Help

```
#include <stdlib.h>
#include  "bs2d_std2d.h"
#include "error_msg.h"
#define BIG_DOUBLE 1.0e6

int CALC(DynamicHedgingSimulator)(void *Opt,void *Mod,PricingMethod *Met,DynamicTest *Test)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  int  type_generator,error,init_mc;
  long path_number,hedge_number,i,j;
  double g,step_hedge,initial_stock1,initial_stock2,initial_time,stock1,stock2,
    selling_price,delta1,delta2,previous_delta1,previous_delta2;
  double cash_account,stock1_account,stock2_account,cash_rate,stock1_rate,stock2_rate;
  double pl_sample,mean_pl,var_pl,min_pl,max_pl;
  double exp_trend1xh,exp_trend2xh,sigma1xsqrth,correl2xsqrth,free2xsqrth;
  double r,divid1,divid2;

  /* Variables needed for exercise time of american options */
  int n_us;
  double sigma_us, /* Square deviation for the simulation of n_us */
    m_us;    /* Mean --- */

  /* Variables needed for Brownian bridges */
  double Bridge1=0., d_Bridge1, Bridge1T1, Stock1T1, sigma1, mu1; /* First Brownian bridge */
  double Bridge2=0., d_Bridge2, Bridge2T1, Stock2T1, sigma2, mu2; /* Second Brownian bridge */
  double currentT, H, T1, correl2, free2;

  /* Variables needed for Graphic outputs */
  double *stock1_array, *pl_array, *stock2_array, current_mean_pl, median_pl=0.;
```

```
int  k;
long size;
double current_date;

/******** Initialization of the test's parameters *******
  */
initial_stock1=ptMod->S01.Val.V_PDOUBLE;
initial_stock2=ptMod->S02.Val.V_PDOUBLE;
initial_time=ptMod->T.Val.V_DATE;
current_date=ptMod->T.Val.V_DATE;

type_generator=Test->Par[0].Val.V_INT;
path_number=Test->Par[1].Val.V_LONG;
hedge_number=Test->Par[2].Val.V_LONG;

step_hedge=(ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DA
  TE)/(double)hedge_number;

r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
divid1=log(1.+ptMod->Divid1.Val.V_DOUBLE/100.);
divid2=log(1.+ptMod->Divid2.Val.V_DOUBLE/100.);
cash_rate=exp(r*step_hedge);
stock1_rate=exp(divid1*step_hedge)-1.;
stock2_rate=exp(divid2*step_hedge)-1.;

sigma1xsqrth=ptMod->Sigma1.Val.V_PDOUBLE*sqrt(step_hedge)
  ;
exp_trend1xh=exp(ptMod->Mu2.Val.V_DOUBLE*step_hedge-SQR(
  sigma1xsqrth)/2.);
exp_trend2xh=exp((ptMod->Mu2.Val.V_DOUBLE-SQR(ptMod->Si
  gma2.Val.V_PDOUBLE)/2.0)*step_hedge);
correl2xsqrth=ptMod->Rho.Val.V_RGDOUBLE*ptMod->Sigma2.Val
  .V_PDOUBLE*sqrt(step_hedge);
correl2=ptMod->Rho.Val.V_RGDOUBLE*ptMod->Sigma2.Val.V_PDO
  UBLE;
free2xsqrth=sqrt(1.0-SQR(ptMod->Rho.Val.V_RGDOUBLE))*pt
  Mod->Sigma2.Val.V_PDOUBLE*sqrt(step_hedge);
free2=sqrt(1.0-SQR(ptMod->Rho.Val.V_RGDOUBLE))*ptMod->Si
  gma2.Val.V_PDOUBLE;

mean_pl=0.0;
```

```
var_pl=0.0;
min_pl=BIG_DOUBLE;
max_pl=-BIG_DOUBLE;


init_mc= pnl_rand_init(type_generator,(int)hedge_number,
  path_number);
/* Test after initialization for the generator */
if(init_mc == OK)
  {

    /* Determining exercise time for american options */
    m_us=0.0;
    sigma_us=0.0;

    n_us=hedge_number;
    if ((ptOpt->EuOrAm.Val.V_BOOL==EURO) || (Test->Par[3]
  .Val.V_BOOL == 0)) /* european */
      n_us=hedge_number;

    else if (Test->Par[3].Val.V_BOOL == 1) /* uniform on
  [0,hedge_number] */
      n_us=(int)floor(pnl_rand_uni(type_generator)*(
