

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 Psor_DownOut(double s, NumFunc_1 *p, double l,
double rebate, double t, double r, double divid, double sigma, int N,
int M, double theta, double omega, double epsilon, double *pt
price, double *ptdelta)
{
int      Index, PriceIndex, TimeIndex;
int      j, loops;
double   k, vv, loc, h, z, alpha, beta, gamma, y, alpha1, beta1, gam
ma1, down, upwind_alphacoef;
double   error, norm, x, pricenh, pricen2h, priceph;
double   *P, *Obst, *Rhs;

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

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

/*Space Localisation*/
vv=0.5*SQR(sigma);
z=(r-divid)-vv;
loc=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fabs(z)*t;

/*Space Step*/
x=log(s);
down=log(l);

```

```

h=(x+loc-down)/(double)(N+1);

/*Coefficient of diffusion augmented*/
if ((h*fabs(z))<=vv)
    upwind_alphacoef=0.5;
else {
    if (z>0.) upwind_alphacoef=0.0;
    else upwind_alphacoef=1.0;
}
vv-=z*h*(upwind_alphacoef-0.5);

/*Lhs factor of theta-schema*/
alpha=theta*k*(-vv/(h*h)+z/(2.0*h));
beta=1.0+k*theta*(r+2.*vv/(h*h));
gamma=k*theta*(-vv/(h*h)-z/(2.0*h));

/*Rhs factor of theta-schema*/
alpha1=k*(1.0-theta)*(vv/(h*h)-z/(2.0*h));
beta1=1.0-k*(1.0-theta)*(r+2.*vv/(h*h));
gamma1=k*(1.0-theta)*(vv/(h*h)+z/(2.0*h));

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

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
{
    /*Init Rhs*/
    for(j=1;j<=N;j++)
        Rhs[j]=P[j]*beta1+alpha1*P[j-1]+gamma1*P[j+1];

    /*Psor Cycle*/
    loops=0;
    do
    {

```

```

error=0.;
norm=0.;

for(j=1;j<=N;j++)
{
    y=(Rhs[j]-alpha*P[j-1]-gamma*P[j+1])/beta;
    y=MAX(Obst[j],P[j]+omega*(y-P[j]));

    error+=(double)(j+1)*fabs(y-P[j]);
    norm+=fabs(y);
    P[j]=y;
}

if (norm<1.0) norm=1.0;
error=error/norm;

loops++;
}
while ((error>epsilon) && (loops<MAXLOOPS));
}
Index=(int)floor((x-down)/h);

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

/*Delta*/
pricen=P[Index+1]+(P[Index+2]-P[Index+1])*(exp(x+h)-exp(
down+(Index+1)*h))/(exp(down+(Index+2)*h)-exp(down+(Index+1)
*h));
if (Index>0) {
    pricep=P[Index-1]+(P[Index]-P[Index-1])*(exp(x-h)-exp(
down+(Index-1)*h))/(exp(down+(Index)*h)-exp(down+(Index-1)*
h));
    *ptdelta=(pricen-pricep)/(2*s*h);
} else {
    pricen2=P[Index+2]+(P[Index+3]-P[Index+2])*(exp(x+2*h)
-exp(down+(Index+2)*h))/(exp(down+(Index+3)*h)-exp(down+(
Index+2)*h));
    *ptdelta=(4*pricen-pricen2-3*(ptprice))/(2.*s*h);
}

```

```

/*Memory Desallocation*/
free(P);
free(Obst);
free(Rhs);

return OK;
}

int CALC(FD_Psor_DownOut)(void *Opt,void *Mod,Pricing
    Method *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 Psor_DownOut(ptMod->S0.Val.V_PDOUBLE,ptOpt->PayO
        ff.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->Par[3].Val.V_
            RGDOUBLE12,Met->Par[4].Val.V_RGDOUBLE,
                &(Met->Res[0].Val.V_DOUBLE),&(Met->
            Res[1].Val.V_DOUBLE));
}

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

    if ((opt->Parisian).Val.V_BOOL==WRONG)
        if ( (strcmp( ((Option*)Opt)->Name,"CallDownOutAmer")==
            0) || (strcmp( ((Option*)Opt)->Name,"PutDownOutAmer")==0)

```

```

    )
    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;
        Met->Par[3].Val.V_RGDOUBLE=1.5;
        Met->Par[4].Val.V_RGDOUBLE=1.0e-7;

    }

    return OK;
}

PricingMethod MET(FD_Psor_DownOut)=
{
    "FD_Psor_DownOut",
    {"SpaceStepNumber",INT2,{100},ALLOW    },{"TimeStepNumber",INT2,{100},ALLOW},
    {"Theta",RGDOUBLE051,{100},ALLOW},{"Omega",RGDOUBLE12,{100},ALLOW}, {"Epsilon",RGDOUBLE,{100},ALLOW}, {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(FD_Psor_DownOut),
    {"Price",DOUBLE,{100},FORBID}, {"Delta",DOUBLE,{100},FORBID}, {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(FD_Psor_DownOut),
    CHK_psor,
    MET(Init)
};

```

References