

[Help](#)

```

#include "bs1d_lim.h"

static int RogersStapleton_DownOut_97(int am,
    double S,NumFunc_1 *p,double T,double down, double re
    bate,double r,double divid,double sigma,double
    step_space, double *ptprice,double *ptdelta)
{

    double *P;
    double pu,pd;
    int A0,npoints,i,j,m,n,npts;
    double A,pulim,pdlim,G,Prix;
    double mu,c,B1,B2,B3,C1,C2,y,alpha3;
    double stock,lower,upper;
    double moy,v,u,d,x1,x2,Q,Delta;
    double U1,U2,W1,W2,pr,pro1,pro2,disc;

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

    /*Intrinsic value initialisation*/
    A=log(S/down)/u;
    A0=(int) floor(A);
    x1=log(S)+A0*d;
    x2=log(down);

    if (A0==A)
        pulim=0.;
    else
        pulim=(exp(-2.*c*x2)-exp(-2.*c*x1))/(exp(-2.*
            c*x2)-exp(-2.*c*(x1+u)));

```

```

pdlim=1.-pulim;

/*Calcul de l'esperence et la variance de tau1*
/
moy=(u/mu)*tanh(c*u);
/* v=((sigma/mu)*(sigma/mu)*moy)-((u/mu)*(u/mu)
)+(moy*moy);*/
v=SQR(moy)+SQR(sigma/mu)*moy-SQR(u/mu);
v=sqrt(v);

/*Calcul de alpha3*/
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;

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

/*Construction de l'arbre*/

do{

U1=U2;
W1=W2;

U2=(T-(double)(n+1)*moy)/(v*sqrt((double)(n+1)
));
W2=((1.-SQR(U2))*exp(-SQR(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
{
    /*printf("%e\n",Q);
    printf("%d\n",n);*/
    Q+=pr;
    disc=exp(-r*T/(double)n);

    if (n >= A0) /*on touche la Barrier*/
    {

        upper=S*exp((double)n*u);
        stock=upper;

        m=(int) floor((n-A0)/2);
        npoints=A0+m;
        npts=n-A0;

        if(A0==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.*d);
        }
```

```

/*Terminal Values*/

/*Terminal Values*/
if((n-A0)%2==0)
{
  npoints--;
  for (i=1;i<=npts;i++)
  {
    if(i%2==0)
    {
      for (j=0;j<npoints;j++)
        P[j]=disc*(pu*P[j]+pd*P[j+1])
    }
    ;

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

  }
}

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

    }
  else
  {
    for (j=0;j<npoints;j++)

```

```

        P[j]=disc*(pu*P[j]+pd*P[j+1])
    ;

    P[npoints]=disc*(pdlim*rebate+
pulim*P[npoints]);
    npoints--;
    }
    }
}

for (i=1;i<A0;i++)
{
    for (j=0;j<=A0-i;j++)
        P[j]=disc*(pu*P[j]+pd*P[j+1]);
}

/*Price*/
if (A0==0)
{
    G=disc*(pdlim*rebate+pulim*P[0]);
    Delta=Delta+(P[0]-G)*pr/(S*(exp(u)-1)
);
    P[0]=disc*(pdlim*rebate+pulim*P[0]);
}
else
{
    Delta=Delta+(P[0]-P[1])*pr/(S*(exp(u)
-exp(d)));
    P[0]=disc*(pu*P[0]+pd*P[1]);
}

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

else /*Si on ne touche pas la Barrier*/
{
    /*Terminal Values*/
    lower=S*exp((double)n*d);

    stock=lower;
    /*Price, intrinsic value arrays*/

```

```

    P=(double *)malloc((n+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++;
    }
}
while (Q<0.99999);

/*Price and Delta*/

*ptprice=Prix;
*ptdelta=Delta;

return OK;

```

```
}

```

```
int CALC(TR_RogersStapleton_DownOut)(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)->Compute)((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_DownOut_97(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Maturity.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_DownOut)(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==DOWN)
                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=0.02;

        first=0;
    }

    return OK;
}

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

```

References