```
   Help
#include  "hes1d_std.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h"
#include  "math/alfonsi.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2012+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_AM_Alfonsi_LongstaffSchwartz)(void *
    Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_MALLIAVIN_HESTON)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double **X, **Nu, **NuW, **NuWT, **NuI;
static double *TP, *TPE, *TPS, *TR;
static double *Dpath,*Npath;



static void memory_allocation(int Ntraj, int Nst)
{
  int i;


  TP=(double *)malloc((Ntraj)*sizeof(double));

  TPE=(double *)malloc((Ntraj)*sizeof(double));

  TPS=(double *)malloc((Ntraj)*sizeof(double));
```

```
  TR=(double *)malloc((Ntraj)*sizeof(double));

  Dpath=(double *)malloc((Ntraj)*sizeof(double));

  Npath=(double *)malloc((Ntraj)*sizeof(double));

 // Nu parmeters ////////////////////////////////////////
    ///////////////////////////////
  Nu=(double **)calloc(Nst,sizeof(double *));
  for (i=0;i<Nst;i++)
    Nu[i]=(double *)calloc(Ntraj,sizeof(double));

  NuW=(double **)calloc(Nst,sizeof(double *));
  for (i=0;i<Nst;i++)
    NuW[i]=(double *)calloc(Ntraj,sizeof(double));

  NuWT=(double **)calloc(Nst,sizeof(double *));
  for (i=0;i<Nst;i++)
    NuWT[i]=(double *)calloc(Ntraj,sizeof(double));

  NuI=(double **)calloc(Nst,sizeof(double *));
  for (i=0;i<Nst;i++)
    NuI[i]=(double *)calloc(Ntraj,sizeof(double));

// log spot/S0 /////////////////////////////////////////
    ///////////////////////////////
  X=(double **)calloc(Nst,sizeof(double *));
  for (i=0;i<Nst;i++)
    X[i]=(double *)calloc(Ntraj,sizeof(double));

}


static void free_memory(int Nst)
{
  int i;

  for (i=0;i<Nst;i++)
    free(X[i]);
  free(X);
```

```
  for (i=0;i<Nst;i++)
    free(NuI[i]);
  free(NuI);

  for (i=0;i<Nst;i++)
    free(NuW[i]);
  free(NuW);

  for (i=0;i<Nst;i++)
    free(NuWT[i]);
  free(NuWT);

  for (i=0;i<Nst;i++)
    free(Nu[i]);
  free(Nu);

  free(TP);
  free(TPE);
  free(TPS);
  free(TR);
  free(Dpath);
  free(Npath);
}

static void CompCash(double dt, int Jindex, double tau,
    int Ntraj, int Nts)
{
  double output;
  int i;

  for (i = 0; i < Ntraj; i++) {
     if(Jindex==Nts){output = TPE[i];
   }else{
     if(TP[i]*exp(-dt*dt*tau) > TPE[i]){
     if((int)Jindex < (int)Nts-1){output = TR[i]*exp(-dt*
    dt*tau);
     }else{output = TPS[i]*exp(-dt*dt*tau);}
     }else{output = TPE[i];}
   }
   TR[i]=output;
```

```
  }
}

////////////////////////////////////////////////////////////
    ////////////////////////////////
// Path choice
////////////////////////////////////////////////////////////
    ////////////////////////////////
static void CompLambda(int Tindex, int Ntraj){

  double sumD, sumN, vD, vN, cDN, lambda1, lambda2;
  int l;

  sumN = 0.0;
  sumD = 0.0;
  vD = 0.0;
  vN = 0.0;
  cDN = 0.0;

  for (l = 0; l < Ntraj; l++) {
      sumN += Npath[l];
      sumD += Dpath[l];
    vN += Npath[l]*Npath[l];
    vD += Dpath[l]*Dpath[l];
    cDN += Npath[l]*Dpath[l];
  }

  sumD = sumD/Ntraj;
  sumN = sumN/Ntraj;
  vD = vD/Ntraj - sumD*sumD;
  vN = vN/Ntraj - sumN*sumN;
  cDN = cDN/Ntraj - sumD*sumN;

  if(sumN*sumN*vD > sumD*sumD*vN){
    lambda1 = MIN(1,0.5 + (sumD*cDN/(2*sumN*vD)));
    sumD = 0.0;
    sumN = 0.0;
    for (l = 0; l < Ntraj; l++) {
       sumD += Dpath[l];
    }