  double)hedge_number)+1;

    else if (Test->Par[3].Val.V_BOOL == 2) /* "Integer"
  gaussian centered on the middle of [0,hedge_number] */
      {
        m_us=(int)floor(hedge_number/2.0);
        sigma_us=(int)floor(hedge_number/6.0);
        n_us=(int)floor(m_us+sigma_us*pnl_rand_normal(ty
  pe_generator))+1;
        if (n_us<0)
          n_us=0;
        else if (n_us>hedge_number)
          n_us=hedge_number;
      };


    /* Some initializations for Brownian Bridges */
    sigma1=ptMod->Sigma1.Val.V_PDOUBLE;
    sigma2=ptMod->Sigma2.Val.V_PDOUBLE;
    mu1=ptMod->Mu1.Val.V_DOUBLE;
    mu2=ptMod->Mu2.Val.V_DOUBLE;
    T1=Test->Par[6].Val.V_DATE-ptMod->T.Val.V_DATE;
    Stock1T1=Test->Par[5].Val.V_PDOUBLE;
    Stock2T1=Test->Par[7].Val.V_PDOUBLE;
    Bridge1T1=(log(Stock1T1/initial_stock1)-(mu1-SQR(sigma1)/2.0)*T1)/sigma1;
    Bridge2T1=(log(Stock2T1/initial_stock2)-(mu2-SQR(sigma2)/2.0)*T1)/sigma2;


    /* Graphic outputs initializations and dynamical memory allocutions */
    current_mean_pl=0.0;
    size=hedge_number+1;

    if ((stock1_array= malloc(size*sizeof(double)))==NULL)
      return MEMORY_ALLOCATION_FAILURE;
    if ((stock2_array= malloc(size*sizeof(double)))==NULL)
      return MEMORY_ALLOCATION_FAILURE;
    if ((pl_array= malloc(size*sizeof(double)))==NULL)
      return MEMORY_ALLOCATION_FAILURE;

    for (k=5;k<=14;k++)
      {
        pnl_vect_resize (Test->Res[k].Val.V_PNLVECT, size); /* Time */
      }


    pnl_vect_resize (Test->Res[15].Val.V_PNLVECT, 2); /* Brownian Target*/
    pnl_vect_resize (Test->Res[16].Val.V_PNLVECT, 2); /* Brownian Target*/
    pnl_vect_resize (Test->Res[17].Val.V_PNLVECT, 2); /* exercise Time*/

    for (k=0;k<=hedge_number;k++)
```

```
    Test->Res[5].Val.V_PNLVECT->array[k]=current_date+
k*step_hedge;

  if (Test->Par[4].Val.V_BOOL==1)
    {
      Test->Res[15].Val.V_PNLVECT->array[0]=current_da
te+T1;
      Test->Res[15].Val.V_PNLVECT->array[1]=Stock1T1;
      Test->Res[16].Val.V_PNLVECT->array[0]=current_da
te+T1;
      Test->Res[16].Val.V_PNLVECT->array[1]=Stock2T1;
    }
  else
    {
      Test->Res[15].Val.V_PNLVECT->array[0]=current_da
te;
      Test->Res[15].Val.V_PNLVECT->array[1]=initial_sto
ck1;
      Test->Res[16].Val.V_PNLVECT->array[0]=current_da
te;
      Test->Res[16].Val.V_PNLVECT->array[1]=initial_sto
ck2;
    }

  /******** Trajectories of the stock ********/
  for (i=0;i<path_number;i++)
    {
      /* computing selling-price and delta */
      ptMod->S01.Val.V_PDOUBLE= initial_stock1;
      ptMod->S02.Val.V_PDOUBLE= initial_stock2;
      ptMod->T.Val.V_DATE= initial_time;
      if ((error=(Met->Compute)(Opt,Mod,Met)))
        {
          ptMod->T.Val.V_DATE=initial_time;
          ptMod->S01.Val.V_PDOUBLE=initial_stock1;
          ptMod->S02.Val.V_PDOUBLE= initial_stock2;
          return error;
        };
      selling_price=Met->Res[0].Val.V_DOUBLE;
      delta1=Met->Res[1].Val.V_DOUBLE;
      delta2=Met->Res[2].Val.V_DOUBLE;
      /* computing cash_account and stock_account */
      cash_account=selling_price-delta1*initial_stock1-
delta2*initial_stock2;
      stock1_account=delta1*initial_stock1;
