Help

```
#include  "doublehes1d_std.h"
#include  "math/alfonsi.h"
#include "pnl/pnl_random.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(MC_DoubleHeston)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_DoubleHeston)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double maxplus(double x)
{
  return MAX(0., x);
}

static double qem_distribution(int generator, double psi, double mean, double A,
{
  double p, deuxsurpsi, unsurbeta, petita, petitc, petitg;
  if (psi > 1.5)
    {
      p = (psi - 1.) / (psi + 1.);
      unsurbeta = mean / (1. - p);
      if (((1. - A * unsurbeta * p) * (1. - A * unsurbeta) > 0.) && ((1. - A * u
        {
          *correction = -log((1. - A * unsurbeta * p) / (1 - A * unsurbeta));
          *martingalecorrection = 1;
        }
      if (pnl_rand_uni(generator) < p)
        {
          return 0.;
        }
      return -unsurbeta * log(pnl_rand_uni(generator));
    }
  deuxsurpsi = 2. / psi;
```

```
  petita = mean / (deuxsurpsi + sqrt(maxplus(deuxsurpsi * deuxsurpsi - deuxsurps
  petitc = sqrt(maxplus(deuxsurpsi - 1. + sqrt(maxplus(deuxsurpsi * deuxsurpsi -
  petitg = pnl_rand_normal(generator);
  if ((1. - 2.*A * petita) > 0.)
    {
      *correction = -A * petitc * petitc * petita / (1. - 2.*A * petita) + 0.5 *
      *martingalecorrection = 1;
    }
  return petita * (petitg + petitc) * (petitg + petitc);
}


int MCDoubleHeston(double s0, NumFunc_1  *p, double T, double r, double divid, d
{
  double K;
  // uninitialized variables
  double lgs, sv1tm, sv2tm, V1tm, V2tm, V1tmplusun, V2tmplusun, K01, K02, K11, K
  // Zhu precomputed variables
  double zhuprecomp12, zhuprecomp13, zhuprecomp14;
  double zhuprecomp22, zhuprecomp23, zhuprecomp24;
  double lambdazhu1, lambdazhu2, actzhu1, actzhu2, varzhu1, varzhu2, stdzhu1, st
  // Alfonsi's variables
  PnlMat *SpotPaths0 ;
  PnlMat *VarPaths1;
  PnlMat *AveragePaths0;
  PnlMat *VarPaths2;
  PnlMat *VarInt;
  // QEM variables
  double psi1, psi2;
  int martingalecorrection1 = 0, martingalecorrection2 = 0;
  // QEM precomputed variables
  double C11, C21, C31, C41, C12, C22, C32, C42;
  // algorithm variables
  int pth, tm;
  double z;
  //initialized variables
  double deltat = T / m;
  double sdeltat = sqrt(deltat);

  double tildeK;
  double error = 0.;
```

```
double test = 0.;
double vartest = 0.;
int flag_call;

SpotPaths0   = pnl_mat_create(m + 1, n);
VarPaths1    = pnl_mat_create(m + 1, n);
AveragePaths0 = pnl_mat_create(m + 1, n);
VarPaths2    = pnl_mat_create(m + 1, n);
VarInt      = pnl_mat_create(m + 1, n);

K = p->Par[0].Val.V_PDOUBLE;
tildeK = K * exp(-r * T);
if ((p->Compute) == &Call)
  flag_call = 1;
else
  flag_call = 0;

//initialization of the variables
pnl_rand_init(generator, 1, (long) n * m);
*ptprice = 0.;

K01 = -rho1 * b1 * theta1 * deltat / sigma1;
K02 = -rho2 * b2 * theta2 * deltat / sigma2;
K11 = deltat * 0.5 * (b1 * rho1 / sigma1 - 0.5) - rho1 / sigma1;
K12 = deltat * 0.5 * (b2 * rho2 / sigma2 - 0.5) - rho2 / sigma2;
K21 = K11 + 2.*rho1 / sigma1;
K22 = K12 + 2.*rho2 / sigma2;
K31 = deltat * 0.5 * (1. - rho1 * rho1);
K32 = deltat * 0.5 * (1. - rho2 * rho2);
A1 = (K21 + 0.5 * K31);
A2 = (K22 + 0.5 * K32);
tildeA1 = (K11 + 0.5 * K31);
tildeA2 = (K12 + 0.5 * K32);
// precomputed terms
switch (flag_scheme)
  {
  case 1: // Zhu scheme
    zhuprecomp12 = exp(-0.5 * b1 * deltat);
    actzhu1 = exp(-b1 * deltat);
    varzhu1 = 0.25 * sigma1 * sigma1 * (1. - exp(-b1 * deltat)) / b1;
    stdzhu1 = sqrt(varzhu1);
```

```
    if ((1. - 2.*varzhu1 * A1) <= 0.)
