```
   Help
#include  "bs2d_std2d.h"

static DoubleArray ttime=
  {
  0,NULL
  };
static DoubleArray stock1min=
  {
  0,NULL
  };
static DoubleArray stock1max=
  {
  0,NULL
  };
static DoubleArray stock1mean=
  {
  0,NULL
  };
static DoubleArray plmin=
  {
  0,NULL
  };
static DoubleArray plmax=
  {
  0,NULL
  };
static DoubleArray plmean=
  {
  0,NULL
  };
static DoubleArray stock2min=
  {
  0,NULL
  };
static DoubleArray stock2max=
  {
  0,NULL
  };
static DoubleArray stock2mean=
  {
```

```
  0,NULL
  };
static DoubleArray target1=
  {
  0,NULL
  };
static DoubleArray target2=
  {
  0,NULL
  };
static DoubleArray exercise=
  {
  0,NULL
  };

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_sto
    ck2,initial_time,stock1,stock2,
    selling_price,delta1,delta2,previous_delta1
    ,previous_delta2;
  double cash_account,stock1_account,stock2_ac
    count,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 am
    erican 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, d_Bridge1, Bridge1T1, Stock1T1
  , sigma1, mu1; /* First Brownian bridge */
double Bridge2, 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;
int  k;
long size;
double current_date;

/******** Initialization of the test's para
  meters ********/
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_DATE)/(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->Sigma2.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->Sig
  ma2.Val.V_PDOUBLE;
free2xsqrth=sqrt(1.0-SQR(ptMod->Rho.Val.V_RG
  DOUBLE))*ptMod->Sigma2.Val.V_PDOUBLE*sqrt(step_hedge);
free2=sqrt(1.0-SQR(ptMod->Rho.Val.V_RGDOUBLE))
  *ptMod->Sigma2.Val.V_PDOUBLE;

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


init_mc= InitGenerator(type_generator,(int)hed
  ge_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) /* uni
  form on [0,hedge_number] */
  n_us=(int)floor(Uniform(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*Gauss_Abramow
  itzStegun(type_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_DA
  TE;
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 dynami
  cal 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++)
  {
  if ((Test->Res[k].Val.V_DOUBLEARRAY->array=
  malloc(size*sizeof(double)))==NULL)  /* Time */
    return MEMORY_ALLOCATION_FAILURE;
  else
    Test->Res[k].Val.V_DOUBLEARRAY->size=siz
  e;
  }

if ((Test->Res[15].Val.V_DOUBLEARRAY->array=
  malloc(2*sizeof(double)))==NULL)
  return MEMORY_ALLOCATION_FAILURE;    /*Br
  ownian Target*/
else
  Test->Res[15].Val.V_DOUBLEARRAY->size=2;
if ((Test->Res[16].Val.V_DOUBLEARRAY->array=
  malloc(2*sizeof(double)))==NULL)
  return MEMORY_ALLOCATION_FAILURE;    /*Br
  ownian Target*/
else
  Test->Res[16].Val.V_DOUBLEARRAY->size=2;
if ((Test->Res[17].Val.V_DOUBLEARRAY->array=
  malloc(2*sizeof(double)))==NULL)
  return MEMORY_ALLOCATION_FAILURE;    /*
  exercise Time*/
else
  Test->Res[17].Val.V_DOUBLEARRAY->size=2;


for (k=0;k<=hedge_number;k++)
  Test->Res[5].Val.V_DOUBLEARRAY->array[k]=cu
  rrent_date+k*step_hedge;

if (Test->Par[4].Val.V_BOOL==1)
  {
  Test->Res[15].Val.V_DOUBLEARRAY->array[0]=
  current_date+T1;
  Test->Res[15].Val.V_DOUBLEARRAY->array[1]=
```

```
