Help

```
/************************************************************
    *************/
/* Author Lokman A. Abbas-Turki   <lokman.abbas-turki@lapos
    te.net>      */
/*
                */
/* Pricing American options using Malliavin calculus and
    non-parametric */
/* variance reduction methods based on conditionning and a
    judicious    */
/* choice of the number of paths used for the approximatio
    n of the    */
/* numerator and the denomenator that intervene in the cond
    itional    */
/* expectation (the continuation).
       */
/*                                    */

/* This method does not use any other variance reduction
    method except  */
/* the ones described in:
     */
/*                                    */

/* American Options Based on Malliavin Calculus and Nonpara
    metric    */
/* Variance Reduction Methods, Lokman Abbas-Turki and Bern
    ard Lapeyre,  */
/* preprint on arXiv.org.
     */
/*                                    */
/* In this version of the program, the author provides only
     the pricing */
/* method for the multidimensional non-correlated Black &
    Scholes model.*/
/* As mentioned in the paper above, we do not use controle
    variate for  */
/* the variance reduction.
     */
```

```
/***************************************************************
    **************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "bsnd_stdnd.h"
#include "black.h"
#include "optype.h"
#include "enums.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_vector.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_MalliavinAmer)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_MalliavinAmer)(void *Opt, void *Mod, Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


static double **X,**W;
static double **Denom1D, *Denom, **Denom1DVar, *DenomVar;
static double *TP, *TPE, *TPS, *TR;
static double **WLoc;
static double *Dpath,*Npath;


////////////////////////////////////////////////////////////
    ////////////////////////////////
// Memory allocation of the computation parameters
```

```
////////////////////////////////////////////////////////
    ////////////////////////////////
void memory_allocation(int Ntraj, int Dim)
{
  int i;

  ////////////////////////////////////////////////////////
    ////////////////////////////////
  // The computation parameters
  ////////////////////////////////////////////////////////
    ////////////////////////////////

  Denom1D=(double **)calloc(Dim,sizeof(double *));
  for (i=0;i<Dim;i++)
    Denom1D[i]=(double *)calloc(Ntraj,sizeof(double));

  Denom1DVar=(double **)calloc(Dim,sizeof(double *));
  for (i=0;i<Dim;i++)
    Denom1DVar[i]=(double *)calloc(Ntraj,sizeof(double));


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

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

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

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

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

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

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

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

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

```
  ////////////////////////////////////////////////////////
    ////////////////////////////////////

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

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

}


void free_memory(int Ntraj, int Dim)
{
  int i;

  for (i=0;i<Dim;i++)
    free(Denom1D[i]);
  free(Denom1D);

  for (i=0;i<Dim;i++)
    free(Denom1DVar[i]);
  free(Denom1DVar);

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

  for (i=0;i<Dim;i++)
    free(W[i]);
  free(W);

  for (i=0;i<Dim;i++)
    free(WLoc[i]);
  free(WLoc);

