```
   Help
#include <stdlib.h>
#include  "bs1d_limdisc.h"

static int fd_call_down_out(int am,double t,NumFunc_1  *p,
    double r,double divid,double sigma,double limit,int nb_mon_date,
    double x,int N,double *pt_price,double *pt_delta)
{

  double K,theta,s;
  int i,TimeIndex,j,M;
  double sigma2;
  double x_min,x_max;
  double   *alpha_l,*beta_l,*gamma_l,*alpha_r,*beta_r,*gam
    ma_r_,*vect_t;
  double   *vect_s,*V,*Vp,*beta_p,*Price,*Obst,*monit_date,
    *Old_Price;
  double   a,b,c,a1,b1,c1;
  double hi,hip,xis,xips,xims,boundary_inf,boundary_sup;
  double INC_DELTA=0.0001;
  double price_l,price_r,rebate,barrier_down;
  int down_index;

  K=p->Par[0].Val.V_PDOUBLE;
  theta=0.5;
  s=x/K;
  sigma2=SQR(sigma);
  rebate=0./K;
  barrier_down=limit/K;

  /*Time Step number*/
  M=20*nb_mon_date;

  /*Memory Allocation*/
  alpha_l= malloc((N+1)*sizeof(double));
  beta_l= malloc((N+1)*sizeof(double));
  gamma_l= malloc((N+1)*sizeof(double));
  alpha_r= malloc((N+1)*sizeof(double));
  beta_r= malloc((N+1)*sizeof(double));
  gamma_r_= malloc((N+1)*sizeof(double));
  vect_t= malloc((M+1)*sizeof(double));
```

```
monit_date= malloc((M+1)*sizeof(double));
vect_s= malloc((N+1)*sizeof(double));
V= malloc((N+1)*sizeof(double));
Vp= malloc((N+1)*sizeof(double));
beta_p= malloc((N+1)*sizeof(double));
Price= malloc((N+1)*sizeof(double));
Old_Price= malloc((N+1)*sizeof(double));
Obst= malloc((N+2)*sizeof(double));

/*Space Localisation*/
x_min=0.;
x_max=2.*s;

/*Time Discretisation*/
for(i=0;i<=M;i++)
  vect_t[i]=((double)i)*(t)/(double)M;

/*Monitoring Dates*/
for(i=1;i<=nb_mon_date;i++)
  monit_date[i]=((double)i)*(t)/(double)nb_mon_date;

/*Mesh Points*/
for(i=0;i<=N;i++)
  {
    vect_s[i]=x_min+((double)i)*(x_max-x_min)/(double)N;
  }

/*Compute barrier level*/
i=0;
while (vect_s[i]<barrier_down) i++;
down_index=i;

/*Terminal Values*/
for(i=0;i<=N;i++)
  {
    if(i<=down_index)
Price[i]=rebate;
    else
Price[i]=MAX(0.,vect_s[i]-1.);

    Obst[i]=Price[i];
    }

  /*Finite Difference Cycle*/
  for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
    {
      for(j=1;j<nb_mon_date;j++)

  if (vect_t[TimeIndex]==monit_date[j])
    {
      if((am==1)&&(rebate==0.))
        for(i=0;i<=down_index;i++)
    Price[i]=MAX(0,vect_s[i]-1);
      else
        for(i=0;i<=down_index;i++)
    Price[i]=rebate;
    }

      boundary_inf=rebate;
      boundary_sup=x_max*exp(-divid*(vect_t[TimeIndex]))-
    exp(-r*(vect_t[TimeIndex]));

      a=(1.+r*(vect_t[TimeIndex]-vect_t[TimeIndex-1])*thet
    a)/2.;
      b=theta*(vect_t[TimeIndex]-vect_t[TimeIndex-1])/4.;
      c=theta*(vect_t[TimeIndex]-vect_t[TimeIndex-1])/2.;

      a1=(1.-r*(vect_t[TimeIndex]-vect_t[TimeIndex-1])*(1.-
    theta))/2.;
      b1=(1.-theta)*(vect_t[TimeIndex]-vect_t[TimeIndex-1])
    /4.;
      c1=(1.-theta)*(vect_t[TimeIndex]-vect_t[TimeIndex-1])
    /2.;

      for(i=1;i<N;i++)
  {
    hi=vect_s[i]-vect_s[i-1];
    hip=vect_s[i+1]-vect_s[i];

    xis=vect_s[i]*vect_s[i];
    xips=vect_s[i+1]*vect_s[i+1];
    xims=vect_s[i-1]*vect_s[i-1];
    /*Computation of Lhs coefficients*/
    alpha_l[i]=-b*sigma2*(xis+xims)/hi-c*(sigma2*vect_s[i]
    -(r-divid)*vect_s[i]);
    beta_l[i]=a*(hi+hip)+
      b*(sigma2*(xis+xims)/hi+sigma2*(xips+xis)/hip);