      stock2_account=delta2*initial_stock2;

      stock1=initial_stock1;
      stock2=initial_stock2;

      stock1_array[0]=stock1;
      stock2_array[0]=stock2;
      pl_array[0]=0;

      /* Brownian bridge's initialization */
      if (Test->Par[4].Val.V_BOOL==1) /* With brownian
bridge */
        {
          H=0.0;
          Bridge1=0.0;
          Bridge2=0.0;
        }

      /******** Dynamic Hedge ********/
      for (j=1;(j<hedge_number) && (j<n_us);j++)
        {
          ptMod->T.Val.V_DATE=ptMod->T.Val.V_DATE+step_
hedge;

          previous_delta1=delta1;
          previous_delta2=delta2;

          /* Capitalization of cash_account and yield
ing dividends */
          cash_account*=cash_rate;
          stock1_account*=stock1_rate;
          stock2_account*=stock2_rate;
          cash_account+=stock1_account+stock2_account;

          /* computing the new stock's value */
          currentT=j*step_hedge;
          H=step_hedge/(T1-currentT);
          if ((currentT<T1)&&(H<=1)&&(Test->Par[4].Val.
V_BOOL==1)) /* Using Brownian Bridge */
            {
              d_Bridge1=(Bridge1T1-Bridge1)*H+sqrt(step
_hedge*(1-H))*pnl_rand_normal(type_generator);
              Bridge1+=d_Bridge1;

              d_Bridge2=(Bridge2T1-Bridge2)*H+sqrt(step
_hedge*(1-H))*pnl_rand_normal(type_generator);
              Bridge2+=d_Bridge2;

              stock1*=exp_trend1xh*exp(sigma1*d_Bridge1
);
              stock2*=exp_trend2xh*exp(correl2*d_Brid
ge1+free2*d_Bridge2);
            }
          else /* After or without using brownian brid
ge */
            {
              g=pnl_rand_normal(type_generator);
              stock1*=exp_trend1xh*exp(sigma1xsqrth*g);
              stock2*=exp_trend2xh*exp(correl2xsqrth*g+
free2xsqrth*pnl_rand_normal(type_generator));
            }

          /* computing the new selling-price and the ne
w delta */
          ptMod->S01.Val.V_PDOUBLE=stock1;
          ptMod->S02.Val.V_PDOUBLE=stock2;
          if ((error=(Met->Compute)(Opt,Mod,Met)))
            {
              ptMod->T.Val.V_DATE=initial_time;

              ptMod->S01.Val.V_PDOUBLE=initial_stock1;
              ptMod->S02.Val.V_PDOUBLE= initial_stock2;
              return error;
            };
          delta1=Met->Res[1].Val.V_DOUBLE;
          delta2=Met->Res[2].Val.V_DOUBLE;
          /* computing new cash_account and new stock_
account */
          cash_account-=(delta1-previous_delta1)*stock1
+(delta2-previous_delta2)*stock2;
          stock1_account=delta1*stock1;
          stock2_account=delta2*stock2;

          stock1_array[j]=stock1;
          stock2_array[j]=stock2;
          pl_array[j]=cash_account-Met->Res[0].Val.V_
DOUBLE+delta1*stock1+delta2*stock2;

        } /*j*/

      /******** Last hedge *******/
      /* Capitalization of cash_account and yielding
dividends */
      cash_account*=cash_rate;
      stock1_account*=stock1_rate;
      stock2_account*=stock2_rate;

      /* computing the last stock's value */
      currentT=j*step_hedge;
      H=step_hedge/(T1-currentT);
      if ((currentT<T1)&&(H<=1)&&(Test->Par[4].Val.V_BO
OL==1)) /* Using Brownian Bridge */
        {