      {
        //printf("Zhu--martingale correction for V1 : disabled \ n");
        zhuprecomp13 = 0.;
        zhuprecomp14 = K01;
        tildeA1 = 0.;
      }
    else
      {
        //printf("Zhu--martingale correction for V1 : enabled \ n");
        zhuprecomp13 = A1 / (1. - 2.*varzhu1 * A1);
        zhuprecomp14 = 0.5 * log((1. - 2.*varzhu1 * A1));
      }
    zhuprecomp22 = exp(-0.5 * b2 * deltat);
    actzhu2 = exp(-b2 * deltat);
    varzhu2 = 0.25 * sigma2 * sigma2 * (1. - exp(-b2 * deltat)) / b2;
    stdzhu2 = sqrt(varzhu2);
    if (1. - 2.*varzhu2 * A2 <= 0.)
      {
        //printf("Zhu--martingale correction for V2 : disabled \ n");
        zhuprecomp23 = 0.;
        zhuprecomp24 = K02;
        tildeA2 = 0.;
      }
    else
      {
        //printf("Zhu--martingale correction for V2 : enabled \ n");
        zhuprecomp23 = A2 / (1. - 2.*varzhu2 * A2);
        zhuprecomp24 = 0.5 * log((1. - 2.*varzhu2 * A2));
      }
    break;
  case 2: // Alfonsi 2nd order scheme
    //printf("Alfonsi 2nd order scheme--without martingale correction\ n");
    HestonSimulation_Alfonsi_Modified(0, SpotPaths0, 1, VarPaths1, 0, AverageP
    HestonSimulation_Alfonsi_Modified(0, SpotPaths0, 1, VarPaths2, 0, AverageP
    break;
  case 3: // Alfonsi 3rd order scheme
    //printf("Alfonsi 3nd order scheme--without martingale correction\ n");
    HestonSimulation_Alfonsi_Modified(0, SpotPaths0, 1, VarPaths1, 0, AverageP
    HestonSimulation_Alfonsi_Modified(0, SpotPaths0, 1, VarPaths2, 0, AverageP
    break;
```

```
  case 4: // Quadratic exponential martingale
    //printf("QEM scheme--without martingale correction\ n");
    C11 = sigma1 * sigma1 * exp(-b1 * deltat) * (1. - exp(-b1 * deltat)) / b1;
    C21 = theta1 * sigma1 * sigma1 * 0.5 * (1. - exp(-b1 * deltat)) / b1;
    C31 = exp(-b1 * deltat);
    C41 = theta1 * (1. - exp(-b1 * deltat));
    C12 = sigma2 * sigma2 * exp(-b2 * deltat) * (1. - exp(-b2 * deltat)) / b2;
    C22 = theta2 * sigma2 * sigma2 * 0.5 * (1. - exp(-b2 * deltat)) / b2;
    C32 = exp(-b2 * deltat);
    C42 = theta2 * (1. - exp(-b2 * deltat));
    break;
  case 5 : // Euler  scheme
    break;
  }
// loops
for (pth = 0; pth < n; pth++) //Monte Carlo iterations
  {
    lgs = log(s0);
    V1tm = V01;
    V2tm = V02;
    sv1tm = sqrt(V1tm);
    sv2tm = sqrt(V2tm);
    V1tmplusun = V01;
    V2tmplusun = V02;
    for (tm = 0; tm < m ; tm++) //Time step discretization
      {
        V1tm = V1tmplusun;
        V2tm = V2tmplusun;
        switch (flag_scheme)
          {
          case 1: // Zhu scheme
            /* sv(t+dt)=lambdazhu+exp(-0.5*b*deltat)(sv(t)-lambdazhu)+stdzhu*G
             *  varzhu^2=(0.5*sigma)^2*(1-exp(-b*deltat))/b
             *  lambdazhu is chosen to preserve the second-order moment
             */
            lambdazhu1 = (sqrt(maxplus(theta1 + (V1tm - theta1) * actzhu1 - va