      for (l = 0; l < Ntraj*lambda1; l++) {
```

```
       sumN += Npath[l];
    }

    TP[Tindex]=sumN/(lambda1*sumD);

  }else{
    lambda2 = MIN(1,0.5 + (sumN*cDN/(2*sumD*vN)));
    sumD = 0.0;
    sumN = 0.0;
    for (l = 0; l < Ntraj; l++) {
       sumN += Npath[l];
    }
      for (l = 0; l < Ntraj*lambda2; l++) {
       sumD += Dpath[l];
    }

    TP[Tindex]=(lambda2*sumN)/sumD;

  }
}

static void CondExp(int Ntraj, double rho, double r, int jd
    x, int idx, double dt){

int i;
double dx, loc, st;

st = ((double)jdx + 1.0)*dt*dt;

  for (i = 0; i < Ntraj; i++) {

    loc = exp((1.0-0.5*rho*rho)*NuI[jdx][i] - r*st - rho*
    NuWT[jdx][i] -
          sqrt(1.0-rho*rho)*(NuI[jdx][i]/NuI[jdx+1][i])*
    NuW[jdx+1][i] -
          0.5*(1.0-rho*rho)*NuI[jdx][i]*NuI[jdx][i]/NuI[
    jdx+1][i]);

    dx = (-X[jdx][idx] + r*st + rho*NuWT[jdx][i] + (rho*
    rho-1.5)*NuI[jdx][i] +
       sqrt(1.0-rho*rho)*(NuI[jdx][i]/NuI[jdx+1][i])*NuW
```

```
    [jdx+1][i] +
       (1.0-rho*rho)*NuI[jdx][i]*NuI[jdx][i]/NuI[jdx+1][
    i])*sqrt(NuI[jdx+1][i])/
       sqrt((1.0-rho*rho)*(NuI[jdx+1][i]*NuI[jdx][i]-NuI
    [jdx][i]*NuI[jdx][i]));

    Dpath[i] = loc*exp(-0.5*dx*dx)*sqrt(NuI[jdx+1][i])/
             (sqrt((2.0*M_PI)*(1-rho*rho)*(NuI[jdx+1][i]
    *NuI[jdx][i]-NuI[jdx][i]*NuI[jdx][i])));
    Npath[i] = TR[i]*Dpath[i];
  }
}


static int MC_Malliavin_Heston(NumFunc_1 *p, double Spot,
    double Maturity, double r, double Dividend_Rate, double V0,
    double k, double theta, double sigma, double rho, long Ntraj,
    int Nb_Exercice_Dates,int NbrStepPerPeriod,int generator,
    int flag_cir,double *price,double *error)
{
    PnlMat *SpotPaths, *AveragePaths, *VarPaths, *
    VarianceInt;
  // Indices used for trajectories and dimensions
  int ii, jj;
  // Indices needed to compute the sum end the sum square
  double sum, sum2;
  // The square root of the time increment
  double dt = sqrt((double)Maturity/Nb_Exercice_Dates);

  pnl_rand_init(generator, 1, 1);
  // Memory allocation for tables that containes the asset
    , denom, num, ..
  memory_allocation(Ntraj, Nb_Exercice_Dates);
  sum = 0.0;
  sum2 = 0.0;

  SpotPaths = pnl_mat_create(Nb_Exercice_Dates+1, Ntraj);
  AveragePaths = pnl_mat_create(1, 1);
  VarPaths = pnl_mat_create(Nb_Exercice_Dates+1, Ntraj);
  VarianceInt = pnl_mat_create(Nb_Exercice_Dates+1, Ntraj)
    ;
```

```
  HestonSimulation_Alfonsi_Modified(1, SpotPaths, 1, VarP
    aths, 0, AveragePaths, VarianceInt, Spot,
                    Maturity, r, Dividend_Rate,
    V0, k, theta, sigma, rho,
                    Ntraj, (Nb_Exercice_Dates+1),
     NbrStepPerPeriod, generator, flag_cir);

  for (jj = 0; jj < Nb_Exercice_Dates; jj++){
     for (ii = 0; ii < Ntraj; ii++) {
      X[jj][ii] = log(pnl_mat_get(SpotPaths, jj+1, ii)/
    Spot);
      Nu[jj][ii] = pnl_mat_get(VarPaths, jj+1, ii);
      NuI[jj][ii] = pnl_mat_get(VarianceInt, jj+1, ii);
      NuWT[jj][ii] = (Nu[jj][ii]-k*(theta*dt*dt*((double)
    jj+1.0)-NuI[jj][ii]))/sigma;
      NuW[jj][ii] = (X[jj][ii]-r*dt*dt*((double)jj+1.0)+0
    .5*NuI[jj][ii]-rho*NuWT[jj][ii])/sqrt(1.0-rho*rho);
     }
  }

  for (ii = 0; ii < Ntraj; ii++) {
      TPE[ii] = p->Compute(p->Par,Spot*exp(X[Nb_Exercice_Da
    tes-1][ii]));
      TPS[ii] =  TPE[ii];
  }

  // comparison
     CompCash(dt, Nb_Exercice_Dates-1, r, Ntraj, Nb_Exerc
    ice_Dates-1);

