

[Help](#)

```
#include "bs1d_std.h"

static DoubleArray ttime=
{
    0,NULL
};
static DoubleArray stockmin=
{
    0,NULL
};
static DoubleArray stockmax=
{
    0,NULL
};
static DoubleArray stockmean=
{
    0,NULL
};
static DoubleArray plmin=
{
    0,NULL
};
static DoubleArray plmax=
{
    0,NULL
};
static DoubleArray plmean=
{
    0,NULL
};
static DoubleArray hedgetimemin=
{
    0,NULL
};
static DoubleArray hedgespotmin=
{
    0,NULL
};
static DoubleArray hedgetimemax=
{
```

```

        0,NULL
    };
    static DoubleArray hedgespotmax=
    {
        0,NULL
    };
    static DoubleArray hedgetimemean=
    {
        0,NULL
    };
    static DoubleArray hedgespotmean=
    {
        0,NULL
    };
    static DoubleArray deltamin=
    {
        0,NULL
    };
    static DoubleArray deltamax=
    {
        0,NULL
    };
    static DoubleArray deltamean=
    {
        0,NULL
    };

int CALC(DynamicHedgingSimulatorTempsdeter1)(void
    *Opt,void *Mod,PricingMethod *Met,DynamicTest *
    Test)
{

    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    int  type_generator,error;
    long path_number,step_number,hedge_number,i,
    j;
    double step,step_hedge,initial_stock,initial_

```

```

time,stock,selling_price,delta,previous_delta;
double cash_account,stock_account,cash_rate,
stock_rate;
double pl_sample,mean_pl,var_pl,min_pl,max_pl
;
double exp_trendxh,sigmaxsqrth;
double r,divid;
int hedgenow;

/* Variables needed for Graphic outputs */
double *stock_array, *pl_array, *hedge_time,
*hedge_spot, current_mean_pl, median_pl;
double *delta_array;
int k,indicehedge;
long size, size2;
double current_date;

/***** Initialization of the test's para
meters *****/
initial_stock=ptMod->S0.Val.V_PDOUBLE;
initial_time=ptMod->T.Val.V_DATE;

type_generator=Test->Par[0].Val.V_INT;
path_number=Test->Par[1].Val.V_LONG;
step_number=Test->Par[2].Val.V_LONG;
current_date=ptMod->T.Val.V_DATE;
hedge_number=Test->Par[3].Val.V_LONG;

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

r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
cash_rate=exp(r*step);
stock_rate=exp(divid*step)-1.;

sigmaxsqrth=ptMod->Sigma.Val.V_PDOUBLE*sqrt(
step);
exp_trendxh=exp(ptMod->Mu.Val.V_DOUBLE*step-0

```

```

        .5*SQR(sigmamaxsrth));

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

    InitGenerator (type_generator,1,path_number);

/* Graphic outputs initializations and dynamical memory allocutions */
    current_mean_pl=0.0;
    size=step_number+1;
    size2=hedge_number+1;

    if ((stock_array=malloc(size*sizeof(double)))
        ==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    if ((pl_array=malloc(size*sizeof(double)))==
        NULL)
        return MEMORY_ALLOCATION_FAILURE;
    if ((hedge_time=malloc(size2*sizeof(double)))
        ==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    if ((hedge_spot=malloc(size2*sizeof(double)))
        ==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    if ((delta_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)
            return MEMORY_ALLOCATION_FAILURE;
        else
            Test->Res[k].Val.V_DOUBLEARRAY->size=siz
e;
    }
    for (k=15;k<=20;k++)

```

```

{
  if ((Test->Res[k].Val.V_DOUBLEARRAY->array=
      malloc(size2*sizeof(double)))==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  else
    Test->Res[k].Val.V_DOUBLEARRAY->size=size2;
}

for (k=0;k<=step_number;k++) /* Time */
  Test->Res[5].Val.V_DOUBLEARRAY->array[k]=
    current_date+k*step;

/***** Trajectories of the stock *****/
for (i=0;i<path_number;i++)
{
  /* computing selling-price and delta */
  ptMod->T.Val.V_DATE=initial_time;
  ptMod->S0.Val.V_PDOUBLE=initial_stock;
  delta=0.;
  hedge_number=Test->Par[3].Val.V_LONG;

  if (error=(Met->Compute)(Opt,Mod,Met))
  {
    ptMod->T.Val.V_DATE=initial_time;
    ptMod->S0.Val.V_PDOUBLE=initial_stock
;
    return error;
  };
  selling_price=Met->Res[0].Val.V_DOUBLE;
  delta=Met->Res[1].Val.V_DOUBLE;
  delta_array[0]=delta;

  hedgenow=1;

  /* computing cash_account and stock_account
  */
  cash_account=selling_price-delta*initial_