            lambdazhu2 = (sqrt(maxplus(theta2 + (V2tm - theta2) * actzhu2 - va
            mean1 = lambdazhu1 + (sv1tm - lambdazhu1) * zhuprecomp12;
            mean2 = lambdazhu2 + (sv2tm - lambdazhu2) * zhuprecomp22;
            // martingale correction
            K01 = -zhuprecomp13 * mean1 * mean1 + zhuprecomp14 - tildeA1 * V1t
```

```
      K02 = -zhuprecomp23 * mean2 * mean2 + zhuprecomp24 - tildeA2 * V2t
      // increment
      z = pnl_rand_normal(generator);
      sv1tm = mean1 + stdzhu1 * z;
      V1tmplusun = sv1tm * sv1tm;
      z = pnl_rand_normal(generator);
      sv2tm = mean2 + stdzhu2 * z;
      V2tmplusun = sv2tm * sv2tm;
      break;
    case 2: // Alfonsi 2nd order scheme without martingale correction
      V1tmplusun = pnl_mat_get(VarPaths1, tm + 1, pth);
      V2tmplusun = pnl_mat_get(VarPaths2, tm + 1, pth);
      break;
    case 3: // Alfonsi 3rd order scheme without martingale correction
      V1tmplusun = pnl_mat_get(VarPaths1, tm + 1, pth);
      V2tmplusun = pnl_mat_get(VarPaths2, tm + 1, pth);
      break;
    case 4: // Quadratic exponential martingale with martingale correcti
      mean1 = theta1 + (V1tm - theta1) * exp(-b1 * deltat);
      mean2 = theta2 + (V2tm - theta2) * exp(-b2 * deltat);
      psi1 = (C11 * V1tm + C21) / ((C31 * V1tm + C41) * (C31 * V1tm + C4
      psi2 = (C12 * V1tm + C22) / ((C32 * V2tm + C42) * (C32 * V2tm + C4
      // increment
      V1tmplusun = qem_distribution(generator, psi1, mean1, A1, &K01, &m
      V2tmplusun = qem_distribution(generator, psi2, mean2, A2, &K02, &m
      // martingale correction
      K01 = K01 - martingalecorrection1 * tildeA1 * V1tm;
      K02 = K02 - martingalecorrection2 * tildeA2 * V2tm;
      break;
    case 5: // Euler full truncation scheme
      z = pnl_rand_normal(generator);
      V1tmplusun = V1tm + b1 * (theta1 - maxplus(V1tm)) * deltat + sigma
      z = pnl_rand_normal(generator);
      V2tmplusun = V2tm + b2 * (theta2 - maxplus(V2tm)) * deltat + sigma
      break;
    }
  // increment of the logpriceprocess(lgs)
  z = pnl_rand_normal(generator);
  lgs = lgs + K01 + K02 + K11 * V1tm + K21 * V1tmplusun + K12 * V2tm + K
  z = pnl_rand_normal(generator);
  lgs = lgs + sqrt(maxplus(K32 * (V2tm + V2tmplusun))) * z;
```

```
        }
      test += exp(lgs) / ((double) n);
      vartest += exp(2.*lgs) / ((double) n);

      //Call
      if (flag_call)
        {
          *ptprice = (*ptprice) + maxplus(exp(lgs - divid * T) - tildeK) - exp(l
          error = error + (maxplus(exp(lgs - divid * T) - tildeK) - exp(lgs) + s
        }
      else//Put
        {
          *ptprice = (*ptprice) + maxplus(tildeK - exp(lgs - divid * T)) + exp(l
          error = error + (maxplus(-exp(lgs - divid * T) + tildeK) + exp(lgs) -
        }

    }
  /* Price*/
  *ptprice = *ptprice / ((double) n);

