

[Help](#)

```
#include "bs1d_lim.h"

static int Ritchken_95_UpIn(int am,double s,
    NumFunc_1*p,double rebate,double l,double t,double r,
    double divid,double sigma,int N,double *ptprice,
    double *ptdelta)
{
    int i,j,npoints,eta0,dummy;
    double h,pu,pm,pd,z,u,d,stock,upperstock,eta,
    lambda,price,delta;
    double *P,*iv;

    /*Price, intrinsic value arrays*/
    npoints=2*N+1;
    P=(double *)malloc(npoints*sizeof(double));
    if (P==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    iv=(double *)malloc(npoints*sizeof(double));
    if (iv==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    /*Up and Down factors*/
    h=t/(double) N;
    eta=log(1/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);
}
```

```

/*Intrinsic 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]=rebate;
    stock*=u;
}

npoints=N+eta0;
if ((p->Compute)==&Call)
    dummy=Call_BlackScholes_73(1,p->Par[0].
Val.V_PDOUBLE,0.,r,divid,sigma,&price,&delta);
else
    dummy=Put_BlackScholes_73(1,p->Par[0].
Val.V_PDOUBLE,0.,r,divid,sigma,&price,&delta);
P[npoints]=price;

/*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]);
    }
    if ((p->Compute)==&Call)
        dummy=Call_BlackScholes_73(1,p->
Par[0].Val.V_PDOUBLE,(double)i*h,r,divid,sigma,&pr
ice,&delta);
    else
        dummy=Put_BlackScholes_73(1,p->
Par[0].Val.V_PDOUBLE,(double)i*h,r,divid,sigma,&pr
ice,&delta);
}

```

```

        P[npoints]=price;
    }
    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\_UpIn)(void *Opt,void *Mod,Pr
icingMethod *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)->Compu

```

```

te)((ptOpt->Limit.Val.V_NUMFUNC_1)->Par,ptMod->T.
Val.V_DATE);
rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->
Compute)((ptOpt->Rebate.Val.V_NUMFUNC_1)->Par,ptMod-
>T.Val.V_DATE);

return Ritchken_95_UpIn(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));
}

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

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

    return WRONG;
}

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\_Ritchken\_UpIn)=
{
    "TR_RitchkenUpIn",
    {"StepNumber",INT2,100,ALLOW},{" ",END,0
,FORBID}},
    CALC(TR\_Ritchken\_UpIn),
    {"Price",DOUBLE,100,FORBID},{"Delta",
DOUBLE,100,FORBID} ,{" ",END,0,FORBID}},
    CHK_OPT(TR\_Ritchken\_UpIn),
    CHK_tree,
    MET(Init)
};
```

References