

[Help](#)

```
#include "
href../../../../mod/bs1d/bs1d_std/bs1d_std_h_src.pdfbs1d_std.h"

#define CALLOC_1D(P,N)  P=(double*)calloc(N+1,sizeof(double));\
    if (P==NULL)\
        return 1;\

#define CALLOC_2D(P,N)  P=(double**)calloc(N+1,sizeof(double*));\
    if (P==NULL)\
        return 1;\
    for (i=0;i<N+1;i++)\
    {\
        P[i]=(double*)calloc(N+1,sizeof(double));\
        if (P[i]==NULL)\
return 1;\
    }\

#define DESALLOC_2D(P,N)  for (i=0;i<N+1;i++)\
    free(P[i]);\
    free(P)\

#define DESALLOC_1D(P,N)  free(P)

static int CoxPatry2_98(double s, NumFunc_1 *p, double t, double r, double divid
                        double *ptprice, double *ptdelta, double *ptvariance, in
{
    int i, iStar, j, n, ii, jj;
    double u, d, pu, pd, a1, price, stock, lowerstock, h;
    double **Spot, **Price, **CurrentVStar, **PrevVStar, **Delta;
    double *SqrSpot, *SqrPrice, *SpotPrice, *CurrentV;
    double obstacle_value, current_value, price_minus_delta_spot, price_minus_alph

    /*Price, Variance arrays*/
    CALLOC_2D(Spot, N);
    CALLOC_2D(Price, N);
    CALLOC_2D(CurrentVStar, N);
```

```

CALLOC_2D(PrevVStar, N);
CALLOC_2D(Delta, N);

CALLOC_1D(CurrentV, N);
CALLOC_1D(SqrSpot, N);
CALLOC_1D(SqrPrice, N);
CALLOC_1D(SpotPrice, N);

/*Up and Down factors*/
h = t / (double)N;
a1 = exp(h * (r - divid));
u = exp(sigma * sqrt(h));
d = 1. / u;

/*Risk-Neutral Probability*/
pu = (a1 - d) / (u - d);
pd = 1. - pu;

/*FirstStep: Spot, Price, VStarZero (PrevVStar) computation*/
/*Price initialisation*/
lowerstock = s;
for (i = 0; i < N; i++)
    lowerstock *= d;

stock = lowerstock * exp(-r * t);

for (i = 0; i < (N + 1); i++)
{
    price = Price[N][i] = (p->Compute)(p->Par, stock * exp(r * t)) * exp(-r * t);

    Spot[N][i] = stock;
    SqrSpot[i] = stock * stock;
    SqrPrice[i] = price * price;
    SpotPrice[i] = stock * price;
    stock *= (u / d);
}

/*Backward Resolution*/
for (i = N - 1; i >= 0; i--)
    for (j = 0; j <= i; j++)
    {

```

```

        price = Price[i][j] = pu * Price[i + 1][j + 1] + pd * Price[i + 1][j];
        stock = Spot[i][j] = pu * Spot[i + 1][j + 1] + pd * Spot[i + 1][j];
        SqrSpot[j] = pu * SqrSpot[j + 1] + pd * SqrSpot[j];
        SqrPrice[j] = pu * SqrPrice[j + 1] + pd * SqrPrice[j];
        SpotPrice[j] = pu * SpotPrice[j + 1] + pd * SpotPrice[j];
        Delta[i][j] = (Price[i + 1][j + 1] - Price[i + 1][j]) * exp(r * h) / (st

        PrevVStar[i][j] = SqrPrice[j] - price * price + \
                        (Delta[i][j]) * (Delta[i][j]) * (SqrSpot[j] - stock *
    }

    iStar = 1;

    /*SecondStep: Vstar_n computation*/

    if (N_Hedge == 0)
    {
        iStar = 1;
        CurrentVStar[0][0] = PrevVStar[0][0];
    }
    else
    {
        if (N == N_Hedge)
        {
            iStar = 0;
            CurrentVStar[0][0] = 0.;
        }
        else
        {
            for (n = 1; n <= N_Hedge - 1; n++)
            {
                for (i = N - n - 1; i >= 0; i--)
                for (j = 0; j <= i; j++)
                {
                    /*CurrentV Initialisation*/
                    for (jj = 0; jj <= N - n; jj++)
                    {
                        price_minus_delta_spot = Price[N - n][jj] - Delta[i][j]
                        CurrentV[jj] = price_minus_delta_spot * price_minus_delt
                    }
                }
            }
        }
    }