```

```

stock;
    stock_account=delta*initial_stock;

    stock=initial_stock;
    stock_array[0]=stock;
    pl_array[0]=0;
    hedge_time[0]=0.;
    hedge_spot[0]=stock;
    indicehedge=1;

    /***** Dynamic Hedge *****/
    for (j=1;j<step_number;j++)
    {
        previous_delta=delta;

        /* Capitalization of cash_account and y
        ielding dividends */
        cash_account*=cash_rate;
        cash_account+=stock_rate*stock_accou
nt;

        /* computing the new stock's value */
        stock*=exp_trendxh*exp(sigmamaxsqrth*Gauss
_AbramowitzStegun(type_generator));

        /* computing the new selling-price and
        the new delta */
        ptMod->T.Val.V_DATE=ptMod->T.Val.V_DA
TE+step;
        ptMod->S0.Val.V_PDDOUBLE=stock;
        if (error=(Met->Compute)(Opt,Mod,Met)
)
        {
            ptMod->T.Val.V_DATE=initial_time;
            ptMod->S0.Val.V_PDDOUBLE=initial_
stock;

            return error;
        }
    }
};

```

```

        if ((hedge_number!=0)&&(j%(step_number/
hedge_number))==0)
        {
            delta=Met->Res[1].Val.V_DOUBLE;
            hedge_time[indicehedge]=ptMod->T.Val.
V_DATE;
            hedge_spot[indicehedge]=stock;
            indicehedge++;
        }

        delta_array[j]=delta;

        /* computing new cash_account and new
stock_account */
            cash_account-=(delta-previous_delta)*
stock;
            stock_account=delta*stock;

            stock_array[j]=stock;
            pl_array[j]=cash_account-Met->Res[0].
Val.V_DOUBLE+delta*stock;

        } /*j*/

    /***** Last hedge *****/
    /* Capitalization of cash_account and yield
ing dividends */
        cash_account*=cash_rate;
        cash_account+=stock_rate*stock_account;

    /* computing the new stock's value */
        stock*=exp_trendxh*exp(sigmaksqrth*Gauss_
AbramowitzStegun(type_generator));

    delta_array[step_number]=delta;
    hedge_time[hedge_number]=ptMod->T.Val.V_DA
TE;
    hedge_spot[hedge_number]=stock;

    /* Capitalization of cash_account and compu

```

```

ting the P&L using the PayOff*/
    cash_account=cash_account-((ptOpt->PayOff.
Val.V_NUMFUNC_1)->Compute)((ptOpt->PayOff.Val.V_
NUMFUNC_1)->Par,stock)+delta*stock;
    pl_sample=cash_account;

stock_array[step_number]=stock;
pl_array[step_number]=pl_sample;

    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<=step_number; k++)
        {
            Test->Res[6].Val.V_DOUBLEARRAY->
array[k]=stock_array[k];
            Test->Res[7].Val.V_DOUBLEARRAY->
array[k]=stock_array[k];
            Test->Res[8].Val.V_DOUBLEARRAY->
array[k]=stock_array[k];
            Test->Res[9].Val.V_DOUBLEARRAY->
array[k]=pl_array[k];
            Test->Res[10].Val.V_DOUBLEARRAY->
array[k]=pl_array[k];
            Test->Res[11].Val.V_DOUBLEARRAY->
array[k]=pl_array[k];
            Test->Res[12].Val.V_DOUBLEARRAY->ar
ray[k]=delta_array[k];
            Test->Res[13].Val.V_DOUBLEARRAY->ar
ray[k]=delta_array[k];
            Test->Res[14].Val.V_DOUBLEARRAY->ar
ray[k]=delta_array[k];
        }
        for (k=0; k<size2; k++)
        {

```



```

        Test->Res[15].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
        Test->Res[16].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
        Test->Res[17].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
        Test->Res[18].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
        Test->Res[19].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
        Test->Res[20].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
    }
    median_pl=pl_sample;
}
else
{
    current_mean_pl=mean_pl/i;
    if (pl_sample==min_pl)
    {
        for (k=0; k<=step_number; k++)
        {
            Test->Res[6].Val.V_DOUBLEARR
AY->array[k]=stock_array[k];
            Test->Res[9].Val.V_DOUBLEARR
AY->array[k]=pl_array[k];
            Test->Res[12].Val.V_DOUBLEARRAY->
array[k]=delta_array[k];
        }
        for (k=0; k<size2; k++)
        {
            Test->Res[15].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
            Test->Res[16].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
        }
    }
    else if (pl_sample==max_pl)
    {
        for (k=0; k<=step_number; k++)
        {

