Help

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

int CALC(DynamicHedgingSimulatorPatry1)(void *Opt, void *Mod, PricingMethod *Met
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  int  type_generator, error;
  long path_number, step_number, hedge_number, i, j;
  double step_hedge, initial_stock, initial_time, stock, selling_price, delta, p
  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, med
  double *delta_array;
  int  k, indicehedge;
  long size, size2;
  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;
  step_number = Test->Par[2].Val.V_LONG;
  current_date = ptMod->T.Val.V_DATE;
  hedge_number = Test->Par[3].Val.V_LONG;
  step_hedge = (ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE) / (double)step

  Met->Par[0].Val.V_INT2 = step_number;
  Met->Par[1].Val.V_INT = 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 = 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++)
  {
    pnl_vect_resize(Test->Res[k].Val.V_PNLVECT, size);
  }
for (k = 15; k <= 20; k++)
  {
    pnl_vect_resize(Test->Res[k].Val.V_PNLVECT, size2);
  }
```

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


/******** 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;
    Met->Par[2].Val.V_DOUBLE = 0.; /*currentdelta*/
    /*delta=0.;*/
    Met->Par[0].Val.V_INT2 = step_number;
    hedge_number = Test->Par[3].Val.V_LONG;
    Met->Par[1].Val.V_INT = hedge_number;

    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[2].Val.V_DOUBLE;
    delta = Met->Res[0].Val.V_DOUBLE;
    Met->Par[2].Val.V_DOUBLE = delta;
    delta_array[0] = delta;
    Met->Par[0].Val.V_INT2--; /*stepnumber--*/

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

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

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

    /* computing the new stock's value */
    stock *= exp_trendxh * exp(sigmaxsqrth * pnl_rand_normal(type_generato

    /* 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;
      };
    hedgenow = Met->Res[3].Val.V_BOOL;
    if (hedgenow == 0)
      {
        delta = Met->Res[0].Val.V_DOUBLE;
        Met->Par[2].Val.V_DOUBLE = delta; /*currentdelta*/
        hedge_number--;
        Met->Par[1].Val.V_INT = hedge_number;
        hedge_time[indicehedge] = ptMod->T.Val.V_DATE;
        hedge_spot[indicehedge] = stock;
        indicehedge++;
      }

    delta_array[j] = delta;
    Met->Par[0].Val.V_INT2--; /*stepnumber--*/

    /* 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[2].Val.V_DOUBLE + delta * stock;

  } /*j*/

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

/* computing the new stock's value */
stock *= exp_trendxh * exp(sigmaxsqrth * pnl_rand_normal(type_generator));


delta_array[step_number] = delta;

/* Capitalization of cash_account and computing the P&L using the PayOff*/
cash_account = cash_account - ((ptOpt->PayOff.Val.V_NUMFUNC_1)->Compute)((
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_PNLVECT->array[k] = stock_array[k];
        Test->Res[7].Val.V_PNLVECT->array[k] = stock_array[k];
        Test->Res[8].Val.V_PNLVECT->array[k] = stock_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] = delta_array[k];
        Test->Res[13].Val.V_PNLVECT->array[k] = delta_array[k];
        Test->Res[14].Val.V_PNLVECT->array[k] = delta_array[k];
```

```
      }
    for (k = 0; k < size2; k++)
      {
        Test->Res[15].Val.V_PNLVECT->array[k] = hedge_time[k];
        Test->Res[16].Val.V_PNLVECT->array[k] = hedge_spot[k];
        Test->Res[17].Val.V_PNLVECT->array[k] = hedge_time[k];
        Test->Res[18].Val.V_PNLVECT->array[k] = hedge_spot[k];
        Test->Res[19].Val.V_PNLVECT->array[k] = hedge_time[k];
        Test->Res[20].Val.V_PNLVECT->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_PNLVECT->array[k] = stock_array[k];
            Test->Res[9].Val.V_PNLVECT->array[k] = pl_array[k];
            Test->Res[12].Val.V_PNLVECT->array[k] = delta_array[k];
          }
        for (k = 0; k < size2; k++)
          {
            Test->Res[15].Val.V_PNLVECT->array[k] = hedge_time[k];
            Test->Res[16].Val.V_PNLVECT->array[k] = hedge_spot[k];
          }
      }
    else if (pl_sample == max_pl)
      {
        for (k = 0; k <= step_number; k++)
          {
            Test->Res[7].Val.V_PNLVECT->array[k] = stock_array[k];
            Test->Res[10].Val.V_PNLVECT->array[k] = pl_array[k];
            Test->Res[13].Val.V_PNLVECT->array[k] = delta_array[k];
          }
        for (k = 0; k < size2; k++)
          {
            Test->Res[17].Val.V_PNLVECT->array[k] = hedge_time[k];
            Test->Res[18].Val.V_PNLVECT->array[k] = hedge_spot[k];
```

```
              }
            }
          else if (SQR(pl_sample - current_mean_pl) < SQR(median_pl - current_me
            {
              median_pl = pl_sample;
              for (k = 0; k <= step_number; k++)
                {
                  Test->Res[8].Val.V_PNLVECT->array[k] = stock_array[k];
                  Test->Res[11].Val.V_PNLVECT->array[k] = pl_array[k];
                  Test->Res[14].Val.V_PNLVECT->array[k] = delta_array[k];
                }
              for (k = 0; k < size2; k++)
                {
                  Test->Res[19].Val.V_PNLVECT->array[k] = hedge_time[k];
                  Test->Res[20].Val.V_PNLVECT->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;
  int i;
  if (first)
    {
      Test->Par[0].Val.V_INT = 0;         /* Random Generator */
      Test->Par[1].Val.V_LONG = 1;      /* PathNumber */
      Test->Par[2].Val.V_LONG = 100; /* StepNumber */
      Test->Par[3].Val.V_LONG = 10; /*hedgenumber*/
      Test->Par[4].Vtype = PREMIA_NULLTYPE;

      for (i = 5 ; i <= 20 ; i++)
        {
          Test->Res[i].Val.V_PNLVECT = pnl_vect_create(0);
        }
      Test->Res[21].Vtype = PREMIA_NULLTYPE;
      first = 0;
    }

  return OK;
}
int CHK_TEST(testpatry1)(void *Opt, void *Mod, PricingMethod *Met)
{
  if (strcmp(Met->Name, "TR_PatryMartini1") == 0)
    return OK;
  else
    return WRONG;
}

DynamicTest MOD_OPT(testpatry1) =
{
  "bs1d_std_testpatry1",

  { {"RandomGenerator", INT, {100}, ALLOW},
    {"PathNumber", LONG, {100}, ALLOW},
    {"StepNumber", LONG, {100}, ALLOW},
    {"HedgeNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
```

```
  CALC(DynamicHedgingSimulatorPatry1),

  { {"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", 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(testpatry1),
  CHK_ok,
  TEST(Init)
};
```