Help
```
#include <stdlib.h>
#include  "bs1d_lim.h"
#include "error_msg.h"
#define PRECISION 1.0e-7 /*Precision for the localization
    of FD methods*/

static int Gauss_UpOut(int am,double s,NumFunc_1  *p,
    double l,double rebate,double t,double r,double divid,double si
    gma,int N,int M,double theta,double *ptprice,double *ptdelt
    a)
{
  int      Index,PriceIndex,TimeIndex;
  double   k,vv,loc,h,z,alpha,beta,gamma,y,alpha1,beta1,gam
    ma1,up;
  double   *Obst,*A,*B,*C,*P,*S,pricenh,pricep2h,priceph;

  /*Memory Allocation*/
  Obst= malloc((N+2)*sizeof(double));
  if (Obst==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  A= malloc((N+2)*sizeof(double));
  if (A==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  B= malloc((N+2)*sizeof(double));
  if (B==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  C= malloc((N+2)*sizeof(double));
  if (C==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  P= malloc((N+2)*sizeof(double));
  if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
  S= malloc((N+2)*sizeof(double));
  if (S==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  /*Time Step*/
  k=t/(double)M;

  /*Space Localisation*/
```

```
vv=sigma*sigma;
z=(r-divid)-vv/2.0;
loc=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fabs(z)*t;

/*Space Step*/
y=log(s);
up=log(l);
h=(up-(y-loc))/(double)(N+1);

/*Lhs Factor of theta-schema*/
alpha=theta*k*(-vv/(2.0*h*h)+z/(2.0*h));
beta=1.0+k*theta*(r+vv/(h*h));
gamma=k*theta*(-vv/(2.0*h*h)-z/(2.0*h));
for(PriceIndex=1;PriceIndex<=N;PriceIndex++)
  {
    A[PriceIndex]=alpha;
    B[PriceIndex]=beta;
    C[PriceIndex]=gamma;
  }
/*Rhs Factor of theta-schema*/
alpha1=k*(1.0-theta)*(vv/(2.0*h*h)-z/(2.0*h));
beta1=1.0-k*(1.0-theta)*(r+vv/(h*h));
gamma1=k*(1.0-theta)*(vv/(2.0*h*h)+z/(2.0*h));

/*Set Gauss*/
for(PriceIndex=N-1;PriceIndex>=1;PriceIndex--)
  B[PriceIndex]=B[PriceIndex]-C[PriceIndex]*A[PriceIndex+
  1]/B[PriceIndex+1];
for(PriceIndex=1;PriceIndex<=N;PriceIndex++)
  A[PriceIndex]=A[PriceIndex]/B[PriceIndex];
for(PriceIndex=1;PriceIndex<N;PriceIndex++)
  C[PriceIndex]=C[PriceIndex]/B[PriceIndex+1];

/*Tenminal Values*/
for(PriceIndex=0;PriceIndex<=N;PriceIndex++) {
  Obst[PriceIndex]=(p->Compute)(p->Par,exp(y-loc+(double)
  PriceIndex*h));
  P[PriceIndex]= Obst[PriceIndex];
}
P[N+1]=rebate;
```

```
/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
  {
    /*Set Rhs*/
    S[1]=beta1*P[1]+gamma1*P[2]+alpha1*P[0]-alpha*P[0];
    for(PriceIndex=2;PriceIndex<=N-1;PriceIndex++)
S[PriceIndex]= alpha1*P[PriceIndex-1]+beta1*P[PriceInd
  ex]+gamma1*P[PriceIndex+1];
    S[N]=beta1*P[N]+alpha1*P[N-1]+gamma1*P[N+1]-gamma*P[
  N+1];

    for(PriceIndex=N-1;PriceIndex>=1;PriceIndex--)
S[PriceIndex]=S[PriceIndex]-C[PriceIndex]*S[PriceIndex+1
  ];

    /*Solve the system*/
    P[1] =S[1]/B[1];

    for(PriceIndex=2;PriceIndex<=N;PriceIndex++)
P[PriceIndex]=S[PriceIndex]/B[PriceIndex]-A[PriceIndex]*
  P[PriceIndex-1];

    /*Splitting for the american case*/
    if (am)
for(PriceIndex=1;PriceIndex<=N;PriceIndex++)
  P[PriceIndex]=MAX(Obst[PriceIndex],P[PriceIndex]);
  }

Index=(int)floor(loc/h);

/*Price*/
*ptprice=P[Index]+(P[Index+1]-P[Index])*(exp(y)-exp(y-
  loc+Index*h))/(exp(y-loc+(Index+1)*h)-exp(y-loc+Index*h));

/*Delta*/
priceph=P[Index-1]+(P[Index]-P[Index-1])*(exp(y-h)-exp(y-
  loc+(Index-1)*h))/(exp(y-loc+(Index)*h)-exp(y-loc+(Index-1)*
  h));
if (y!=up) {
  pricenh=P[Index+1]+(P[Index+2]-P[Index+1])*(exp(y+h)-
  exp(y-loc+(Index+1)*h))/(exp(y-loc+(Index+2)*h)-exp(y-loc+(
```

```
    Index+1)*h));
    *ptdelta=(pricenh-priceph)/(2*s*h);
  } else {
    pricep2h=P[Index-2]+(P[Index-3]-P[Index-2])*(exp(y-2*h)
    -exp(y-loc+(Index-2)*h))/(exp(y-loc+(Index-3)*h)-exp(y-
    loc+(Index-2)*h));
    *ptdelta=(-4*priceph+pricep2h+3*(*ptprice))/(2*s*h);
  }

  /*Memory Desallocation*/
  free(Obst);
  free(A);
  free(B);
  free(C);
  free(P);
  free(S);

  return OK;
}

int CALC(FD_Gauss_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)->Compute)((ptOpt->     Limit.Val.V_NUMFUN
  rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt-
    >Rebate.Val.V_NUMFUNC_1)->Par,ptMod->T.Val.V_DATE);

  return Gauss_UpOut(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val
    .V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,
         limit,rebate,ptOpt->Maturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,
         r,divid,ptMod->Sigma.Val.V_PDOUBLE,
         Met->Par[0].Val.V_INT2,Met->Par[1].Val.V_INT2,
    Met->Par[2].Val.V_RGDOUBLE051,
         &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_
```

```
    DOUBLE));
}

static int CHK_OPT(FD_Gauss_UpOut)(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==UP)
      if ((opt->Parisian).Val.V_BOOL==WRONG)
  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=100;
      Met->Par[1].Val.V_INT2=100;
      Met->Par[2].Val.V_RGDOUBLE=0.5;


    }

  return OK;
}

PricingMethod MET(FD_Gauss_UpOut)=
{
  "FD_Gauss_UpOut",
  {{"SpaceStepNumber",INT2,{100},ALLOW  },{"TimeStepNumber"
    ,INT2,{100},ALLOW},
   {"Theta",RGDOUBLE051,{100},ALLOW},{" ",PREMIA_NULLTYPE,{
    0},FORBID}},
  CALC(FD_Gauss_UpOut),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
```

```
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(FD_Gauss_UpOut),
  CHK_split,
  MET(Init)
};
```

References