  free(Denom);
  free(DenomVar);
  free(TP);
```

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


////////////////////////////////////////////////////////////
    /////////////////
// Using uniformly distributed variables and the Brownian
    bridge technique
// this function generates the Brownian motion paths
////////////////////////////////////////////////////////////
    /////////////////
void BrownCMRG(double dt, int stck, int Nindex, int Jindex,
     int Ntraj, int generator)
{
  int i;                    // loop index
  double t = dt*dt;

  for (i = 0; i < Ntraj; i++) {
    if (Nindex-1>=Jindex){
      W[stck][i] = (((double)Jindex/(Jindex+1.0))*W[stck][
    i] +
                  (sqrt((double)Jindex*t/(Jindex+1.0))*pn
    l_rand_normal(generator)));
    }
    else{W[stck][i] = sqrt((double)Nindex*t)*pnl_rand_nor
    mal(generator);
    }
  }
}

////////////////////////////////////////////////////////////
    /////////////////
// Actualization of the stock price
////////////////////////////////////////////////////////////
    /////////////////
void ActStock(double dt, int Jindex, int stck, double tau,
    int Ntraj,
```

```
                double *ps, double *pd, double *pv, double **
    prho)
{
  int j;
  int i;
  double call;

  for (i = 0; i < Ntraj; i++) {
    // The initial value of the stock
    call = ps[stck];

    // First Pass
    call = call*exp((tau-pd[stck])*dt*dt*((double)Jindex) +

                    pv[stck]*prho[stck][0]*(W[stck][i]-dt*
    dt*((double)Jindex)*prho[stck][0]*0.5*pv[stck]));

    if (stck >= 1) {
      for (j = 1; j <= stck ; j++) {
        call = call*exp(prho[stck][j]*pv[stck]*
                        (W[stck][i]-dt*dt*((double)Jindex)*
    prho[stck][j]*0.5*pv[stck]));
      }
    }
    X[i][stck] = call;
  }
}

////////////////////////////////////////////////////////////
    //////////////////
// Compute denomenator for Path choice
////////////////////////////////////////////////////////////
    //////////////////
void CompD(double dt, int Jindex, int Tindex, int Ntraj,
    int Dim, double *pv)
{
  double d2, expd2, expmult;
  int i, stck;
  double jp1dj, jdjp1;
  jp1dj = ((double)Jindex+1.0)/Jindex;
  jdjp1 = (double)Jindex/(Jindex+1.0);
```

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

    d2 = sqrt(jp1dj)*(W[0][Tindex]-(jdjp1)*WLoc[0][i])/dt
      + dt*sqrt(jdjp1)*pv[0];

    expd2 = sqrt((1.0 + (double)Jindex))*exp(-0.5*d2*d2);

    expmult = exp(-jdjp1*pv[0]*(WLoc[0][i]+0.5*pv[0]*dt*dt*
    Jindex));

    Dpath[i] = expmult*expd2;

    if(Dim > 1){
      for (stck = 1; stck < Dim; stck++) {

        d2 = sqrt(jp1dj)*(W[stck][Tindex]-(jdjp1)*WLoc[stc
    k][i])/dt
          + dt*sqrt(jdjp1)*pv[stck];

        expd2 = sqrt((1.0 + (double)Jindex))*exp(-0.5*d2*d2
    );

        expmult = exp(-jdjp1*pv[stck]*(WLoc[stck][i]+0.5*pv
    [stck]*dt*dt*Jindex));

        Dpath[i] = Dpath[i]*expmult*expd2;
      }
    }
  }
}

////////////////////////////////////////////////////////////
    /////////////////
// Compute Numerator for Path choice
////////////////////////////////////////////////////////////
    /////////////////
void CompN(int Ntraj)
{
  int i;
```

```
  for (i = 0; i < Ntraj; i++) {
    Npath[i] = TR[i]*Dpath[i];
  }
}

////////////////////////////////////////////////////////////
    /////////////////
// Compute denomenator
////////////////////////////////////////////////////////////
    /////////////////
void CompDenom(double dt, int Jindex, int Ntraj, int Dim,
    double *pv)
{
  double call, d2call, expd2;
  int i, stck;
  double dt_sqrt_J;
  double dt2 = dt*dt;

  dt_sqrt_J = dt*sqrt((double)Jindex);

  for (stck = 0; stck < Dim ; stck++) {
    for (i = 0; i < Ntraj; i++) {
      call = (W[stck][i] + pv[stck]*Jindex*dt2)/dt_sqrt_J;

      Denom1D[stck][i] = exp(-0.5*call*call);

      d2call = (W[stck][i] + 2.0*pv[stck]*Jindex*dt2)/dt_sq
    rt_J;
      expd2 = exp(-0.5*d2call*d2call);
      call = 2*M_PI*(1.0 + (double)Jindex + dt2*Jindex*pv[
    stck]*pv[stck])*(1.0-cdf_nor(d2call)) +
        sqrt(2*M_PI)*(d2call - 2*dt_sqrt_J*pv[stck])*expd2;
      Denom1DVar[stck][i] = call*exp(pv[stck]*pv[stck]*Jind
    ex*dt2);

