

[Help](#)

```
#include "bs1d_std.h"

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

#define CALLOC_2D(P,N)  P=calloc(N+1,sizeof(
    double*));{
    if (P==NULL){
    return 1;{
    for (i=0;i<N+1;i++){
        {{
        P[i]=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 sigma,int N,int
    N_Hedge,
        double *ptprice,double *ptdelt
    a,double *ptvariance,int *pthedge,double alphacou
    rant)
{
    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 alphaStar,obstacle_value,current_value,
    price_minus_delta_spot,price_minus_alpha_spot;

/*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*SqrPri
        ce[j];
        SpotPrice[j]=pu*SpotPrice[j+1]+ pd*Spot
        Price[j];
        Delta[i][j]=(Price[i+1][j+1]-Price[i+1][
        j])*exp(r*h)/(stock*(u-d));

        PrevVStar[i][j]=SqrPrice[j]-price*price+
        {
            (Delta[i][j])*(Delta[i][j])*(SqrSpot[
            j]-stock*stock)-2*Delta[i][j]*(SpotPrice[j]-pric
            e*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;

```

```

        alphaStar=(Price[1][1]-Price[1][0])*
exp(r*h)/s/(u-d);
        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]*Spot[N-n][jj];
                    CurrentV[jj]=price_minus_delta_spot*price_minus_delta_spot;
                }

                /*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[jj];
                        price_minus_delta_spot=Price[ii][jj]-Delta[i][j]*Spot[ii][jj];
                        obstacle_value=price_minus_delta_spot*price_minus_delta_spot+PrevVStar[ii][jj];

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

                    price_minus_delta_spot=Price[i][j]-Delta[i][j]*Spot[i][j];

```

```

        current_value=CurrentV[j]-price_minus_delta_spot*price_minus_delta_spot;
        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*Spot[N-N_Hedge][ii];
    CurrentV[ii]=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+PrevVStar[i][j];

        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_spot;
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,Pricing
    Method *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-ptMod->T.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->Re
        s[0].Val.V_DOUBLE),

```

```

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

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)
{
    static int first=1;

    if (first)
    {
        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*/

        first=0;
    }

    return OK;
}

```

```
PricingMethod MET(TR_Patry1)=
{
  "TR_PatryMartini1",
  {
    {"StepNumber",INT2,100,ALLOW},
    {"HedgeNumber",INT, 10,ALLOW},
    {"CurrentDelta",DOUBLE, 10,IRRELEVANT},

    {" ",END,0,FORBID}},
  CALC(TR_Patry1),
  {
    {"OptimalDelta",DOUBLE,100,FORBID} ,
    {"Variance",DOUBLE,100,FORBID},
    {"OptimalPrice",DOUBLE,100,FORBID} ,
    {"HedgeNow",BOOL,0,FORBID} ,
    {" ",END,0,FORBID}},
  CHK_OPT(TR_Patry1),
  CHK_tree,
  MET(Init)
};
```

References