  Stock1T1;
  Test->Res[16].Val.V_DOUBLEARRAY->array[0]=
  current_date+T1;
  Test->Res[16].Val.V_DOUBLEARRAY->array[1]=
  Stock2T1;
  }
else
  {
  Test->Res[15].Val.V_DOUBLEARRAY->array[0]=
  current_date;
  Test->Res[15].Val.V_DOUBLEARRAY->array[1]=
  initial_stock1;
  Test->Res[16].Val.V_DOUBLEARRAY->array[0]=
  current_date;
  Test->Res[16].Val.V_DOUBLEARRAY->array[1]=
  initial_stock2;
  }

/******** 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_sto
  ck2;
    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 br
ownian 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 y
ielding dividends */
  cash_account*=cash_rate;
  stock1_account*=stock1_rate;
  stock2_account*=stock2_rate;
  cash_account+=stock1_account+stock2_ac
count;

  /* 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))*Gauss_AbramowitzStegun(type_gen
erator);
    Bridge1+=d_Bridge1;

    d_Bridge2=(Bridge2T1-Bridge2)*H+sqrt(
step_hedge*(1-H))*Gauss_AbramowitzStegun(type_gen
erator);
    Bridge2+=d_Bridge2;

    stock1*=exp_trend1xh*exp(sigma1*d_Br
idge1);
    stock2*=exp_trend2xh*exp(correl2*d_Br
idge1+free2*d_Bridge2);
    }
    else /* After or without using browni
an bridge */
    {
    g=Gauss_AbramowitzStegun(type_genera
tor);
    stock1*=exp_trend1xh*exp(sigma1xsqrt
h*g);
    stock2*=exp_trend2xh*exp(correl2xsqrt
h*g+free2xsqrth*Gauss_AbramowitzStegun(type_gen
erator));
    }

  /* computing the new selling-price and
the new 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_sto
ck1;
    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 yield
ing 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_BOOL==1)) /* Using Brownian Bridge */
  {
    d_Bridge1=(Bridge1T1-Bridge1)*H+sqrt(
step_hedge*(1-H))*Gauss_AbramowitzStegun(type_gen
erator);
    Bridge1+=d_Bridge1;

    d_Bridge2=(Bridge2T1-Bridge2)*H+sqrt(
step_hedge*(1-H))*Gauss_AbramowitzStegun(type_gen
erator);
    Bridge2+=d_Bridge2;
```

```
    stock1*=exp_trend1xh*exp(sigma1*d_Br
idge1);
    stock2*=exp_trend2xh*exp(correl2*d_Br
idge1+free2*d_Bridge2);
  }
  else
  {
  g=Gauss_AbramowitzStegun(type_generator)
;
  stock1*=exp_trend1xh*exp(sigma1xsqrth*g)
;
  stock2*=exp_trend2xh*exp(correl2xsqrth*g
+free2xsqrth*Gauss_AbramowitzStegun(type_genera
tor));
  }

/* Capitalization of cash_account and compu
ting 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_DOUBLEARRAY->array
[k]=stock1_array[k];
    Test->Res[7].Val.V_DOUBLEARRAY->array
[k]=stock1_array[k];
    Test->Res[8].Val.V_DOUBLEARRAY->array
[k]=stock1_array[k];
    Test->Res[9].Val.V_DOUBLEARRAY->array
[k]=pl_array[k];
    Test->Res[10].Val.V_DOUBLEARRAY->ar
ray[k]=pl_array[k];
    Test->Res[11].Val.V_DOUBLEARRAY->ar
ray[k]=pl_array[k];
    Test->Res[12].Val.V_DOUBLEARRAY->ar
ray[k]=stock2_array[k];
    Test->Res[13].Val.V_DOUBLEARRAY->ar
ray[k]=stock2_array[k];
    Test->Res[14].Val.V_DOUBLEARRAY->ar
ray[k]=stock2_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_DOUBLEARRAY->ar
```

```
  ray[k]=stock1_array[k];
        Test->Res[9].Val.V_DOUBLEARRAY->ar
  ray[k]=pl_array[k];
        Test->Res[12].Val.V_DOUBLEARRAY->
  array[k]=stock2_array[k];
        }
      }
    else if (pl_sample==max_pl)
