```
   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 CoxPatry_98(double s,NumFunc_1 *p,
    double t,double r,double divid,double sigma,int N,int
    N_Hedge,int N_Sample,double alpha_min,double alph
    a_max,double *ptprice,double *ptdelta,double *pt
    variance,int *pthedge,double alphacourant)
  {
  int i,iStar,j,k,n;
  double u,d,pu,pd,a1,price,stock,lowerstock,h;
  double **Spot,**Price,**CurrentVStar,**PrevVS
    tar;
  double *SqrSpot,*SqrPrice,*SpotPrice,*CurrentV
    ;
  double alpha,alpha_step,alphaStar,obstacle_val
    ue,current_value,price_minus_alpha_spot,alphaSt
    ar0;
```

```
/*Price, Variance arrays*/
CALLOC_2D(Spot,N);
CALLOC_2D(Price,N);
CALLOC_2D(CurrentVStar,N);
CALLOC_2D(PrevVStar,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 (PrevVSta
  r) computation*/
/*Price initialisation*/
lowerstock=s;
for (i=0;i<N;i++)
   lowerstock*=d;

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

for (i=0;i<=N;i++)
  {
   price=Price[N][i]=(p->Compute)(p->Par,sto
  ck*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];

     PrevVStar[i][j]=SqrPrice[j]-price*pric
  e-{
        (SpotPrice[j]-price*stock)*(SpotPr
  ice[j]-price*stock)/(SqrSpot[j]-stock*stock);
     }

alphaStar0=(SpotPrice[0]-price*stock)/(SqrSpo
  t[0]-stock*stock);

iStar=1;

alphaStar=alphaStar0;

/*SecondStep: Vstar_n computation*/
alpha_step=(alpha_max-alpha_min)/(double)N_Sa
  mple;

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
    {
    if (PrevVStar[0][0]==0.)
      {iStar=0;CurrentVStar[0][0]=0.;}
    else
      {
      for (n=1;n<=N_Hedge;n++)
        {
        alpha=alpha_min;
        if (n<N_Hedge)
          alphaStar=alpha_min;
        for (k=0;k<=N_Sample;k++)
           {
           /*CurrentV Initialisation*/
          for (j=0;j<=N-n;j++)
             {
             price_minus_alpha_spot=Price[N-
    n][j]-alpha*Spot[N-n][j];
             CurrentV[j]=price_minus_alpha_
    spot*price_minus_alpha_spot;
             }

          /*We start the computation at
    time N-n-1*/
          for (i=N-n-1;i>=0;i--)
            for (j=0;j<=i;j++)
              {
              CurrentV[j]=pu*CurrentV[j+1]
    +pd*CurrentV[j];
              price_minus_alpha_spot=Pric
    e[i][j]-alpha*Spot[i][j];
              obstacle_value=price_minus_
    alpha_spot*price_minus_alpha_spot+PrevVStar[i][j]
    ;

              if (CurrentV[j]>obstacle_val
```

```
ue)
              {
              if ((n==N_Hedge) && (alph
a==alphacourant) && (i==0))
               iStar=0;
              CurrentV[j]=obstacle_valu
e;
              }

            /*Compute the new current mi
nimum of V_n(alpha)*/
            current_value=CurrentV[j]-pr
ice_minus_alpha_spot*price_minus_alpha_spot;
            if (k>0)
              {
              if (CurrentVStar[i][j]>cu
rrent_value)
             {
                if ((n==(N_Hedge-1)) &
& (i==0))
                  alphaStar=alpha;
                CurrentVStar[i][j]=cu
rrent_value;
             }
              }
           else CurrentVStar[i][j]=curr
ent_value;
           } /*End j*/
       alpha+=alpha_step;
       }/*End k*/

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

  }
}
}
```

```
  if (PrevVStar[0][0]<1.0e-7)
    alphaStar=alphaStar0;


  *ptprice=Price[0][0];
  *ptdelta=alphaStar;
  *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);

  return 0;
  }


int CALC(TR_Patry)(void *Opt,void *Mod,Pricing
    Method *Met)
  {
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid,alphamin,alphamax;

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

   alphamin=-1.;
   alphamax=1.;

  return CoxPatry_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->Par[3].Val.V_INT2,
         alphamin,alphamax,
         &(Met->Res[2].Val.V_DOUBLE),&(Met->Re
    s[0].Val.V_DOUBLE),
         &(Met->Res[1].Val.V_DOUBLE),&(Met->Re
    s[3].Val.V_BOOL),
       Met->Par[2].Val.V_DOUBLE);
  }

int CHK_OPT(TR_Patry)(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;    /*hedgenumb
    er*/
    Met->Par[2].Val.V_DOUBLE=0.;    /*curren
    tdelta*/
        Met->Par[3].Val.V_INT2=100;  /*hedgesam
    pling*/

    Met->Res[0].Val.V_DOUBLE=0.;  /*optimalde
    lta*/
```

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

    first=0;
    }

  return OK;
  }

PricingMethod MET(TR_Patry)=
  {
  "TR_PatryMartini",
  {
    {"StepNumber",INT2,100,ALLOW},
    {"HedgeNumber",INT, 10,ALLOW},
    {"CurrentDelta",DOUBLE, 10,IRRELEVANT},
  {"HedgeSampling",INT2,100,ALLOW},

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

References