```

```

        Test->Res[7].Val.V_DOUBLEARR
AY->array[k]=stock_array[k];
        Test->Res[10].Val.V_DOUBLEARR
AY->array[k]=pl_array[k];
        Test->Res[13].Val.V_DOUBLEARRAY->
array[k]=delta_array[k];
    }
    for (k=0; k<size2; k++)
    {
        Test->Res[17].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
        Test->Res[18].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
    }
}
    else if (SQR(pl_sample-current_mean_
pl) < SQR(median_pl-current_mean_pl))
    {
        median_pl=pl_sample;
        for (k=0; k<=step_number; k++)
        {
            Test->Res[8].Val.V_DOUBLEARR
AY->array[k]=stock_array[k];
            Test->Res[11].Val.V_DOUBLEARR
AY->array[k]=pl_array[k];
            Test->Res[14].Val.V_DOUBLEARRAY->
array[k]=delta_array[k];
        }
        for (k=0; k<size2; k++)
        {
            Test->Res[19].Val.V_DOUBLEARRAY->
array[k]=hedge_time[k];
            Test->Res[20].Val.V_DOUBLEARRAY->
array[k]=hedge_spot[k];
        }
    }
}
} /*i*/

free(stock_array);
free(pl_array);

```

```

free(hedge_time);
free(hedge_spot);
free(delta_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;
Test->Res[4].Val.V_DOUBLE=median_pl;

ptMod->T.Val.V_DATE=initial_time;
ptMod->S0.Val.V_PDOUBLE=initial_stock;

return OK;
}

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 Generator */
Test->Par[1].Val.V_LONG=10;        /* PathNumb
er */
Test->Par[2].Val.V_LONG=200;       /* StepNumb
er */
Test->Par[3].Val.V_LONG=20;        /*hedgenu
mber*/
Test->Par[4].Vtype=END;

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

```

```

Test->Res[7].Val.V_DOUBLEARRAY=&stockmax;
Test->Res[8].Val.V_DOUBLEARRAY=&stockmean;
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=&deltamin;
Test->Res[13].Val.V_DOUBLEARRAY=&deltamax;
Test->Res[14].Val.V_DOUBLEARRAY=&deltamean;
Test->Res[15].Val.V_DOUBLEARRAY=&hedgetimem
in;
Test->Res[16].Val.V_DOUBLEARRAY=&hedgespotm
in;
Test->Res[17].Val.V_DOUBLEARRAY=&hedgetimemax
;
Test->Res[18].Val.V_DOUBLEARRAY=&hedgespotmax
;
Test->Res[19].Val.V_DOUBLEARRAY=&hedgetimemea
n;
Test->Res[20].Val.V_DOUBLEARRAY=&hedgespotmea
n;

Test->Res[21].Vtype=END;

first=0;
}

return OK;
}
int CHK_TEST(test3)(void *Opt, void *Mod, Pricing
Method *Met)
{
if ( (strcmp( Met->Name,"TR_PatryMartini")==0)
|| (strcmp( Met->Name,"TR_PatryMartini1")==0))
return WRONG;
else
return OK;
}

DynamicTest MOD_OPT(test3)=
{
"bs1d_std_test3",

```

```

{"RandomGenerator",INT,100,ALLOW},
{"PathNumber",LONG,1000,ALLOW},
{"StepNumber",LONG,200,ALLOW},
{"HedgeNumber",LONG,20,ALLOW},
{" ",END,0,FORBID}},

CALC(DynamicHedgingSimulatorTempsdeter1),

{"Mean_P&l",DOUBLE,100,FORBID},
{"Var_P&l",DOUBLE,100,FORBID},
{"Min_P&l",DOUBLE,100,FORBID},
{"Max_P&l",DOUBLE,100,FORBID},
{"Median_P&l",DOUBLE,100,FORBID},

{"Time",DOUBLEARRAY,100,FORBID},
{"Stockmin",DOUBLEARRAY,0,FORBID},
{"Stockmax",DOUBLEARRAY,0,FORBID},
{"Stockmean",DOUBLEARRAY,0,FORBID},
{"PLmin",DOUBLEARRAY,0,FORBID},
{"PLmax",DOUBLEARRAY,0,FORBID},
{"PLmean",DOUBLEARRAY,0,FORBID},
{"deltamin",DOUBLEARRAY,0,FORBID},
{"deltamax",DOUBLEARRAY,0,FORBID},
{"deltamean",DOUBLEARRAY,0,FORBID},
{"HedgeTimemin",DOUBLEARRAY,0,FORBID},
{"HedgeSpotmin",DOUBLEARRAY,0,FORBID},
{"HedgeTimemax",DOUBLEARRAY,0,FORBID},
{"HedgeSpotmax",DOUBLEARRAY,0,FORBID},
{"HedgeTimemean",DOUBLEARRAY,0,FORBID},
{"HedgeSpotmean",DOUBLEARRAY,0,FORBID},
{" ",END,0,FORBID}},

CHK_TEST(test3),
  CHK_ok,
  TEST(Init)
};

```

References