

## Help

```
#include <stdlib.h>
#include "bs1d_std.h"
#include "error_msg.h"
#define BIG_DOUBLE 1.0e6

int CALC(DynamicHedgingSimulatorPatry4)(void *Opt, void *Mod,
                                         PricingMethod *Met, DynamicTest *Test)
{
    TYPEOPT *ptOpt = (TYPEOPT *) Opt;
    TYPEMOD *ptMod = (TYPEMOD *) Mod;
    int type_generator, error;
    long path_number, hedge_number, i, j;
    double 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, sigmaxsqqrth;
    double r, divid;
    double temp;
    int indicehedge;
    int nbcouv;
    int sumnbcouv;

    /* Variables needed for Graphic outputs */
    double *stock_array, *pl_array, *hedge_time, *hedge_spot,
           current_mean_pl, median_pl = 0.;
    double *delta_array;
    int k;
    long size;
    double current_date;

    /****** Initialization of the test's parameters *****/
    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;
    hedge_number = Test->Par[2].Val.V_LONG;
```

```

current_date = ptMod->T.Val.V_DATE;

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_hedge);
stock_rate = exp(divid * step_hedge) - 1.;

sigmaxsqrth = ptMod->Sigma.Val.V_PDOUBLE * sqrt(step_hedge);
exp_trendxh = exp(ptMod->Mu.Val.V_DOUBLE * step_hedge - 0.5 * SQR(sigmaxsqrth)

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

pnl_rand_init(type_generator, 1, path_number);

/* Graphic outputs initializations and dynamical memory allocutions */
current_mean_pl = 0.0;
size = 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(size * sizeof(double))) == NULL)
    return MEMORY_ALLOCATION_FAILURE;
if ((hedge_spot = malloc(size * sizeof(double))) == NULL)
    return MEMORY_ALLOCATION_FAILURE;
if ((delta_array = malloc(size * sizeof(double))) == NULL)
    return MEMORY_ALLOCATION_FAILURE;

for (k = 9; k <= 24; k++)
{
    pnl_vect_resize(Test->Res[k].Val.V_PNLVECT, size);
}

```

```

for (k = 0; k <= hedge_number; k++) /* Time */
    Test->Res[9].Val.V_PNLVECT->array[k] = current_date + k * step_hedge;

sumnbcouv = 0;

/***** 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;
    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;

    /* 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;
    delta_array[0] = delta;
    hedge_time[0] = ptMod->T.Val.V_DATE;
    hedge_spot[0] = stock;
    indicehedge = 1;

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

        /* Capitalization of cash_account and yielding dividends */
        cash_account *= cash_rate;

```

```

cash_account += stock_rate * stock_account;

stock *=
    exp_trendxh * exp(sigmaxsqtrth * pnl_rand_normal(type_generator));

/* computing the new selling-price and the new delta */
ptMod->T.Val.V_DATE = ptMod->T.Val.V_DATE + step_hedge;
ptMod->S0.Val.V_PDOUBLE = stock;
if ((error = (Met->Compute)(Opt, Mod, Met)))
{
    ptMod->T.Val.V_DATE = initial_time;
    ptMod->S0.Val.V_PDOUBLE = initial_stock;
    return error;
};
temp = fabs((Met->Res[1].Val.V_DOUBLE - delta) / delta);
if (temp > Test->Par[3].Val.V_DOUBLE)
{
    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 */

nbcouv = indicehedge;
sumnbcouv += nbcouv;

for (j = indicehedge; j <= hedge_number; j++)
{
    hedge_time[j] = hedge_time[j - 1];
    hedge_spot[j] = hedge_spot[j - 1];

```

```

    }

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

    /* Computing the stock's last value */

    stock *=
        exp_trendxh * exp(sigmamaxsqtrh * 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_1)->Compute)((ptOpt->PayOff.Val.V_NUMFUNC_
            Par, stock) + delta * stock;
    pl_sample = cash_account;

    stock_array[hedge_number] = stock;
    pl_array[hedge_number] = pl_sample;
    delta_array[hedge_number] = delta;

    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[10].Val.V_PNLVECT->array[k] = stock_array[k];
            Test->Res[11].Val.V_PNLVECT->array[k] = stock_array[k];
            Test->Res[12].Val.V_PNLVECT->array[k] = stock_array[k];
            Test->Res[13].Val.V_PNLVECT->array[k] = pl_array[k];
        }
    }

