

Help

```

#include <stdlib.h>
#include "bs1d_doublim.h"
#include "error_msg.h"

static int Gauss_Out(int am,double s,NumFunc_1 *L,
    NumFunc_1 *U,NumFunc_1 *Rebate,NumFunc_1 *p,double t,double r,
    double divid,double sigma,int N,int M,double theta,double *pt
    price,double *ptdelta)
{
    int      Index,PriceIndex,TimeIndex;
    double   k,vv,h,z,alpha,beta,gamma,y,alpha1,beta1,gamma1,
        down,u,l,rebate,up,upwind_alphacoef;
    double   *Obst,*A,*B,*C,*P,*S,pricenh,pricen2h,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 Step*/
    u=(U->Compute)(U->Par,0);
    l=(L->Compute)(L->Par,0);

```

```

rebate=(Rebate->Compute)(Rebate->Par,0);
y=log(s);
down=log(1);
up=log(u);
h=(up-down)/(double)(N+1);

/*Peclet Condition-Coefficient of diffusion augmented */
vv=0.5*SQR(sigma);
z=(r-divid)-vv;
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));

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/(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));

/*Set Up 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];

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

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
{
    /*Init 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[
PriceIndex]+
        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[PriceI
ndex+1];

    P[1] =S[1]/B[1];

    for(PriceIndex=2;PriceIndex<=N;PriceIndex++)
        P[PriceIndex]=S[PriceIndex]/B[PriceIndex]-A[PriceI
ndex]*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((y-down)/h);

```

```

/*Price*/
if ((y==up)&&(y==down))
    *ptprice=P[0];
else
    *ptprice=P[Index]+(P[Index+1]-P[Index])*(exp(y)-exp(down+Index*h))/(exp(down+(Index+1)*h)-exp(down+Index*h));

/*Delta*/
if ((y==up)&&(y==down))
    *ptdelta=0.0;
else {
    pricenh=P[Index+1]+(P[Index+2]-P[Index+1])*(exp(y+h)-exp(down+(Index+1)*h))/(exp(down+(Index+2)*h)-exp(down+(Index+1)*h));
    if (Index>0) {
        priceph=P[Index-1]+(P[Index]-P[Index-1])*(exp(y-h)-exp(down+(Index-1)*h))/(exp(down+(Index)*h)-exp(down+(Index-1)*h));
        *ptdelta=(pricenh-priceph)/(2*s*h);
    } else {
        pricen2h=P[Index+2]+(P[Index+3]-P[Index+2])*(exp(y+2*h)-exp(down+(Index+2)*h))/(exp(down+(Index+3)*h)-exp(down+(Index+2)*h));
        *ptdelta=(4*pricenh-pricen2h-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_Out)(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 Gauss_Out(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val.
        V_PDOUBLE,
                        ptOpt->LowerLimit.Val.V_NUMFUNC_1,ptOpt->
        >UpperLimit.Val.V_NUMFUNC_1,ptOpt->Rebate.Val.V_NUMFUNC_1,
        ptOpt->PayOff.Val.V_NUMFUNC_1,
                        ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.
        V_DATE,r,divid,ptMod->Sigma.Val.V_PDOUBLE,
                        Met->Par[0].Val.V_INT,Met->Par[1].Val.V_
        INT, Met->Par[2].Val.V_RGDOUBLE,
                        &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1
        ].Val.V_DOUBLE));
}

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

    if ((opt->Parisian).Val.V_BOOL==WRONG)
        if ((opt->OutOrIn).Val.V_BOOL==OUT)
            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=1.0;
    }

    return OK;
}

PricingMethod MET(FD_Gauss_Out)=
{
    "FD_Gauss_Out",
    {{"SpaceStepNumber",INT2,{100},ALLOW},{"TimeStepNumber",
        INT2,{100},ALLOW},
        {"Theta",RGDOUBLE051,{100},ALLOW},{" " ,PREMIA_NULLTYPE,{
            0},FORBID}}},
    CALC(FD_Gauss_Out),
    {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
        ID} ,{" " ,PREMIA_NULLTYPE,{0},FORBID}}},
    CHK_OPT(FD_Gauss_Out),
    CHK_split,
    MET(Init)
};

```

References