      // The value returned in the one-dimensional case
      if (Dim == 1) {
        Denom[i] = Denom1D[0][i];
        DenomVar[i] = Denom1DVar[0][i]-Denom[i]*Denom[i];
      }
    }
```

```
  }

  if (Dim > 1) {
    for (i = 0; i < Ntraj; i++) {
      call = Denom1D[0][i];
      d2call = Denom1DVar[0][i];
      for (stck = 1; stck < Dim ; stck++) {
        call = call*Denom1D[stck][i];
        d2call = d2call*Denom1DVar[stck][i];
      }
      Denom[i] = call;
      DenomVar[i] = d2call-Denom[i]*Denom[i];
    }
  }
}


////////////////////////////////////////////////////////////
    /////////////////
// Compute Cash-Flow
////////////////////////////////////////////////////////////
    /////////////////
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
////////////////////////////////////////////////////////////
    ////////////////////////////////
void CompLambda(int Tindex, int Ntraj){

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

  sumN = 0.0;
  vN = 0.0;
  cDN = 0.0;

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


  sumD = Denom[Tindex];
  sumN = sumN/Ntraj;

  vD = DenomVar[Tindex];
  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;

  }
}

int ImprovedMalliavin(PnlVect *BS_Spot,
                      NumFunc_nd *p,
                      double Maturity,
                      double r,
                      PnlVect *BS_Dividend_Rate,
                      PnlVect *BS_Volatility,
                      double rho,
                      long Ntraj,
                      int generator,
                      int Nb_Exercice_Dates,
                      double *price,double *error)
{
  int Dim = BS_Spot->size;
  // Indices used for trajectories and dimensions
  int kk, 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);
  // The correlation matrix
  PnlMat *CorrM;
  // The Cholesky decomposition of the correlation matrix
  double **prho;

  double *pd, *pv, *ps;
```

```
PnlVect VStock;

VStock.size=Dim;
// The model parameters malloc

pd=(double *)malloc((Dim)*sizeof(double));
pv=(double *)malloc((Dim)*sizeof(double));
ps=(double *)malloc((Dim)*sizeof(double));

// From pnlvect to pointer ------------------------------
  ------------------
for (kk = 0; kk < Dim; kk++) {
  pv[kk] = pnl_vect_get (BS_Volatility, kk);
  pd[kk] = pnl_vect_get (BS_Dividend_Rate, kk);
  ps[kk] = pnl_vect_get (BS_Spot, kk);
}

// Create and fill the correlation matrix
CorrM = pnl_mat_create(Dim, Dim);
for(ii = 0; ii < Dim; ii++) {
  for(jj = 0; jj < Dim; jj++) {
    if(ii==jj){pnl_mat_set (CorrM, ii , jj , 1.0);
    }else{pnl_mat_set (CorrM, ii , jj , rho);}
  }
}
pnl_mat_chol (CorrM);

// Malloc and fill the rho matrix with the Cholesky de
  composition of M
prho=(double **)calloc(Dim,sizeof(double *));
for (ii=0;ii<Dim;ii++)
  prho[ii]=(double *)calloc(Dim,sizeof(double));
for (ii = 0; ii < Dim; ii++) {
  for (jj = 0; jj <= ii; jj++) {
    prho[ii][jj] = pnl_mat_get (CorrM, ii, jj);
  }
}

// Memory allocation for tables that containes the asset,
   denom, num,
// Brownian ... values
```

```
memory_allocation(Ntraj, Dim);
sum = 0.0;
sum2 = 0.0;
for (kk = 0; kk < Dim; kk++){

  BrownCMRG(dt, kk, Nb_Exercice_Dates, Nb_Exercice_Dates,
   Ntraj, generator);
  ActStock(dt, Nb_Exercice_Dates, kk, r, Ntraj, ps, pd,
  pv, prho);
}


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

    VStock.array=X[ii];
    TPE[ii] = p->Compute(p->Par,&VStock);
    TPS[ii] =  TPE[ii];
  }

// comparison
CompCash(dt, Nb_Exercice_Dates, r, Ntraj, Nb_Exercice_Da
  tes);

// Backward induction
for (jj = Nb_Exercice_Dates-1; jj > 0; jj--){
      // - time step loop

