Help
```
#include <stdlib.h>
#include  "bs1d_pad.h"
#include "error_msg.h"

static int  Babbs_95_FloatingPut(int am,double s,double
         s_max,NumFunc_2*p,double t,double r,double div
    id,double sigma,int
         N,double *ptprice,double *ptdelta)
{
  int i,j,eta0,npoints;
  double u,d,h,pu,pd,a1,stock,eta,y_0;
  double *P,*iv,*Q,*Boundary;
  double upperstock,old_price=0.;
  int odd;
  double flat_price=0., up_price;

  /*Price, intrisic value arrays*/
  P= malloc((2*N+1)*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Q= malloc((2*N+1)*sizeof(double));
  if (Q==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  iv= malloc((2*N+1)*sizeof(double));
  if (iv==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  Boundary= malloc((N+1)*sizeof(double));
  if (Boundary==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  /*Up and Down factors*/
  h=t/(double)N;
  a1=exp(h*(r-divid));
  u=exp(sigma*sqrt(h));
  d=1./u;

  /*Critical Index*/
  y_0=s_max/s;
  eta=log(y_0)/(sigma*sqrt(h));
  eta0=(int)floor(eta);
```

```
if (eta0>N) eta0=N;

/*Risk-Neutral Probability*/
pu=(a1-d)/(u-d);
pd=1.-pu;
pu*=exp(-r*h)*u;/*Proba Downward here*/
pd*=exp(-r*h)*d;

if(eta0<N)
  {
    /*First Stage:Computation of price value along the
  line
spot=maximum*/

    /*Intrisic value initialisation*/
    for (i=0;i<=N;i++)
Boundary[i]=0.;

    stock=1.;
    for (i=0;i<=N+eta0;i++)
{
  iv[i]=(p->Compute)(p->Par,1.,stock);
  P[i]=iv[i];
  Q[i]=P[i];
  stock*=u;
}

    Boundary[N]=P[0];

    /*Backward Resolution*/
    for (i=1;i<N-eta0;i++)
{
  old_price=P[0];/*Used for the delta in the case s=s_
  max*/
  /*Node on the line stock=1*/
  P[0]=pu*Q[0]+pd*Q[1];
  if (am)
          P[0]=MAX(iv[0],P[0]);
  Boundary[N-i]=P[0];
  /*Nodes above*/
  for (j=1;j<=N+eta0-i;j++)
```

```
        {
      /*Forget about the Q[j] price*/
      P[j]=pu*Q[j-1]+pd*Q[j+1];
      if (am)
            P[j]=MAX(iv[j],P[j]);
        }

  for (j=0;j<=N+eta0-i;j++)
        Q[j]=P[j];
      }
  }

if (s_max>s)
  {
    /*Second Stage:Computation of price value */
    /*Intrinsic value initialization*/
    upperstock=y_0;
    for (i=0;i<N;i++)
upperstock*=u;
    stock=upperstock;
    for (i=0;i<=N+eta0;i++)
{
  iv[i]=(p->Compute)(p->Par,1.,stock);
  stock*=d;
}

    /*Terminal Values*/
    npoints=eta0+(N-eta0)/2;

    for (j=0;j<=npoints;j++)
P[j]=iv[2*j];/*indexed from above*/

    /*Backward Resolution*/
    if (eta0>0) /*The first mesh does not breach the bar
  rier*/
{
  /*First part-the barrier is active*/
  odd=1;
  for (i=eta0;i<N-1;i++)
    odd=!odd;
```

```
if (!odd) npoints=npoints-1;

for (i=1;i<=N-eta0;i++)
  {
    for (j=0;j<npoints;j++)
{
  P[j]=pd*P[j]+ pu*P[j+1];
  if (am)
    P[j]=MAX(iv[i+2*j],P[j]);
}
    /*Special handling of the critical node*/
    if (odd)
{
  P[npoints]=pd*P[npoints]+pu*Boundary[N+1-i];
  if (am)
    P[npoints]=MAX(iv[i+2*npoints],P[npoints]);
}

    /*For the critical node at the next iteration*/
    if (!odd)
{
  npoints=npoints-1;
}
    odd=!odd;
  }

/*Second part-the barrier is strictly below the tree*/
npoints=eta0-1;
for (i=N-eta0+1;i<N;i++)
  {
    for (j=0;j<=npoints;j++)
{
  P[j]=pd*P[j]+ pu*P[j+1];
  if (am)
    P[j]=MAX(iv[i+2*j],P[j]);
}

    npoints=npoints-1;
  }
/*Delta*/
*ptdelta=(P[1]-P[0])/(y_0*(d-u));
```

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

      else /*eta0=0, the first mesh breaches the barrier*/
  {
    /*The barrier is always active*/
    odd=1;
    for (i=eta0;i<N-1;i++)
      odd=!odd;

    if (!odd) npoints=npoints-1;

    for (i=1;i<=N-eta0-1;i++) /*We go backward until the
    next date*/
      {
        flat_price=P[1]; /*Only for the delta*/

        for (j=0;j<npoints;j++)
    {
      P[j]=pd*P[j]+ pu*P[j+1];
      if (am)
        P[j]=MAX(iv[i+2*j],P[j]);
    }
        /*Special handling of the critical node*/
        if (odd)
    {
      P[npoints]=pd*P[npoints]+pu*Boundary[N+1-i];
      if (am)
        P[npoints]=MAX(iv[i+2*npoints],P[npoints]);;
    }


        if (!odd)
    {
      npoints=npoints-1;
```

```
  }
      odd=!odd;

    }

  up_price=P[0];   /*For the delta*/

  /*First time step*/
  P[0]=pd*P[0]+pu*P[1];

  /*Special handling of the critical node*/
  P[0]=pd*P[0]+pu*Boundary[1];
  if (am)
    P[0]=MAX(iv[N],P[0]);

  /*Delta*/
  /*Corresponds to setting a third point at level s bet
  ween u*s and d*s*/
  /*One computes the finite difference approximation bet
  ween s and us*/
  *ptdelta=(up_price-(exp(-r*h)*flat_price+exp(r*h)*P[0]
  )*.5)/(y_0*(u-1.));
} /*eta0=0*/
  }

else /*s=s_max*/
  *ptdelta=(P[1]-old_price)/(u-1.);

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

/*Delta*/

*ptdelta=(*ptdelta)*(-y_0)+P[0];

/*Memory Desallocation*/
free(P);
free(Q);
free(iv);
free(Boundary);
```

```
  return OK;

}

int CALC(TR_Babbs_Put)(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  Babbs_95_FloatingPut(ptOpt->EuOrAm.Val.V_BOOL,pt
    Mod->S0.Val.V_PDOUBLE,(ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4
    ].Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_2,ptOpt->Matu
    rity.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_Babbs_Put)(void *Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"    LookBackPutFloatingEuro")==0) || (strcmp
    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=1000;

    }

  return OK;
}
```

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

References