

[Help](#)

```
#include "bs1d_lim.h"

static int Gauss_DownIn(int am,double s,NumFunc_1
    *p,double l,double rebate,double t,double r,
    double divid,double sigma,int N,int M,double theta,
    double *ptprice,double *ptdelta)
{
    int      Index,PriceIndex,TimeIndex;
    double   k,vv,loc,h,z,alpha,beta,gamma,y,alpha1
        ,beta1,gamma1,down,upwind_alphacoef;
    double   *Obst,*A,*B,*C,*P,*S,price1,price2,pr
        icenh,pricen2h,priceph;

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

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

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

```

    bs(z)*t;

/*Space Step*/
y=log(s);
down=log(1);
h=(loc+y-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 if (z<=0.) 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 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+1;PriceIndex++)
{
    Obst[PriceIndex]=(p->Compute)(p->Par,exp(
    down+(double)PriceIndex*h));
    P[PriceIndex]= rebate;
}
price2=(p->Compute)(p->Par,1);

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
{
    /*Set Rhs*/
    price1=Boundary(1,p,TimeIndex*k,r,divid,sig
    ma);
    S[1]=beta1*P[1]+gamma1*P[2]+alpha1*price2-
    alpha*price1;
    price2=price1;

    for(PriceIndex=1+1;PriceIndex<N;PriceIndex+
    +)
        S[PriceIndex]= alpha1*P[PriceIndex-1]+bet
        a1*P[PriceIndex]+
        gamma1*P[PriceIndex+1];
    S[N]=alpha1*P[N-1]+beta1*P[N]+gamma1*P[N+1]
    -gamma*P[N+1];

    /*Solve the system*/
    for(PriceIndex=N-1;PriceIndex>=1;PriceInd
    ex--)
        S[PriceIndex]=S[PriceIndex]-C[PriceIndex]
        *S[PriceIndex+1];

    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((y-down)/h);
    P[0]=price1;

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

    /*Delta*/
    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+(Ind
            ex+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_DownIn)(void *Opt,void *Mod,Pr
    icingMethod *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 Gauss_DownIn(ptOpt->EuOrAm.Val.V_BOOL,
        ptMod->S0.Val.V_PDOUBLE,pt
    Opt->PayOff.Val.V_NUMFUNC_1,
        limit,rebate,ptOpt->Maturi
    ty.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_RGDOUBLE05
    1,
        &(Met->Res[0].Val.V_DOUBLE)
    ,&(Met->Res[1].Val.V_DOUBLE));
}

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

    if ((opt->OutOrIn).Val.V_BOOL==IN)
        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;
        Met->Par[1].Val.V_INT2=100;
        Met->Par[2].Val.V_RGDOUBLE=0.5;

        first=0;
    }

    return OK;
}

PricingMethod MET(FD_Gauss_DownIn)=
{
    "FD_Gauss",
    {{ "SpaceStepNumber", INT2, 100, ALLOW    }, {"TimeStepNumber", INT2, 100, ALLOW},
    {"Theta", RGDOUBLE051, 100, ALLOW}, {" " , END, 0, FORBID}}},
    CALC(FD_Gauss_DownIn),
    {{ "Price", DOUBLE, 100, FORBID}, {"Delta", DOUBLE, 100, FORBID}, {" " , END, 0, FORBID}}},
    CHK_OPT(FD_Gauss_DownIn),
    CHK_split,
    MET(Init)
};

```

References