```
   Help
#include  "bs1d_lim.h"

static int RogersStapleton_UpOut_97(int am,
    double S,NumFunc_1  *p,double T,double up, double reba
    te,double r,double divid,double sigma,double step_
    space, double *ptprice,double *ptdelta)
{

  double *P;
  double pu,pd;
  int Eta0,npoints,i,j,m,n,npts;
  double Eta,pulim,pdlim,G,Prix;
  double mu,c,B1,B2,B3,C1,C2,y,alpha3;
  double stock,lower;
  double moy,v,u,d,x1,x2,Q,Delta;
  double U1,U2,W1,W2,pr,pro1,pro2,disc;


  /*Up and Down probabilities*/
  u=step_space;
  d=-u;
  mu=(r-divid)-((sigma*sigma)/2.);
  c=mu/(sigma*sigma);
  pu=(exp(2.*c*u)-1.)/(exp(2.*c*u)-exp(-2.*c*u));
  pd=1.-pu;


  Eta=(log(up/S))/u;
  Eta0=(int) floor(Eta);
  x1=log(S)+(Eta0*u);
  x2=log(up);

  if (Eta0==Eta)
  pdlim=0.;
  else
  pdlim=(exp(-2.*c*x1)-exp(-2.*c*x2))/(exp(-2.*
    c*(x1-u))-exp(-2.*c*x2));

 pulim=1.-pdlim;
```

```
/*moments of tau1*/

moy=(u/mu)*tanh(c*u);
v=((sigma/mu)*(sigma/mu)*moy)-((u/mu)*(u/mu))+(
  moy*moy);
v=sqrt(v);

B1=12.*c*u*(-exp(-4.*c*u)-exp(-2.*c*u));
B2=8.*c*c*u*u*(-exp(-2.*c*u)+exp(-4.*c*u));
B3=3.*(1-exp(-2.*c*u)+exp(-4.*c*u)-exp(-6.*c*u)
  );
C1=u*(B1+B2-B3);
y=(-exp(-2.*c*u)-1.);
C2=pow(c,5.)*pow(sigma,6.)*pow(y,3.);
alpha3=C1/C2;


/*Initialization*/
U2=(T-moy)/v;
W2= ((1.-U2*U2)*exp(-U2*U2/2.))/sqrt(72.*PI);
Q=0.0;
Prix=0.;
Delta=0.;
n=1;

/*recursion on the number of time-steps*/

do{/*computation of the probability of nu=n*/

U1=U2;
W1=W2;
U2=(T-(double)(n+1)*moy)/(v*sqrt((double)(n+1)
  ));
W2= ((1-U2*U2)*exp(-U2*U2/2))/sqrt(72*PI*(
  double)(n+1));

pro1=N(U1);
pro2=N(U2);


pr=(pro1-pro2)+alpha3*(W1-W2);
```

```
if (pr<0.000005)
  {
  Q+=pr;
  n++;
  }

else
  {/*contribution for a fixed number of time-
  steps*/

  Q=Q+pr;
  disc=exp(-r*T/(double)n);

  if (n >= Eta0) /*Barrier hit*/
    {

    lower=S*exp((double)n*d);
    stock=lower;

    m=(int) floor((n-Eta0)/2);
    npoints=Eta0+m;
    npts=n-Eta0;

    if(Eta0==0) npts=n-1;

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

    for (i=0;i<=npoints;i++)
      {
      P[i]=(p->Compute)(p->Par,stock);
      stock=stock*exp(2.*u);
      }


    /*Terminal Values*/
    if((n-Eta0)%2==0)
```

```
        {
        npoints--;
        for (i=1;i<=npts;i++)
          {
          if(i%2==0)
            {
            for (j=0;j<npoints;j++)
              P[j]=disc*(pd*P[j]+pu*P[j+1])
    ;

            P[npoints]=disc*(pulim*rebate+
    pdlim*P[npoints]);
            npoints--;
            }
          else
            {
            for (j=0;j<=npoints;j++)
              P[j]=disc*(pd*P[j]+pu*P[j+1])
    ;

            }
          }
        }

      else
        {
        for (i=1;i<=npts;i++)
          {
          if(i%2==0)
            {
            for (j=0;j<=npoints;j++)
              P[j]=disc*(pd*P[j]+pu*P[j+1])
    ;
            }
          else
            {
            for (j=0;j<npoints;j++)
              P[j]=disc*(pd*P[j]+pu*P[j+1])
    ;

            P[npoints]=disc*(pulim*rebate+
```

```
pdlim*P[npoints]);
        npoints--;
        }
      }
    }

  for (i=1;i<Eta0;i++)
    {
    for (j=0;j<=Eta0-i;j++)
      P[j]=disc*(pd*P[j]+pu*P[j+1]);
    }

  /*Price*/
  if (Eta0==0)
    {
    G=disc*(pulim*rebate+pdlim*P[0]);
    Delta=Delta+(G-P[0])*pr/(S*(exp(u)-1)
);
    P[0]=disc*(pulim*rebate+pdlim*P[0]);
    }
  else
    {
    Delta=Delta+(P[1]-P[0])*pr/(S*(exp(u)
-exp(d)));
    P[0]=disc*(pu*P[0]+pd*P[1]);
    }

  P[0]=P[0]*pr;
  }

else     /*Barrier not hit*/
  {

  /*Terminal Values*/
  lower=S*exp((double)n*d);

  stock=lower;
  /*Price, intrinsic value arrays*/
  P=(double *)malloc((npoints+1)*sizeof(
double));
  if (P==NULL)
```

```
        return MEMORY_ALLOCATION_FAILURE;
      for (i=0;i<=n;i++)
        {
        P[i]=(p->Compute)(p->Par,stock);
        stock=stock*exp(2.*u);
        }

      /*Backward Resolution*/

      for (i=1;i<n;i++)
        {
        for (j=0;j<=n-i;j++)
          P[j]=disc*(pd*P[j]+pu*P[j+1]);
        }

      /*Price*/
      Delta=Delta+(P[1]-P[0])*pr/(S*(exp(u)-
    exp(d)));
      P[0]=disc*(pd*P[0]+pu*P[1]);
      P[0]=P[0]*pr;

      }
    Prix=Prix+P[0];

    /*Memory Desallocation*/
    free(P);

    n++;
    }


  }
  /*end of the recursion*/
  while (Q<0.99999);

  /*Price and Delta*/

  *ptprice=Prix;
  *ptdelta=Delta;

  return OK;
```

```
}


int CALC(TR_RogersStapleton_UpOut)(void *Opt,void
     *Mod,PricingMethod *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 RogersStapleton_UpOut_97(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,limit,rebate,
        r,divid,ptMod->Sigma.Val.V_PDOUBLE,Met->
    Par[0].Val.V_DOUBLE,
        &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1]
    .Val.V_DOUBLE));
}

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

  if ((opt->EuOrAm).Val.V_BOOL==EURO)
   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)
{
    static int first=1;

    if (first)
    {
        Met->Par[0].Val.V_DOUBLE=0.01;

        first=0;
    }

    return OK;
}

PricingMethod MET(TR_RogersStapleton_UpOut)=
{
    "TR_RogersStapleton",
    {{"Space Step",DOUBLE,100,ALLOW},{" ",END,0,
    FORBID}},
    CALC(TR_RogersStapleton_UpOut),
    {{"Price",DOUBLE,100,FORBID},{"Delta",DOUBLE,
    100,FORBID} ,{" ",END,0,FORBID}},
    CHK_OPT(TR_RogersStapleton_UpOut),
    CHK_tree,
    MET(Init)
};
```

# References