  // Price of the asset
  for (kk = 0; kk < Dim; kk++){
    for (ii = 0; ii < Ntraj; ii++) {
      WLoc[kk][ii] = W[kk][ii];
    }
    BrownCMRG(dt, kk, Nb_Exercice_Dates, jj,  Ntraj,     generator);
    ActStock(dt, jj, kk, r, Ntraj, ps, pd, pv, prho);
  }

  for (ii = 0; ii < Ntraj; ii++) {
    VStock.array=X[ii];
    TPE[ii]=p->Compute(p->Par,&VStock) ;
  }
```

```
  // continuation
  CompDenom(dt,jj,Ntraj,Dim,pv);
  for (ii = 0; ii < Ntraj; ii++) {
    CompD(dt,jj,ii,Ntraj,Dim,pv);
    CompN(Ntraj);
    CompLambda(ii,Ntraj);
  }

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

  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,
  BS_Spot));
*error = 1.96*sqrt((exp(-2*r*dt*dt)/(Ntraj-1))*(sum2 - (
  sum*sum)/Ntraj))/sqrt(Ntraj);

// Free the memory
free_memory(Ntraj, Dim);
pnl_mat_free(&CorrM);
for (ii=0;ii<Dim;ii++)
  free(prho[ii]);
free(prho);
free(ps);
free(pd);
free(pv);

return OK;
```

```
}


int CALC(MC_MalliavinAmer)(void *Opt, void *Mod, Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r;
  int i, res;
  PnlVect *divid = pnl_vect_create(ptMod->Size.Val.V_PINT);
  PnlVect *spot, *sig;

  spot = pnl_vect_compact_to_pnl_vect (ptMod->S0.Val.V_PNLV
    ECTCOMPACT);
  sig = pnl_vect_compact_to_pnl_vect (ptMod->Sigma.Val.V_PN
    LVECTCOMPACT);
  for(i=0; i<ptMod->Size.Val.V_PINT; i++)
    pnl_vect_set (divid, i,
                  log(1.+ pnl_vect_compact_get (ptMod->Div
    id.Val.V_PNLVECTCOMPACT, i)/100.));

  r= log(1.+ptMod->R.Val.V_DOUBLE/100.);
  pnl_rand_init(Met->Par[1].Val.V_ENUM.value, 1, 1);
  res=ImprovedMalliavin(spot,
                        ptOpt->PayOff.Val.V_NUMFUNC_ND,
                        ptOpt->Maturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,
                        r, divid, sig,
                        ptMod->Rho.Val.V_DOUBLE,
                        Met->Par[0].Val.V_LONG,
                        Met->Par[1].Val.V_ENUM.value,Met->
    Par[2].Val.V_INT,
                        &(Met->Res[0].Val.V_DOUBLE),&(Met->
    Res[1].Val.V_DOUBLE));

  pnl_vect_free(&divid);
  pnl_vect_free (&spot);
  pnl_vect_free (&sig);
  return res;
}
```

```
static int CHK_OPT(MC_MalliavinAmer)(void *Opt, void *Mod)
{
  Option* ptOpt= (Option*)Opt;
  TYPEOPT* opt= (TYPEOPT*)(ptOpt->TypeOpt);
  if ((opt->EuOrAm).Val.V_BOOL==AMER)
    return OK;
  return  WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;
      Met->HelpFilenameHint = "mc_longstaffschwatrz_nd";
      Met->Par[0].Val.V_LONG=1000;
      Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[2].Val.V_INT=10;

    }
  return OK;
}

PricingMethod MET(MC_MalliavinAmer)=
{
  "MC_MalliavinAmer",
  {{"N iterations",LONG,{100},ALLOW},
      {"RandomGenerator",ENUM,{0},ALLOW},
      {"Number of Exercise Dates",INT,{100},ALLOW},
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_MalliavinAmer),
  {{"Price",DOUBLE,{100},FORBID},
      {"Error",DOUBLE,{100},FORBID},
      {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_MalliavinAmer),
  CHK_mc,
  MET(Init)
};
```

References