      {
      for (k=0; k<=hedge_number; k++)
        {
        Test->Res[7].Val.V_DOUBLEARRAY->ar
  ray[k]=stock1_array[k];
        Test->Res[10].Val.V_DOUBLEARRAY->
  array[k]=pl_array[k];
        Test->Res[13].Val.V_DOUBLEARRAY->
  array[k]=stock2_array[k];
        }
      }
    else if (SQR(pl_sample-current_mean_pl)
  < SQR(median_pl-current_mean_pl))
      {
      median_pl=pl_sample;
      for (k=0; k<=hedge_number; k++)
        {
        Test->Res[8].Val.V_DOUBLEARRAY->ar
  ray[k]=stock1_array[k];
        Test->Res[11].Val.V_DOUBLEARRAY->
  array[k]=pl_array[k];
        Test->Res[14].Val.V_DOUBLEARRAY->
  array[k]=stock2_array[k];
        }
      }
    }

  }/*i*/

Test->Res[17].Val.V_DOUBLEARRAY->array[0]=curr
  ent_date+n_us*step_hedge;
Test->Res[17].Val.V_DOUBLEARRAY->array[1]=ini
  tial_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_hedge;

  return OK;
  }
  else return init_mc;
  }

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

  if (first)
  {
  Test->Par[0].Val.V_INT=0;       /* Random G
    enerator */
  Test->Par[1].Val.V_LONG=1000;    /* PathNumb
    er */
  Test->Par[2].Val.V_LONG=250;    /* HedgeNumb
    er */
  Test->Par[3].Val.V_BOOL=0;      /* exercis
```

```
    eType */
    Test->Par[4].Val.V_BOOL=1;      /* Browni
    an Bridge */
  Test->Par[5].Val.V_PDOUBLE=90.;    /* SpotTa
    rget1 */
  Test->Par[6].Val.V_DATE=0.5;    /* TimeTarg
    et */
  Test->Par[7].Val.V_PDOUBLE=110.;  /* SpotTarg
    et2 */

  Test->Res[5].Val.V_DOUBLEARRAY=&ttime;
  Test->Res[6].Val.V_DOUBLEARRAY=&stock1min;

  Test->Res[7].Val.V_DOUBLEARRAY=&stock1max;

  Test->Res[8].Val.V_DOUBLEARRAY=&stock1mean;
  Test->Res[9].Val.V_DOUBLEARRAY=&plmin;
  Test->Res[10].Val.V_DOUBLEARRAY=&plmax;
  Test->Res[11].Val.V_DOUBLEARRAY=&plmean;
  Test->Res[12].Val.V_DOUBLEARRAY=&stock2min;
  Test->Res[13].Val.V_DOUBLEARRAY=&stock2max;

  Test->Res[14].Val.V_DOUBLEARRAY=&stock2mean;
  Test->Res[15].Val.V_DOUBLEARRAY=&target1;

  Test->Res[16].Val.V_DOUBLEARRAY=&target2;
  Test->Res[17].Val.V_DOUBLEARRAY=&exercise;

  first=0;
  }
    if (pt->EuOrAm.Val.V_INT==EURO)
        Test->Par[3].Viter=IRRELEVANT;

    return OK;
  }
int CHK_TEST(test)(void *Opt, void *Mod, Pricing
    Method *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 "gau
    ssian" */
  {"BrownianBridge",BOOL,100,ALLOW},     /* 0: w
    ithout brownian bridge; 1: with brownian bridge *
    /
  {"SpotTarget1",PDOUBLE,100,ALLOW},
  {"TimeTarget",DATE,100,ALLOW},
  {"SpotTarget2",PDOUBLE,100,ALLOW},
  {" ",END,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",DOUBLEARRAY,100,FORBID},
  {"Stock1min",DOUBLEARRAY,0,FORBID},
  {"Stock1max",DOUBLEARRAY,0,FORBID},
  {"Stock1mean",DOUBLEARRAY,0,FORBID},
  {"PLmin",DOUBLEARRAY,0,FORBID},
  {"PLmax",DOUBLEARRAY,0,FORBID},
  {"PLmean",DOUBLEARRAY,0,FORBID},
  {"Stock2min",DOUBLEARRAY,0,FORBID},
  {"Stock2max",DOUBLEARRAY,0,FORBID},
  {"Stock2mean",DOUBLEARRAY,0,FORBID},
  {"SpotTarget1",DOUBLEARRAY,0,FORBID},
  {"SpotTarget2",DOUBLEARRAY,0,FORBID},
  {"exerciseTime",DOUBLEARRAY,0,FORBID},

  {" ",END,0,FORBID}},
  CHK_TEST(test),
  CHK_ok,
```

```
TEST(Init)
};
```

# References