2.0 - (sigmaI * sigmaI
      gamma2 = sqrt(b2 * b2 - 4.0 * a2 * c2);
      AX = alpha * theta * (gamma2 - b2) / (2.0 * a2) * tm1[J - 1] - alpha * the
      BX = BY + (gamma2 - b2 - 2.0 * a2 * BY) / (2.0 * a2) * ((1.0 - exp(gamma2
    }
  else if (flag_freezing_type == 1)
    {
      a1 = epsilon * epsilon / 2.0;
      b1 = I * u * epsilon * sigmaI * rhoIV - (alpha + epsilon / V0 * sum);
      c1 = -I * u * sigmaI * sigmaI / 2.0 - sigmaI * sigmaI * u * u / 2.0;
      gamma1 = sqrt(b1 * b1 - 4.0 * a1 * c1);
      theta = alpha * theta / (alpha + epsilon / V0 * sum);
      AY = (alpha + epsilon / V0 * sum) * theta * (gamma1 - b1) / (2.0 * a1) * (
      BY = (gamma1 - b1) / (2.0 * a1) * ((1.0 - exp(gamma1 * (tm1[J] - tm1[J - 1
      a2 = a1;
      b2 = I * u * epsilon * (sigmaI * rhoIV - sigmaI * rhoIV) - (alpha + epsilo
      c2 = I * u * (sigmaI * sigmaI - sigmaI * sigmaI) / 2.0 - (sigmaI * sigmaI
      gamma2 = sqrt(b2 * b2 - 4.0 * a2 * c2);
      AX = (alpha + epsilon / V0 * sum) * theta * (gamma2 - b2) / (2.0 * a2) * t
      BX = BY + (gamma2 - b2 - 2.0 * a2 * BY) / (2.0 * a2) * ((1.0 - exp(gamma2
    }
}

complex<double> Phi(double epsilon, double alpha, double theta, double V0, doubl
{
  int J = caplet_number;
  CalculateParameters(epsilon, alpha, theta, V0, I0, sigmaI, F0 , sigmaF, rhoFI,
  /* for calculate X[i](t0)=log(I[i]/I[i-1])=log(Pr(t0,tm1[j])/Pn(t0,tm1[j])*Pr(
  double CrealNominalT1 = 1.0; //   PR[j]/PN[j]
  double CrealNominalT2 = 1.0; //   PR[j-1])/P[j-1]
  for (int i = 1; i < J; i++)
    {
      CrealNominalT1 = CrealNominalT1 * (1.0 + ZCSRT[J]);
      CrealNominalT2 = CrealNominalT2 * (1.0 + ZCSRT[J - 1]);
    }
  return complex<double>(exp(real(AY + AX * BX * V0 + I * u * log(CrealNominalT1
```

```
}

complex<double> Integrated_Function(double epsilon, double alpha, double theta,
{
  return complex<double>(Phi(epsilon, alpha, theta, V0, I0, sigmaI, F0 , sigmaF,
}

complex<double> Fft(double epsilon, double alpha, double theta, double  V0, doub
                    rhoFI, double rhoFV, double rhoIV, double rhoI, int caplet_n
{
  int Nlimit = 1024;
  double h = 0.04;                          //integral discretization step
  //double logstrikestep = 2*M_PI/Nlimit/h;
  double A = (Nlimit - 1) * h;
  complex<double> dzeta;
  double vn = -A / 2;
  double weight = 1. / 3; //1.0;// Simpson's rule weights
  double *z = new double [Nlimit];
  double *z_img = new double [Nlimit];