```

```

/*We start the computation at time N-n-1*/
for (ii = N - n - 1; ii >= i; ii--)
    for (jj = j; jj <= ii - (i - j); jj++)
    {
        CurrentV[jj] = pu * CurrentV[jj + 1] + pd * CurrentV[j]
        price_minus_delta_spot = Price[ii][jj] - Delta[i][j] *
        obstacle_value = price_minus_delta_spot * price_minus_

        if (CurrentV[jj] > obstacle_value)
            CurrentV[jj] = obstacle_value;
    }

    price_minus_delta_spot = Price[i][j] - Delta[i][j] * Spot[i]
    current_value = CurrentV[j] - price_minus_delta_spot * price
    CurrentVStar[i][j] = current_value;
}/*End j*/

for (i = N - n - 1; i >= 0; i--)
    for (j = 0; j <= i; j++)
        PrevVStar[i][j] = CurrentVStar[i][j];
}/*End n*/

/*Last Hedge*/
for (ii = 0; ii <= N - N_Hedge; ii++)
{
    price_minus_alpha_spot = Price[N - N_Hedge][ii] - alphacourant * S
    CurrentV[ii] = price_minus_alpha_spot * price_minus_alpha_spot;
}

for (i = N - N_Hedge - 1; i >= 0; i--)
    for (j = 0; j <= i; j++)
    {
        CurrentV[j] = pu * CurrentV[j + 1] + pd * CurrentV[j];
        price_minus_alpha_spot = Price[i][j] - alphacourant * Spot[i][j]
        obstacle_value = price_minus_alpha_spot * price_minus_alpha_spot

        if (CurrentV[j] > obstacle_value)
        {
            if (i == 0)
                iStar = 0;
            CurrentV[j] = obstacle_value;

```

```

        }
    }
    current_value = CurrentV[0] - price_minus_alpha_spot * price_minus_alpha_star;
    CurrentVStar[0][0] = current_value;

}

}

*ptprice = Price[0][0];
*ptdelta = Delta[0][0];
*ptvariance = CurrentVStar[0][0];
*pthedge = !(iStar == 0); /*pthedge=0 means it's optimal to hedge*/

DESALLOC_1D(SqrSpot, N);
DESALLOC_1D(SqrPrice, N);
DESALLOC_1D(SpotPrice, N);
DESALLOC_1D(CurrentV, N);

DESALLOC_2D(Spot, N);
DESALLOC_2D(Price, N);
DESALLOC_2D(CurrentVStar, N);
DESALLOC_2D(PrevVStar, N);
DESALLOC_2D(Delta, N);

return 0;
}

int CALC(TR_Patry1)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    return CoxPatry2_98(ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt->Maturity.Val.V_DATE,
        r, divid, ptMod->Sigma.Val.V_PDOUBLE,
        Met->Par[0].Val.V_INT2, Met->Par[1].Val.V_INT,
        &(Met->Res[2].Val.V_DOUBLE), &(Met->Res[0].Val.V_DOUBLE),

```

```

        &(Met->Res[1].Val.V_DOUBLE), &(Met->Res[3].Val.V_BOOL),
        Met->Par[2].Val.V_DOUBLE);
    }

static int CHK_OPT(TR_Patry1)(void *Opt, void *Mod)
{
    Option *ptOpt = (Option *)Opt;
    TYPEOPT *opt = (TYPEOPT *) (ptOpt->TypeOpt);

    if ((opt->EuOrAm). Val.V_BOOL == EURO)
        return OK;

    return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;

        Met->Par[0].Val.V_INT2 = 100; /*stepnumber*/
        Met->Par[1].Val.V_INT = 10; /*hedgenumber*/
        Met->Par[2].Val.V_DOUBLE = 0.; /*currentdelta*/

        Met->Res[0].Val.V_DOUBLE = 0.; /*optimaldelta*/
        Met->Res[1].Val.V_DOUBLE = 0.; /*variance*/
        Met->Res[2].Val.V_DOUBLE = 0.; /*optimalprice*/
        Met->Res[3].Val.V_BOOL = 0; /*hedgenow*/

    }

    return OK;
}

PricingMethod MET(TR_Patry1) =
{
    "TR_Patry1",
    {
        {"StepNumber", INT2, {100}, ALLOW},
    }
}

```

```

    {"HedgeNumber", INT, {10}, ALLOW},
    {"CurrentDelta", DOUBLE, {10}, IRRELEVANT},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(TR_Patry1),
{
    {"OptimalDelta", DOUBLE, {100}, FORBID} ,
    {"Variance", DOUBLE, {100}, FORBID},
    {"OptimalPrice", DOUBLE, {100}, FORBID} ,
    {"HedgeNow", BOOL, {0}, FORBID} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(TR_Patry1),
CHK_tree,
MET(Init)
};

```