

[Help](#)

```
#include "bs1d_lim.h"

static double initial_mesh(int upordown,double re
    finement,double x_min,double x_max,double x0)
{

    double atrois;
    double acinq;
    double temp;
    double x;
    double inref;

    inref=1./refinement;
    x=(x0-x_min)/(x_max-x_min)-0.5; /* t in [-0.5,0.
        5] */
    temp=x;

    if (inref>=0.2) /*inref > 0.17007... !).*/
    {
        if (upordown==0)
        {
            acinq = 8*(2.0*inref + 1.0/inref - 3.0);
            atrois = 2*(5.0 - 4.0*inref - 1.0/inref);
            x = x/2.0+0.25;
            temp = inref*x + atrois*x*x*x + acinq*x*x*x
                *x*x;
            temp = 2.0*temp-0.5;
        }
        else
        {
            acinq = 8.0*(2.0*inref + 1.0/inref - 3.0);
            atrois = 2.0*(5.0 - 4.0*inref - 1.0/inref);
            x = x/2.0-0.25;
            temp = inref*x + atrois*x*x*x + acinq*x*x*x
                *x*x;
            temp = 2.0*temp+0.5;
        }
    }

    return (temp+0.5)*(x_max-x_min)+x_min;
```

```

}

static void new_mesh(double time,double *old_x,
    double z,double *new_x,int N)
{
    double new_x_min,new_x_max,rho;
    int i;

    new_x_min = old_x[0] + z*time;
    new_x_max = old_x[N] + z*time;
    rho = (new_x_max - new_x_min)/(old_x[N]-old_x[0]
    );

    for (i=0; i<=N;i++)
    new_x[i] = new_x_min + rho*(old_x[i]-old_x[0])
    ;

    return;
}

static int Fem_Out(int upordown,int am,double s,
    NumFunc_1 *p,NumFunc_1 *l,double rebate,double t,
    double r,double divid,double sigma,int N,int M,double
    theta,double mu2,double nu2,double refinement,
    double *ptprice,double *ptdelta)
{
    int i,TimeIndex;
    double vv,loc,h,z,V0,VN,A0,AN,Dir_low,Dir_up,
        Neu_low,Neu_up,sigma2;
    double nu1,mu1,time_mesh,x_min,x_max,x0,nu_
        low,nu_up,mu_low,mu_up;
    double *alpha,*beta,*gamma,*alpha1,*beta1,*g
        amma1,*old_x;
    double *new_x,*V,*Vp,*beta_p,*P_New,*P_Old,*
        temp;

    /*Memory Allocation*/
    alpha=(double *)malloc((N+1)*sizeof(double));

```

```
if (alpha==NULL)
    return MEMORY_ALLOCATION_FAILURE;

beta=(double *)malloc((N+1)*sizeof(double));
if (beta==NULL)
    return MEMORY_ALLOCATION_FAILURE;

gamma=(double *)malloc((N+1)*sizeof(double));
if (gamma==NULL)
    return MEMORY_ALLOCATION_FAILURE;

alpha1=(double *)malloc((N+1)*sizeof(double));
if (alpha1==NULL)
    return MEMORY_ALLOCATION_FAILURE;

beta1=(double *)malloc((N+1)*sizeof(double));
if (beta1==NULL)
    return MEMORY_ALLOCATION_FAILURE;

gamma1=(double *)malloc((N+1)*sizeof(double));
if (gamma1==NULL)
    return MEMORY_ALLOCATION_FAILURE;

old_x=(double *)malloc((N+1)*sizeof(double));
if (old_x==NULL)
    return MEMORY_ALLOCATION_FAILURE;

new_x=(double *)malloc((N+1)*sizeof(double));
if (new_x==NULL)
    return MEMORY_ALLOCATION_FAILURE;

V=(double *)malloc((N+1)*sizeof(double));
if (V==NULL)
    return MEMORY_ALLOCATION_FAILURE;

Vp=(double *)malloc((N+1)*sizeof(double));
if (Vp==NULL)
    return MEMORY_ALLOCATION_FAILURE;

beta_p=(double *)malloc((N+1)*sizeof(double));
if (beta_p==NULL)
```

```

    return MEMORY_ALLOCATION_FAILURE;

P_New=(double *)malloc((N+1)*sizeof(double));
if (P_New==NULL)
    return MEMORY_ALLOCATION_FAILURE;

P_Old=(double *)malloc((N+1)*sizeof(double));
if (P_Old==NULL)
    return MEMORY_ALLOCATION_FAILURE;

temp=(double *)malloc((N+1)*sizeof(double));
if (temp==NULL)
    return MEMORY_ALLOCATION_FAILURE;

/*Dirichlet on the barrier*/
nu1=0.;
mu1=1.;

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

/*Space Localisation*/
sigma2=sigma*sigma;
vv=0.5*sigma2;
z=(r-divid);

loc=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fa
    bs((z-vv)*t);
h=0.001;

/*Terminal Values*/
if (upordown==0)/*Down Case*/
{
    x_min=log(((l->Compute)(l->Par,t))/s)-z*t;
    x_max=loc;
}
else/*Up Case*/
{
    x_min=-loc;
    x_max=log(((l->Compute)(l->Par,t))/s)-z*t;
}

