```
   Help
#include <stdlib.h>
#include  "bs1d_doublim.h"
#include "error_msg.h"

static int Ritchken_95_In(int am,double s,NumFunc_1*L,
    NumFunc_1*U,NumFunc_1*Rebate,NumFunc_1*p,double t,double r,
    double divid,double sigma,int N,double lambda,double *ptprice,
    double *ptdelta)
{

  int i,j,npoints,eta0,A0;
  double h,puu,pum,pud,pdu,pdd,pdm,rebate,z,up,down,stock,
    lowerstock,eta,a,b,gamma,A,sd,lsh,u,d,price;
  double *P,*G,*iv;

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


  /*Up and Down factors*/
  up=(U->Compute)(U->Par,0);
  down=(L->Compute)(L->Par,0);
  rebate=(Rebate->Compute)(Rebate->Par,0);

  h=t/(double)N;

  eta=log(up/s)/(sigma*sqrt(h));
  eta0=(int)floor(eta);

  /*The Up barrier is too close to S0-the algorithm fails*/
  if (eta0<2)
```

```
  return STEP_NUMBER_TOO_SMALL;

/*Adjustment of lambda to set a level of the tree at the
  barrier*/
/*In case the step number is not sufficient, then take th
  e usual parameter*/
if(eta0>N)
  {
    eta0=N;
    /*In this case lambda keeps the value given in para
  meter*/
  }
else
  lambda=eta/(double)eta0;

lsh=lambda*sigma*sqrt(h);
A=log(s/down)/lsh;
A0=(int)floor(A)-1;

/*The Down barrier is too close to S0-the algorithm fails
  */
if (A0<0)
  return STEP_NUMBER_TOO_SMALL;

if (A0>(N-1))
  {
    A0=N-1;
    gamma=1.;
  }
else
  {
    sd=s*exp(-A0*lsh);
    gamma=log(sd/down)/lsh;
  }

npoints=eta0+A0;

/*Up and Down factors*/
u=exp(lsh);
d=1./u;
```

```
/*Disconunted Probability*/
z=(r-divid)-SQR(sigma)/2.;
puu=(1./(2.*SQR(lambda))+z*sqrt(h)/(2.*lambda*sigma));
pum=(1.-1./SQR(lambda));
pud=(1.-puu-pum);
puu*=exp(-r*h);
pum*=exp(-r*h);
pud*=exp(-r*h);

a=z*sqrt(h)/(lambda*sigma);
b=1.0/SQR(lambda);
pdu=(b+a*gamma)/(1.+gamma);
pdd=(b-a)/(gamma*(1.+gamma));
pdm=(1.-pdu-pdd);
pdu*=exp(-r*h);
pdm*=exp(-r*h);
pdd*=exp(-r*h);

/*Intrinsic value initialization and terminal values*/
lowerstock=s;
for (i=0;i<A0;i++)
  {
    lowerstock*=d;
  }

stock=lowerstock;

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

if (eta0<N)
  {
    price=(p->Compute)(p->Par,up);
    P[npoints]=price;
    G[npoints]=price;
```

```
  }
else
  {
    P[npoints]=rebate;
    G[npoints]=rebate;
  }

/*Backward Resolution*/
if (A0<eta0) /*The Down barrier is hit first*/
  {
    for (i=1;i<=N-eta0;i++)
/*Both barriers are active*/
{
  price=Boundary(down,p,(double)i*h,r,divid,sigma);
  P[0]=pdd*price+pdm*G[0]+pdu*G[1];
  if (am)
    P[0] = MAX(iv[0],P[0]);
  for (j=1;j<npoints;j++)
    {
      P[j]=pud*G[j-1]+pum*G[j]+puu*G[j+1];
      if (am)
  P[j] = MAX(iv[j],P[j]);
    }
  price=Boundary(up,p,(double)i*h,r,divid,sigma);
  P[npoints]=price;
  for (j=0;j<=npoints;j++)
    G[j]=P[j];
}
    for (i=N-eta0+1;i<=N-A0;i++)
/*Only the Down barrier is active*/
{
  npoints-=1;
  price=Boundary(down,p,(double)i*h,r,divid,sigma);
  P[0]=pdd*price+pdm*G[0]+pdu*G[1];
  if (am)
    P[0] = MAX(iv[0],P[0]);
  for (j=1;j<=npoints;j++)
    {
      P[j]=pud*G[j-1]+pum*G[j]+puu*G[j+1];
