Help

```
#include  "bs1d_std.h"

static int CoxRossRubinstein_79(int am,double s,
    NumFunc_1  *p,double t,double r,double divid,double sig
    ma,int N, double *ptprice,double *ptdelta)
    {
       int i,j;
       double u,d,h,pu,pd,a1,stock,upperstock;
       double *P,*iv;


       /*Price, intrisic value arrays*/
       P=(double *)malloc((N+1)*sizeof(double));
       if (P==NULL)
           return MEMORY_ALLOCATION_FAILURE;
       iv=(double *)malloc((2*N+1)*sizeof(double)
    );
       if (iv==NULL)
           return MEMORY_ALLOCATION_FAILURE;

       /*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;

        if ((pd>=1.) || (pd<=0.))
           return NEGATIVE_PROBABILITY;

       pu*=exp(-r*h);
       pd*=exp(-r*h);
       /*Intrisic value initialisation*/
       upperstock=s;
       for (i=0;i<N;i++)
           upperstock*=u;
       stock=upperstock;
```

```
        for (i=0;i<2*N+1;i++)
            {
            iv[i]=(p->Compute)(p->Par,stock);
            stock*=d;
            }

        /*Terminal Values*/
        for (j=0;j<=N;j++)
            P[j]=iv[2*j];

        /*Backward Resolution*/
        for (i=1;i<=N-1;i++)
            for (j=0;j<=N-i;j++)
                {
                P[j]=pu*P[j]+pd*P[j+1];
                if (am)
                    P[j]=MAX(iv[i+2*j],P[j]);
                }

            /*Delta*/
            *ptdelta=(P[0]-P[1])/(s*u-s*d);

            /*First time step*/
            P[0]=pu*P[0]+pd*P[1];
            if (am)
                P[0]=MAX(iv[N],P[0]);

            /*Price*/
            *ptprice=P[0];

            free(P);
            free(iv);

            return OK;
    }

int CALC(TR_CoxRossRubinstein)(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 CoxRossRubinstein_79(ptOpt->EuOrAm.Val
    .V_BOOL,ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Matu
    rity.Val.V_DATE-ptMod->T.Val.V_DATE,
        r,divid,ptMod->Sigma.Val.V_PDOUBLE,Met->
    Par[0].Val.V_INT,&(Met->Res[0].Val.V_DOUBLE),&(Met-
    >Res[1].Val.V_DOUBLE));
    }

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

    return  OK;
    }

static int MET(Init)(PricingMethod *Met)
    {
    static int first=1;

    if (first)
        {
        Met->Par[0].Val.V_INT2=100;

        first=0;
        }

    return OK;
    }

PricingMethod MET(TR_CoxRossRubinstein)=
    {
    "TR_CoxRossRubinstein",
        {{"StepNumber",INT2,100,ALLOW},{" ",END,0
```

```
,FORBID}},
    CALC(TR_CoxRossRubinstein),
    {{"Price",DOUBLE,100,FORBID},{"Delta",
DOUBLE,100,FORBID} ,{" ",END,0,FORBID}},
    CHK_OPT(TR_CoxRossRubinstein),
    CHK_tree,
    MET(Init)
};
```

References