```

```

for(i=0;i<=N;i++)
{
    x0=x_min+((double)i)*(x_max-x_min)/(double)
    N;
    old_x[i]=initial_mesh(upordown,refinement,x_
    min,x_max,x0);
    P_Old[i]=exp(-r*t)*(p->Compute)(p->Par,s*exp
    (old_x[i]+z*t));
}

if (upordown==0)/*Down Case*/
P_Old[0] = exp(-r*t)*rebate;
else/*Up Case*/
P_Old[N] = exp(-r*t)*rebate;

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)
{
    /*New Mesh Computing*/
    if (upordown==0)/*Down Case*/
    {
        x_min=log(((l->Compute) (l->Par,t-(
        double)TimeIndex*time_mesh))/s)-z*(t-(double)TimeInd
        ex*time_mesh);
        x_max=loc;
    }
    else/*Up Case*/
    {
        x_min=-loc;
        x_max=log(((l->Compute)(l->Par,t-(double)
        TimeIndex*time_mesh))/s)-z*(t-(double)TimeIndex*time_
        mesh);
    }
    /*New Mesh Generation*/
    new_mesh(time_mesh,old_x,z,new_x,N);

    /*Computation of Lhs coefficients*/
    for(i=1;i<N;i++)
    {

```

```

    alpha[i]=(-vv*theta*time_mesh*(1.+2.0/(new_x[i]-new_x[i-1]))
              -theta*(old_x[i-1]-new_x[i-1]));
    beta[i]=(new_x[i+1]-new_x[i-1]
             +sigma2*theta*time_mesh*(1.0/(new_x[i+1]-new_x[i])
             +1.0/(new_x[i]-new_x[i-1])));
    gamma[i]=(vv*theta*time_mesh*(1.-2.0/(new_x[i+1]-new_x[i]))
              +theta*(old_x[i+1]-new_x[i+1]));
}

/*Computation of Rhs coefficients*/
for(i=1;i<N;i++)
{
    alpha1[i]=(vv*(1.0-theta)*time_mesh*(1.+2.0/(old_x[i]-old_x[i-1]))
              +(1.0-theta)*(old_x[i-1]-new_x[i-1]));
    beta1[i]=(old_x[i+1]-old_x[i-1]
              -sigma2*(1.0-theta)*time_mesh*(1.0/(old_x[i+1]-old_x[i])
              +1.0/(old_x[i]-old_x[i-1])));
    gamma1[i]=(-vv*(1.0-theta)*time_mesh*(1.-2.0/(old_x[i+1]-old_x[i]))
               -(1.0-theta)*(old_x[i+1]-new_x[i+1]));
}

/*Right factor*/
for (i=1;i<=N-1;i++)
V[i]=alpha1[i]*P_Old[i-1]+beta1[i]*P_Old[i]
+gamma1[i]*P_Old[i+1];

/*Robin Boundary Condition in the Down Case*/
/
if(upordown==0)
{
    Dir_low=exp(-r*(t-(double)TimeIndex*time_