          d_Bridge1=(Bridge1T1-Bridge1)*H+sqrt(step_hed
ge*(1-H))*pnl_rand_normal(type_generator);
          Bridge1+=d_Bridge1;

          d_Bridge2=(Bridge2T1-Bridge2)*H+sqrt(step_hed
ge*(1-H))*pnl_rand_normal(type_generator);
          Bridge2+=d_Bridge2;

          stock1*=exp_trend1xh*exp(sigma1*d_Bridge1);
          stock2*=exp_trend2xh*exp(correl2*d_Bridge1+
free2*d_Bridge2);
        }
      else
        {
```

```
     g=pnl_rand_normal(type_generator);
     stock1*=exp_trend1xh*exp(sigma1xsqrth*g);
     stock2*=exp_trend2xh*exp(correl2xsqrth*g+fre
e2xsqrth*pnl_rand_normal(type_generator));
    }

   /* Capitalization of cash_account and computing
the P&L using the PayOff*/
   cash_account=cash_account-((ptOpt->PayOff.Val.V_
NUMFUNC_2)->Compute)((ptOpt->PayOff.Val.V_NUMFUNC_2)->Par,stock1,
stock2)
    +delta1*stock1+delta2*stock2;
   pl_sample=cash_account*exp((hedge_number-n_us)*
log(cash_rate));

   if (n_us<hedge_number)
    for (k=n_us;k<=hedge_number;k++)
     {
      stock1_array[k]=stock1_array[n_us-1];
      pl_array[k]=pl_array[n_us-1];
      stock2_array[k]=stock2_array[n_us-1];
     }
   else
    {
     stock1_array[hedge_number]=stock1;
     pl_array[hedge_number]=pl_sample;
     stock2_array[hedge_number]=stock2;
    }

   mean_pl=mean_pl+pl_sample;
   var_pl=var_pl+SQR(pl_sample);
   min_pl=MIN(pl_sample,min_pl);
   max_pl=MAX(pl_sample,max_pl);


   /* Selection of trajectories (Spot and P&L) for
graphic outputs */
   if (i==0)
    {
     for (k=0; k<=hedge_number; k++)
      {
              Test->Res[6].Val.V_PNLVECT->array[k]=sto
ck1_array[k];
              Test->Res[7].Val.V_PNLVECT->array[k]=sto
ck1_array[k];
              Test->Res[8].Val.V_PNLVECT->array[k]=sto
ck1_array[k];
              Test->Res[9].Val.V_PNLVECT->array[k]=pl_
array[k];
              Test->Res[10].Val.V_PNLVECT->array[k]=pl_
array[k];
              Test->Res[11].Val.V_PNLVECT->array[k]=pl_
array[k];
              Test->Res[12].Val.V_PNLVECT->array[k]=sto
ck2_array[k];
              Test->Res[13].Val.V_PNLVECT->array[k]=sto
ck2_array[k];
              Test->Res[14].Val.V_PNLVECT->array[k]=sto
ck2_array[k];

            }
          median_pl=pl_sample;
        }
      else
        {
          current_mean_pl=mean_pl/i;
          if (pl_sample==min_pl)
            {
              for (k=0; k<=hedge_number; k++)
                {
                  Test->Res[6].Val.V_PNLVECT->array[k]=
stock1_array[k];
                  Test->Res[9].Val.V_PNLVECT->array[k]=
pl_array[k];
                  Test->Res[12].Val.V_PNLVECT->array[k]
=stock2_array[k];
                }
            }
          else if (pl_sample==max_pl)
            {
              for (k=0; k<=hedge_number; k++)
                {
```

```
         Test->Res[7].Val.V_PNLVECT->array[k]=
stock1_array[k];
         Test->Res[10].Val.V_PNLVECT->array[k]
=pl_array[k];
         Test->Res[13].Val.V_PNLVECT->array[k]
=stock2_array[k];
        }
      }
     else if (SQR(pl_sample-current_mean_pl) < SQ
R(median_pl-current_mean_pl))
      {
       median_pl=pl_sample;
       for (k=0; k<=hedge_number; k++)
        {
         Test->Res[8].Val.V_PNLVECT->array[k]=