```

```

Test->Res[14].Val.V_PNLVECT->array[k] = pl_array[k];
Test->Res[15].Val.V_PNLVECT->array[k] = pl_array[k];
Test->Res[16].Val.V_PNLVECT->array[k] = delta_array[k];
Test->Res[17].Val.V_PNLVECT->array[k] = delta_array[k];
Test->Res[18].Val.V_PNLVECT->array[k] = delta_array[k];
Test->Res[19].Val.V_PNLVECT->array[k] = hedge_time[k];
Test->Res[20].Val.V_PNLVECT->array[k] = hedge_spot[k];
Test->Res[21].Val.V_PNLVECT->array[k] = hedge_time[k];
Test->Res[22].Val.V_PNLVECT->array[k] = hedge_spot[k];
Test->Res[23].Val.V_PNLVECT->array[k] = hedge_time[k];
Test->Res[24].Val.V_PNLVECT->array[k] = hedge_spot[k];
}
Test->Res[5].Val.V_INT = nbcouv;
Test->Res[6].Val.V_INT = nbcouv;
Test->Res[7].Val.V_INT = nbcouv;
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[10].Val.V_PNLVECT->array[k] = stock_array[k];
Test->Res[13].Val.V_PNLVECT->array[k] = pl_array[k];
Test->Res[16].Val.V_PNLVECT->array[k] = delta_array[k];
Test->Res[19].Val.V_PNLVECT->array[k] = hedge_time[k];
Test->Res[20].Val.V_PNLVECT->array[k] = hedge_spot[k];
}
Test->Res[5].Val.V_INT = nbcouv;
}
else if (pl_sample == max_pl)
{
for (k = 0; k <= hedge_number; k++)
{
Test->Res[11].Val.V_PNLVECT->array[k] = stock_array[k];
Test->Res[14].Val.V_PNLVECT->array[k] = pl_array[k];
Test->Res[17].Val.V_PNLVECT->array[k] = delta_array[k];
Test->Res[21].Val.V_PNLVECT->array[k] = hedge_time[k];
Test->Res[22].Val.V_PNLVECT->array[k] = hedge_spot[k];
}
}
}
}

```

```

    }
    Test->Res[6].Val.V_INT = nbcouv;
  }
  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[12].Val.V_PNLVECT->array[k] = stock_array[k];
      Test->Res[15].Val.V_PNLVECT->array[k] = pl_array[k];
      Test->Res[18].Val.V_PNLVECT->array[k] = delta_array[k];
      Test->Res[23].Val.V_PNLVECT->array[k] = hedge_time[k];
      Test->Res[24].Val.V_PNLVECT->array[k] = hedge_spot[k];
    }
    Test->Res[7].Val.V_INT = nbcouv;
  }
}

} /*i */

Test->Res[8].Val.V_DOUBLE = sumnbcouv / (double) Test->Par[1].Val.V_LONG;

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;
    int i;

    if (first)
    {
        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_DOUBLE = 0.1; /* DeltaTarget */
        Test->Par[4].Vtype = PREMIA_NULLTYPE;

        for (i = 9 ; i <= 24 ; i++)
        {
            Test->Res[i].Val.V_PNLVECT = pnl_vect_create(0);
        }

        Test->Res[25].Vtype = PREMIA_NULLTYPE;

        first = 0;
    }
}
```

```
    return OK;
}
```

```
int CHK_TEST(test1)(void *Opt, void *Mod, PricingMethod *Met)
{
```

```
    if ((strcmp(Met->Name, "TR_PatryMartini") == 0) || (strcmp(Met->Name, "TR_Patr
        return WRONG;
    else
        return OK;
```



```

}

DynamicTest MOD_OPT(test1) =
{
  "bs1d_std_test1",
  {
    {"RandomGenerator", INT, { 100}, ALLOW},
    {"PathNumber", LONG, { 100}, ALLOW},
    {"HedgeNumber", LONG, { 100}, ALLOW},
    {"DeltaTarget", DOUBLE, { 0}, ALLOW},
    {" ", PREMIA_NULLTYPE, { 0}, FORBID}
  },
  CALC(DynamicHedgingSimulatorPatry4),
  {
    { "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}    ,
    { "NbHedgemin", INT, {100}, FORBID}      ,
    { "NbHedgemax", INT, {100}, FORBID}      ,
    { "NbHedgemean", INT, {100}, FORBID}      ,
    { "Mean of Number hedging", DOUBLE, {100}, FORBID}      ,
    { "Time", PNLVECT, {100} , FORBID}      ,
    { "Stockmin", PNLVECT, {0}, FORBID}      ,
    { "Stockmax", PNLVECT, {0}, FORBID}      ,
    { "Stockmean", PNLVECT, {0}, FORBID}      ,
    { "PLmin", PNLVECT, {0}, FORBID}          ,
    { "PLmax", PNLVECT, {0}, FORBID}          ,
    { "PLmean", PNLVECT, {0}, FORBID}          ,
    { "deltamin", PNLVECT, {0}, FORBID}      ,
    { "deltamax", PNLVECT, {0}, FORBID}      ,
    { "deltamean", PNLVECT, {0}, FORBID}      ,
    { "HedgeTimemin", PNLVECT, {0}, FORBID}    ,
    { "HedgeSpotmin", PNLVECT, {0}, FORBID}    ,
    { "HedgeTimemax", PNLVECT, {0}, FORBID}    ,
    { "HedgeSpotmax", PNLVECT, {0}, FORBID}    ,
    { "HedgeTimemean", PNLVECT, {0}, FORBID}    ,
    { "HedgeSpotmean", PNLVECT, {0}, FORBID}    ,
    { " ", PREMIA_NULLTYPE, {0}, FORBID}
  },

```

```
    CHK_TEST(test1), CHK_ok, TEST(Init)
};
```