      if (am)
  P[j] = MAX(iv[j],P[j]);
```

```
    }
  for (j=0;j<=npoints;j++)
    G[j]=P[j];
}
  }/*endif*/
else
  if (A0>eta0) /*The Down barrier is hit first*/
    {
for (i=1;i<=N-A0;i++)
  /*Both barriers are active*/
  {
    price=Boundary(down,p,(double)i*h,r,divid,sigma);
    P[0]=pdd*price+pdm*G[0]+pdu*G[1];
    if (am)
      P[0] = MAX(iv[0],P[0]);
    for (j=1;j<npoints;j++)
      {
  P[j]=pud*G[j-1]+pum*G[j]+puu*G[j+1];
  if (am)
    P[j] = MAX(iv[j],P[j]);
      }
    price=Boundary(up,p,(double)i*h,r,divid,sigma);
    P[npoints]=price;
    for (j=0;j<=npoints;j++)
      G[j]=P[j];
  }
for (i=N-A0+1;i<=N-eta0;i++)
  /*Only the Up barrier is active*/
  {
    npoints-=1;
    for (j=0;j<npoints;j++)
      {
  P[j]=pud*P[j]+pum*P[j+1]+puu*P[j+2];
  if (am)
    P[j] = MAX(iv[j+i],P[j]);

      }
    price=Boundary(up,p,(double)i*h,r,divid,sigma);
    P[npoints]=price;
  }
    }/*endelse*/
```

```
  else if (A0==eta0)
    {
for (i=1;i<=N-A0;i++)
  {
    price=Boundary(down,p,(double)i*h,r,divid,sigma);
    P[0]=pdd*price+pdm*G[0]+pdu*G[1];
    if (am)
      P[0] = MAX(iv[0],P[0]);
    for (j=1;j<npoints;j++)
      {
  P[j]=pud*G[j-1]+pum*G[j]+puu*G[j+1];
  if (am)
    P[j] = MAX(iv[j],P[j]);
      }
    price=Boundary(up,p,(double)i*h,r,divid,sigma);
    P[npoints]=price;
    for (j=0;j<=npoints;j++)
      G[j]=P[j];
  }
    }/*endelse*/

/*None of the barriers is active*/
if (A0>eta0)
  A0=eta0;
npoints++;
for (i=N-A0+1;i<N;i++)
  {
    npoints-=2;
    for (j=0;j<npoints;j++)
{
  P[j]=pud*P[j]+pum*P[j+1]+puu*P[j+2];
  if (am)
    P[j] = MAX(iv[j+i-(N-A0+1)],P[j]);

}
  }

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

/*First time step*/
```

```
  P[0]=pud*P[0]+pum*P[1]+puu*P[2];
  if (am)
    P[0]=MAX(iv[A0],P[0]);


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

  free(P);
  free(G);
  free(iv);

  return OK;
}


int CALC(TR_Ritchken_In)(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 Ritchken_95_In(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.
    Val.V_PDOUBLE,
      ptOpt->LowerLimit.Val.V_NUMFUNC_1,ptOpt->UpperLimi
    t.Val.V_NUMFUNC_1,ptOpt->Rebate.Val.V_NUMFUNC_1,ptOpt->PayO
    ff.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_RGDOUBLE,
    &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}


static int CHK_OPT(TR_Ritchken_In)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
```

```
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->Parisian).Val.V_BOOL==WRONG)
    if ((opt->OutOrIn).Val.V_BOOL==IN)
      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;
      Met->Par[1].Val.V_RGDOUBLE12=1.22474;
    }
  return OK;
}

PricingMethod MET(TR_Ritchken_In)=
{
  "TR_Ritchken_In",
  {{"StepNumber",INT2,{100},ALLOW},
   {"Lambda",RGDOUBLE12,{1},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_Ritchken_In),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_Ritchken_In),
  CHK_tree,
  MET(Init)
};
```

References