// Backward induction
  for (jj = Nb_Exercice_Dates-2; jj >= 0; jj--){
          // - time step loop

     // payoff
      for (ii = 0; ii < Ntraj; ii++)
        TPE[ii]=p->Compute(p->Par,Spot*exp(X[jj][ii]));

       // continuation
       for (ii = 0; ii < Ntraj; ii++) {
```

```
      CondExp(Ntraj, rho, r, jj, ii, dt);
      CompLambda(ii, Ntraj);
     }


     // comparison
     CompCash(dt, jj, r, Ntraj, Nb_Exercice_Dates-1);



     for (ii = 0; ii < Ntraj; ii++) {
      TPS[ii] = TPE[ii];
     }

  }

  for (ii = 0; ii < Ntraj; ii++) {
     sum += TR[ii];
     sum2 += TR[ii]*TR[ii];
  }

  // Compute the price final error
  *price = MAX(exp(-r*dt*dt)*(sum/Ntraj),p->Compute(p->
    Par,Spot));
   *error = 1.96*sqrt((exp(-2*r*dt*dt)/(Ntraj-1))*(sum2 -
    (sum*sum)/Ntraj))/sqrt(Ntraj);

  // rajouter une fonction payoff à la fin pour savoir si
    on exerce à l'instant initial

  // Free the memory
  free_memory(Nb_Exercice_Dates);
  pnl_mat_free(&SpotPaths);
  pnl_mat_free(&AveragePaths);
  pnl_mat_free(&VarPaths);
  pnl_mat_free(&VarianceInt);

  return OK;
}

int CALC(MC_MALLIAVIN_HESTON)(void *Opt, void *Mod, Pricing
```

```
    Method *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.);
  Met->Par[1].Val.V_INT = MAX(2, Met->Par[1].Val.V_INT); //
     At least two exercise dates.

  return MC_Malliavin_Heston(ptOpt->PayOff.Val.V_NUMFUNC_1,
                             ptMod->S0.Val.V_PDOUBLE,
                             ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                             r,
                             divid,
                             ptMod->Sigma0.Val.V_PDOUBLE,
                             ptMod->MeanReversion.hal.V_PDO
    UBLE,
                             ptMod->LongRunVariance.Val.V_
    PDOUBLE,
                             ptMod->Sigma.Val.V_PDOUBLE,
                             ptMod->Rho.Val.V_PDOUBLE,
                             Met->Par[0].Val.V_LONG,
                             Met->Par[1].Val.V_INT, Met->
    Par[2].Val.V_INT,
                             Met->Par[3].Val.V_ENUM.value,
    Met->Par[4].Val.V_ENUM.value,
                              &(Met->Res[0].Val.V_DOUBLE),&(
    Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(MC_MALLIAVIN_HESTON)(void *Opt, void *
    Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->EuOrAm).Val.V_BOOL==AMER)
```

```
    return OK;
  else
    return WRONG;
}
#endif //PremiaCurrentVersion

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

      Met->Par[0].Val.V_LONG=1000;
      Met->Par[1].Val.V_INT=30;
      Met->Par[2].Val.V_INT=1;
      Met->Par[3].Val.V_ENUM.value=0;
      Met->Par[3].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[4].Val.V_ENUM.value=2;
      Met->Par[4].Val.V_ENUM.members=&PremiaEnumCirOrder;
    }

  return OK;
}


PricingMethod MET(MC_MALLIAVIN_HESTON)=
{
  "MC_Malliavin_Heston",
  {
    {"N Simulations",LONG,{100},ALLOW},
    {"N Exercise Dates",INT,{100},ALLOW},
    {"N Steps per Period",INT,{100},ALLOW},
    {"RandomGenerator",ENUM,{100},ALLOW},
    {"Cir Order",ENUM,{100},ALLOW},
    {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_MALLIAVIN_HESTON),
  {   {"Price",DOUBLE,{100},FORBID},
      {"Error",DOUBLE,{100},FORBID},
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_MALLIAVIN_HESTON),
  CHK_ok,
```

```
  MET(Init)
};
```

References