```
   Help
#include <stdlib.h>
#include  "bs1d_lim.h"
#include "error_msg.h"

static int Ritchken_95_UpOut(int am,double s,NumFunc_1  *p,
    double rebate,double l,double t,double r,double divid,double si
    gma,int N,double *ptprice,double *ptdelta)
{
  int i,j,npoints,eta0;
  double h,pu,pm,pd,z,u,d,stock,upperstock,eta,lambda;
  double *P,*iv;

  /*Price, intrisic value arrays*/
  npoints=2*N+1;
  P= malloc(npoints*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  iv= malloc(npoints*sizeof(double));
  if (iv==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  /*Up and Down factors*/
  h=t/(double) N;
  eta=log(l/s)/(sigma*sqrt(h));
  eta0=(int) floor(eta);
  lambda=eta/(double)eta0;
  if(eta0>N) {
    eta0=N;
    lambda=1.22474;
  }
  u=exp(lambda*sigma*sqrt(h));
  d=1./u;
  /*Disconunted Probability*/
  z=(r-divid)-SQR(sigma)/2.;
  pu=(1./(2.*SQR(lambda))+z*sqrt(h)/(2.*lambda*sigma));
  pm=(1.-1./SQR(lambda));
  pd=(1.-pu-pm);
  pu*=exp(-r*h);
  pm*=exp(-r*h);
  pd*=exp(-r*h);
```

```
/*Intrisic value initialisation and terminal values*/
upperstock=s;
for (i=0;i<N;i++)
  upperstock*=d;

stock=upperstock;
for(i=0;i<N+eta0;i++) {
  iv[i]=(p->Compute)(p->Par,stock);
  P[i]=iv[i];
  stock*=u;
}

npoints=N+eta0;
P[npoints]=rebate;

/*Backward Resolution*/
for (i=1;i<=N-eta0;i++)
  {
    npoints-=1;
    for (j=0;j<npoints;j++)
{
  P[j]=pd*P[j]+pm*P[j+1]+pu*P[j+2];
  if (am)
    P[j]=MAX(iv[j+i],P[j]);
}
    P[npoints]=rebate;
  }
npoints++;
for (i=N-eta0+1;i<N;i++)
  {
    npoints-=2;
    for (j=0;j<npoints;j++)
{
  P[j]=pd*P[j]+pm*P[j+1]+pu*P[j+2];
  if (am)
    P[j] = MAX(iv[j+i],P[j]);
}
  }
/*Delta*/

*ptdelta=(P[2]-P[0])/(s*u-s*d);
```

```
  /*First time step*/
  P[0]=pd*P[0]+pm*P[1]+pu*P[2];
  if (am)
    P[0]=MAX(iv[N],P[0]);
  /*Price*/
  *ptprice=P[0];

  free(P);
  free(iv);

  return OK;
}

int CALC(TR_Ritchken_UpOut)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=( TYPEOPT*)Opt;
  TYPEMOD* ptMod=( TYPEMOD*)Mod;
  double r,divid,limit,rebate;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
  divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
  limit=((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt->    Limit.Val.V_NUMFUN
  rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt-
    >Rebate.Val.V_NUMFUNC_1)->Par,ptMod->T.Val.V_DATE);

  return Ritchken_95_UpOut(ptOpt->EuOrAm.Val.V_BOOL,ptMod->
    S0.Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,
          rebate,
          limit,ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,r,divid,
          ptMod->Sigma.Val.V_PDOUBLE,
          Met->Par[0].Val.V_INT2,
          &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.
    V_DOUBLE));
}

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

  if ((opt->OutOrIn).Val.V_BOOL==OUT)
    if ((opt->DownOrUp).Val.V_BOOL==UP)
      if ((opt->Parisian).Val.V_BOOL==WRONG)
  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;


    }

  return OK;
}

PricingMethod MET(TR_Ritchken_UpOut)=
{
  "TR_Ritchken_UpOut",
  {{"StepNumber",INT2,{100},ALLOW},{" ",PREMIA_NULLTYPE,{0}
    ,FORBID}},
  CALC(TR_Ritchken_UpOut),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_Ritchken_UpOut),
  CHK_tree,
  MET(Init)
};
```

References