  *inf_price = *ptprice - 1.96 * sqrt((error / ((double) n) - (*ptprice) * (*ptp
  *sup_price = *ptprice + 1.96 * sqrt((error / ((double) n) - (*ptprice) * (*ptp
  pnl_mat_free(&SpotPaths0);
  pnl_mat_free(&VarPaths1);
  pnl_mat_free(&AveragePaths0);
  pnl_mat_free(&VarPaths2);
  pnl_mat_free(&VarInt);


  return OK;
}
int CALC(MC_DoubleHeston)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;

  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
```

```
  return MCDoubleHeston(ptMod->S0.Val.V_PDOUBLE,
                        ptOpt->PayOff.Val.V_NUMFUNC_1,
                        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                        r,
                        divid, ptMod->Sigma0.Val.V_PDOUBLE,
                        ptMod->Sigma0V.Val.V_PDOUBLE,
                        ptMod->LongRunVariance.Val.V_PDOUBLE,
                        ptMod->LongRunVarianceV.Val.V_PDOUBLE,
                        ptMod->MeanReversion.Val.V_PDOUBLE,
                        ptMod->MeanReversionV.Val.V_PDOUBLE,
                        ptMod->Sigma.Val.V_PDOUBLE,
                        ptMod->SigmaV.Val.V_PDOUBLE,
                        ptMod->Rho.Val.V_DOUBLE,
                        ptMod->RhoSV2.Val.V_DOUBLE,
                        Met->Par[0].Val.V_LONG,
                        Met->Par[1].Val.V_INT,
                        Met->Par[2].Val.V_ENUM.value,
                        Met->Par[3].Val.V_ENUM.value,
                        &(Met->Res[0].Val.V_DOUBLE),
                        &(Met->Res[1].Val.V_DOUBLE),
                        &(Met->Res[2].Val.V_DOUBLE)
                       );
}


static int CHK_OPT(MC_DoubleHeston)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt
    return OK;

  return  WRONG;
}
#endif //PremiaCurrentVersion

static PremiaEnumMember DoubleHestonOrder_members[] =
{
  { "Exact Zhu",  1},
  { "Second Order Alfonsi", 2},
  { "Third Order Alfonsi", 3},
  { "Quadratic Exponential Martingale", 4},
  { "Euler scheme", 5},
```

```
  { NULL, NULLINT }
};

static DEFINE_ENUM(DoubleHestonOrder, DoubleHestonOrder_members);

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

      Met->Par[0].Val.V_LONG = 100000;
      Met->Par[1].Val.V_INT = 24;
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
      Met->Par[3].Val.V_ENUM.value = 1;
      Met->Par[3].Val.V_ENUM.members = &DoubleHestonOrder;
    }

  return OK;
}

PricingMethod MET(MC_DoubleHeston) =
{
  "MC_DoubleHeston",
  { {"N iterations", LONG, {100}, ALLOW},
    {"TimeStepNumber", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Scheme type", ENUM, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_DoubleHeston),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Inf Price", DOUBLE, {100}, FORBID},
    {"Sup Price", DOUBLE, {100}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_DoubleHeston),
  CHK_ok,
  MET(Init)
};
```