stock1_array[k];
         Test->Res[11].Val.V_PNLVECT->array[k]
=pl_array[k];
         Test->Res[14].Val.V_PNLVECT->array[k]
=stock2_array[k];
        }
      }
    }

  }/*i*/

 Test->Res[17].Val.V_PNLVECT->array[0]=current_date+n_
us*step_hedge;
 Test->Res[17].Val.V_PNLVECT->array[1]=initial_stock1;

 free(stock1_array);
 free(pl_array);
 free(stock2_array);


 mean_pl=mean_pl/(double)path_number;
 var_pl=var_pl/(double)path_number-SQR(mean_pl);

 Test->Res[0].Val.V_DOUBLE=mean_pl;
 Test->Res[1].Val.V_DOUBLE=var_pl;
 Test->Res[2].Val.V_DOUBLE=min_pl;
```

```
      Test->Res[3].Val.V_DOUBLE=max_pl;

      ptMod->T.Val.V_DATE=initial_time;
      ptMod->S01.Val.V_PDOUBLE=initial_stock1;
      ptMod->S02.Val.V_PDOUBLE=initial_stock2;
      Test->Res[4].Val.V_DOUBLE=current_date+n_us*step_hed
    ge;

      return OK;
    }
  else return init_mc;
}

static int TEST(Init)(DynamicTest *Test,Option *Opt)
{
  static int first=1;
  TYPEOPT* pt=(TYPEOPT*)(Opt->TypeOpt);
  int i;

  if (first)
    {
      first=0;
      Test->Par[0].Val.V_INT=0;         /* Random     Generator */
      Test->Par[1].Val.V_LONG=1000;     /* PathNumber */
      Test->Par[2].Val.V_LONG=250;      /* HedgeNumber */
      Test->Par[3].Val.V_BOOL=0;            /* exerciseTyp
    e */
      Test->Par[4].Val.V_BOOL=1;            /* Brownian Br
    idge */
      Test->Par[5].Val.V_PDOUBLE=90.;       /* SpotTarget1
    */
      Test->Par[6].Val.V_DATE=0.5;      /* TimeTarget */
      Test->Par[7].Val.V_PDOUBLE=110.;  /* SpotTarget2 */

      for ( i=5 ; i<=17 ; i++ )
        {
          Test->Res[i].Val.V_PNLVECT = pnl_vect_create (0);
        }
    }
  if (pt->EuOrAm.Val.V_INT==EURO)
    Test->Par[3].Viter=IRRELEVANT;
  return OK;
}
int CHK_TEST(test)(void *Opt, void *Mod, PricingMethod *
    Met)
{
  return OK;
}

DynamicTest MOD_OPT(test)=
{
  "bs2d_std2d_test",
  {{"RandomGenerator",INT,{100},ALLOW},
   {"PathNumber",LONG,{100},ALLOW},
   {"HedgeNumber",LONG,{100},ALLOW},
   {"exerciseType",BOOL,{100},ALLOW},       /* 0: european;
     1: american "uniform"; 2: american "gaussian" */
   {"BrownianBridge",BOOL,{100},ALLOW},     /* 0: without
    brownian bridge; 1: with brownian bridge */
   {"SpotTarget1",PDOUBLE,{100},ALLOW},
   {"TimeTarget",DATE,{100},ALLOW},
   {"SpotTarget2",PDOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},

  CALC(DynamicHedgingSimulator),
  {{"Mean_P&l",DOUBLE,{100},FORBID},
   {"Var_P&l",DOUBLE,{100},FORBID},
   {"Min_P&l",DOUBLE,{100},FORBID},
   {"Max_P&l",DOUBLE,{100},FORBID},
   {"exerciseTime",DOUBLE,{100},FORBID},

   {"Time",PNLVECT,{100},FORBID},
   {"Stock1min",PNLVECT,{0},FORBID},
   {"Stock1max",PNLVECT,{0},FORBID},
   {"Stock1mean",PNLVECT,{0},FORBID},
   {"PLmin",PNLVECT,{0},FORBID},
   {"PLmax",PNLVECT,{0},FORBID},
   {"PLmean",PNLVECT,{0},FORBID},
   {"Stock2min",PNLVECT,{0},FORBID},
   {"Stock2max",PNLVECT,{0},FORBID},
   {"Stock2mean",PNLVECT,{0},FORBID},
   {"SpotTarget1",PNLVECT,{0},FORBID},
   {"SpotTarget2",PNLVECT,{0},FORBID},
   {"exerciseTime",PNLVECT,{0},FORBID},

   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_TEST(test),
  CHK_ok,
  TEST(Init)
};
```

References