    gamma_l[i]=-b*sigma2*(xips+xis)/hip+c*(sigma2*vect_s[
    i]-(r-divid)*vect_s[i]);

    /*Computation of Rhs coefficients*/
    alpha_r[i]=b1*sigma2*(xis+xims)/hi+c1*(sigma2*vect_s[
    i]-(r-divid)*vect_s[i]);
    beta_r[i]=a1*(hi+hip)-
      b1*(sigma2*(xis+xims)/hi+sigma2*(xips+xis)/hip);
    gamma_r_[i]=b1*sigma2*(xips+xis)/hip-c1*(sigma2*vect_
    s[i]-(r-divid)*vect_s[i]);
  }
      /*Compute Rhs*/
      V[1]=alpha_r[1]*Price[0]+beta_r[1]*Price[1]+gamma_r_[
    1]*Price[2]-alpha_l[1]*boundary_inf;
      for (i=2;i<N-1;i++)
  V[i]=alpha_r[i]*Price[i-1]+beta_r[i]*Price[i]+gamma_r_[
    i]*Price[i+1];

      V[N-1]=alpha_r[N-1]*Price[N-2]+beta_r[N-1]*Price[N-1]
    +gamma_r_[N-1]*Price[N]-gamma_l[N-1]*boundary_sup;

      Price[0]=boundary_inf;
      Price[N]=boundary_sup;

      /*Gauss pivoting*/
      Vp[N-1]=V[N-1];
      beta_p[N-1]=beta_l[N-1];

      for(i=N-2;i>=1;i--)
  {
    beta_p[i]=beta_l[i]-gamma_l[i]*alpha_l[i+1]/beta_p[i+1
    ];
    Vp[i]=V[i]-gamma_l[i]*Vp[i+1]/beta_p[i+1];
  }
```

```
    Price[1]=Vp[1]/beta_p[1];

    for (i=2;i<=N-1;i++)
{
  Price[i]=(Vp[i]-alpha_l[i]*Price[i-1])/beta_p[i];
}

    if(am)
for(i=1;i<=N-1;i++)
  {
    Price[i]=MAX(Price[i],Obst[i]);
  }

  }
/*End of Time Cycle*/

/*Price*/
i=0;
while (vect_s[i]<s) i++;

i=0;
while (vect_s[i]<s*(1.+INC_DELTA)) i++;
price_r=(Price[i]+(Price[i]-Price[i-1])*(s*(1.+INC_DELTA)
  -vect_s[i])/(vect_s[i]-vect_s[i-1]));

i=0;
while (vect_s[i]<s*(1.-INC_DELTA)) i++;
price_l=(Price[i]+(Price[i]-Price[i-1])*(s*(1.-INC_DELTA)
  -vect_s[i])/(vect_s[i]-vect_s[i-1]));

/*Price*/
*pt_price=K*(Price[i]+(Price[i]-Price[i-1])*(s-vect_s[i])
  /(vect_s[i]-vect_s[i-1]));

/*Delta*/
*pt_delta=(price_r-price_l)/(2.*s*INC_DELTA);

/*Memory Desallocation*/
free(alpha_r);
free(beta_r);
free(gamma_r_);
```

```
  free(alpha_l);
  free(beta_l);
  free(gamma_l);
  free(vect_t);
  free(monit_date);
  free(vect_s);
  free(V);
  free(Vp);
  free(beta_p);
  free(Price);
  free(Old_Price);
  free(Obst);

  return OK;
}

int CALC(FD_LimDisc)(void*Opt,void *Mod,PricingMethod *Met)
{
  TYPEOPT* ptOpt=( TYPEOPT*)Opt;
  TYPEMOD* ptMod=( TYPEMOD*)Mod;
  double r,divid,limit,sd;
  int return_value;

  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_NUMFUN
  sd=(ptOpt->Limit.Val.V_NUMFUNC_1)->Par[0].Val.V_DATE;

  if(sd!=ptMod->T.Val.V_DATE)
    {
      Fprintf(TOSCREEN," StartingDate=!t0, untreated case{
    n{n{n");
      return_value = WRONG;
    }
  else
    return_value=fd_call_down_out(ptOpt->EuOrAm.Val.V_BOOL,
    ptOpt->Maturity.Val.V_DATE-sd,ptOpt->PayOff.Val.V_NUMFUNC_1
    ,r,divid,ptMod->Sigma.Val.V_PDOUBLE,limit,(ptOpt->Limit.
    Val.V_NUMFUNC_1)->Par[2].Val.V_INT2,ptMod->S0.Val.V_PDOUBLE,
     Met->Par[0].Val.V_INT,&(Met->Res[0].Val.V_DOUBLE),&(Met->
    Res[1].Val.V_DOUBLE));
  return return_value;

}

static int CHK_OPT(FD_LimDisc)(void *Opt, void *Mod)
{
  return strcmp( ((Option*)Opt)->Name,"CallDownOutDiscEuro"
    );
}

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(FD_LimDisc)=
{
  "FD_LimDisc",
  {{"SpaceStepNumber",INT2,{100},ALLOW},{" ",PREMIA_NULLTYP
    E,{0},FORBID}},
  CALC(FD_LimDisc),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_LimDisc),
  CHK_ok,
  MET(Init)
} ;
```

References