  dzeta = exp(-I * (vn - A / 2.0) * log(strike + 1.0)) * Integrated_Function(eps
  z[0] = weight * real(dzeta);
  z_img[0] = weight * imag(dzeta);
  for (int n = 1; n < Nlimit - 1; n++)
    {
      vn += h;
      weight = (weight < 1) ? 4. / 3 : 2. / 3; //1.0;   //Simpson's rule weights
      dzeta = exp(-I * (vn - A / 2.0) * log(strike + 1.0)) * Integrated_Function
      z[n] = weight * real(dzeta);
      z_img[n] = weight * imag(dzeta);
    }
  vn += h;
  weight = 2. / 3; //1.0;  //Simpson's rule weights
  dzeta = exp(-I * (vn - A / 2.0) * log(strike + 1.0)) * Integrated_Function(eps
  z[Nlimit - 1] = weight * real(dzeta);
  z_img[Nlimit - 1] = weight * imag(dzeta);
  fft1d(z, z_img, Nlimit, -1);
  complex<double> CC = A / (Nlimit - 1.0) * exp(I * log(strike + 1.0) * A / 2.0)
  delete [] z;
  delete [] z_img;
  return CC;
```

```
}

static int ap_mercuriomoreni_inflationlmmheston(double epsilon, double alpha, do
{
  int caplet_number = (int)((cap_maturity - t0) / tenor);
  double eta = 1.5;

  /*Compute ZeroCoupon Bond Price in Creal Economy*/
  CalculatePR(caplet_number, tenor, cap_maturity);

  /*Compute Price*/
  *price = PNT[caplet_number] * exp((-1.) * eta * log(strike + 1.0)) / 2.0 / M_P

  delete [] tm1;
  delete [] PN;
  delete [] PNT;
  delete [] PR;
  delete [] ZCSR;
  delete [] ZCSRT;
  delete [] tm_zcsr;
  delete [] F;
  return OK;
}
#endif //PremiaCurrentVersion


extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
  static int CHK_OPT(AP_MM)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_MM)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
  int CALC(AP_MM)(void *Opt, void *Mod, PricingMethod *Met)
  {
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
```

```
    return ap_mercuriomoreni_inflationlmmheston(ptMod->Sigma2.Val.V_PDOUBLE,
           ptMod->SpeedMeanReversion.Val.V_PDOUBLE,
           ptMod->LongRunVariance.Val.V_PDOUBLE,
           ptMod->Sigma0.Val.V_PDOUBLE,
           ptMod->I0.Val.V_PDOUBLE,
           ptMod->SigmaI.Val.V_PDOUBLE,
           ptMod->F0.Val.V_PDOUBLE,
           ptMod->SigmaF.Val.V_PDOUBLE,
           ptMod->RhoFI.Val.V_PDOUBLE,
           ptMod->RhoFV.Val.V_PDOUBLE,
           ptMod->RhoIV.Val.V_PDOUBLE,
           ptMod->RhoI.Val.V_PDOUBLE,
           ptOpt->FirstResetDate.Val.V_DATE - ptMod->T.Val.V_DATE,
           ptOpt->BMaturity.Val.V_DATE,
           ptOpt->FixedRate.Val.V_PDOUBLE,
           ptOpt->ResetPeriod.Val.V_DATE,
           Met->Par[0].Val.V_ENUM.value,
           &(Met->Res[0].Val.V_DOUBLE));

  }

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

    if ((strcmp(((Option *)Opt)->Name, "InflationIndexedCap") == 0))
      return OK;
    else
      return WRONG;
  }
#endif //PremiaCurrentVersion

  static PremiaEnumMember freezing_tec_members[] =
  {
    {"Technique 1", 0},
    {"Technique 2", 1},
    {NULL, NULLINT}
  };

  static DEFINE_ENUM(freezing_tec, freezing_tec_members);
```

```
  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 = &freezing_tec;
      }
    return OK;
  }

  PricingMethod MET(AP_MM) =
  {
    "AP_MercurioMoreni_InflationLMMHeston",
    { {"Freezing Technique", ENUM, {1}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(AP_MM),
    {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID}*/ , {" ",
    CHK_OPT(AP_MM),
    CHK_ok,
    MET(Init)
  } ;
}
```