Help

```
#include  "bs1d_lim.h"

static int DermanKani_92(int am,double s,NumFunc_
    1  *p,double t,double l, double rebate,double r,
    double divid,double sigma,int N, double *ptprice,
    double *ptdelta)
{
       int i,j,eta0,npoints;
       double u,d,h,pu,pd,a1,stock,upperstock,et
    a,critical_node,pl,one_minus_pl;
       double *P,*iv;
       int odd;
       double flat_price, up_price;

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

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

       /*Number of down moves just before breachi
    ng the barrier*/
       eta=log(s/l)/(sigma*sqrt(h));
       eta0=(int)floor(eta);
```

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

   /*Weights for the linear interpolation at
the critical node*/
   /*Node above the barrier*/
   critical_node=s*exp(-(double)eta0*sigma*sq
rt(h));
   pl=(l-d*critical_node)/(critical_node-d*cr
itical_node);
   one_minus_pl=1.-pl;

   /*Intrinsic value initialization*/
   upperstock=s;
   for (i=0;i<N;i++)
       upperstock*=u;
   stock=upperstock;
   for (i=0;i<=N+eta0;i++)
   {
       iv[i]=(p->Compute)(p->Par,stock);
       stock*=d;
   }

   /*Terminal Values*/
   npoints=eta0+(N-eta0)/2;
   for (j=0;j<=npoints;j++)
       P[j]=iv[2*j];
   /*For the critical node at the next itera
tion*/
   P[npoints+1]=rebate;

   /*Backward Resolution*/
   if (eta0>0) /*The first mesh does not brea
ch the barrier*/
   {

   /*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]=pu*P[j]+ pd*P[j+1];
           if (am)
               P[j]=MAX(iv[i+2*j],P[j]);
       }
       /*Special handling of the critical no
de*/
       if (odd)
       {
           P[npoints]=pl*rebate+one_minus_pl*
P[npoints];
       }
       /*For the critical node at the next it
eration*/
       if (odd)
       {
        npoints=npoints-1;
       }
       else
         P[npoints+1]=rebate;

       odd=!odd;
   }

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

```
        npoints=npoints-1;
    }


    /*Delta*/
    *ptdelta=(P[0]-P[1])/(s*u-s*d);


    /*First time step*/
    P[0]=pu*P[0]+pd*P[1];
    if (am)
        P[0]=MAX(iv[N],P[0]);
    /*Price*/
    *ptprice=P[0];
    }
    else /*eta0=0, the first mesh breaches th
e 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 backwar
d until the next date*/
    {

    flat_price=P[1]; /*Only for the delta*/

        for (j=0;j<=npoints;j++)
        {
            P[j]=pu*P[j]+ pd*P[j+1];
            if (am)
                P[j]=MAX(iv[i+2*j],P[j]);
        }
        /*Special handling of the critical no
de*/
        if (odd)
        {
```

```
            P[npoints]=pl*rebate+one_minus_pl*
P[npoints];
       }


       if (odd)
       {
        npoints=npoints-1;
       }
       else
         P[npoints+1]=rebate;/*For the critic
al node at the next iteration*/

       odd=!odd;

   }


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

   /*First time step*/
   P[0]=pu*P[0]+pd*P[1];
   /*Special handling of the critical node*/
   P[0]=pl*rebate+one_minus_pl*P[0];
   if (am)
       P[0]=MAX(iv[N],P[0]);

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

  /*Delta*/
   /*Corresponds to setting a third point at
level s between u*s and d*s*/
   /*One computes the finite difference appro
ximation between s and us*/
    *ptdelta=(up_price-(exp(-r*h)*flat_price+
exp(r*h)*P[0])*.5)/(u*s-s);

   }
```

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

        return OK;
}


int CALC(TR_DermanKani)(void *Opt,void *Mod,Pric
    ingMethod *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 DermanKani_92(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_INT,
        &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1]
    .Val.V_DOUBLE));
}

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


   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_INT2=100;

        first=0;
    }

    return OK;
}

PricingMethod MET(TR_DermanKani)=
{
    "TR_DermanKani",
    {{"StepNumber",INT2,100,ALLOW},{" ",END,0,FO
    RBID}},
    CALC(TR_DermanKani),
    {{"Price",DOUBLE,100,FORBID},{"Delta",DOUBLE,
    100,FORBID} ,{" ",END,0,FORBID}},
    CHK_OPT(TR_DermanKani),
    CHK_tree,
    MET(Init)
};
```

References