```

```

mesh))*rebate;
    Neu_low=0.;
    nu_low=nu1;
    mu_low=mu1;

    V0 = (new_x[1]-new_x[0])*(mu_low*Dir_low+
nu_low*Neu_low)/
    (mu_low*(new_x[1]-new_x[0])-nu_low);
    V[1] -= alpha[1]*V0; /*Robin low */
    A0 = nu_low/(mu_low*(new_x[1]-new_x[0])-
nu_low);
    beta[1] -= alpha[1]*A0;

    Dir_up=exp(-r*(t-(double)TimeIndex*time_
mesh))*(p->Compute)(p->Par,s*exp(new_x[N]+z*(t-(
double)TimeIndex*time_mesh))));

    /*Neumann condition is computed numeri
cally*/
    Neu_up=exp(-r*(t-(double)TimeIndex*time_
mesh))*((p->Compute)(p->Par,s*exp(new_x[N]+h+z*(t-
(double)TimeIndex*time_mesh)))-(p->Compute)(p->
Par,s*exp(new_x[N]-h+z*(t-(double)TimeIndex*time_
mesh))))/(2.*h);

    nu_up=nu2;
    mu_up=mu2;

    VN = (new_x[N]-new_x[N-1])*(mu_up*Dir_up+
nu_up*Neu_up)/
    (mu_up*(new_x[N]-new_x[N-1])+nu_up);

    V[N-1] -= gamma[N-1]*VN; /*Robin up*/
    AN = -nu_up/(mu_up*(new_x[N]-new_x[N-1])+
nu_up);
    beta[N-1] -= gamma[N-1]*AN;
}

```

```

else /*Robin Boundary Condition in the Up
Case*/
{
    Dir_low=exp(-r*(t-(double)TimeIndex*time_
mesh))*(p->Compute)(p->Par,s*exp(new_x[0]+z*(t-(
double)TimeIndex*time_mesh)));
    Neu_low=exp(-r*(t-(double)TimeIndex*time_
mesh))*((p->Compute)(p->Par,s*exp(new_x[0]+h+z*(t-
(double)TimeIndex*time_mesh)))-(p->Compute)(p->
Par,s*exp(new_x[0]-h+z*(t-(double)TimeIndex*time_
mesh))))/(2.*h);
    nu_low=nu2;
    mu_low=mu2;

    V0 = (new_x[1]-new_x[0])*(mu_low*Dir_low+
nu_low*Neu_low)/
    (mu_low*(new_x[1]-new_x[0])-nu_low);
    V[1] -= alpha[1]*V0; /*Robin low */
    A0 = nu_low/(mu_low*(new_x[1]-new_x[0])-
nu_low);
    beta[1] -= alpha[1]*A0;

    Dir_up=exp(-r*(t-(double)TimeIndex*time_
mesh))*rebat;
    /*Neumann condition is computed numeri
cally*/
    Neu_up=0.;

    nu_up=nu1;
    mu_up=mu1;

    VN = (new_x[N]-new_x[N-1])*(mu_up*Dir_up+
nu_up*Neu_up)/
    (mu_up*(new_x[N]-new_x[N-1])+nu_up);

    V[N-1] -= gamma[N-1]*VN; /*Robin up*/
    AN = -nu_up/(mu_up*(new_x[N]-new_x[N-1])+
nu_up);
    beta[N-1] -= gamma[N-1]*AN;

```



```

}

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

for(i=N-2;i>=1;i--)
{
    beta_p[i]=beta[i]-gamma[i]*alpha[i+1]/beta_p[i+1];
    Vp[i]=V[i]-gamma[i]*Vp[i+1]/beta_p[i+1];
}
P_New[1]=Vp[1]/beta_p[1];

for (i=2;i<=N-1;i++)
    P_New[i]=(Vp[i]-alpha[i]*P_New[i-1])/beta_p[i];

/*Splitting for the american case*/
if(am)
    for (i=1;i<=N-1;i++)
        P_New[i]=MAX(P_New[i],exp(-r*(t-(double)TimeIndex*time_mesh))*(p->Compute)(p->Par,s*exp(old_x[i]+z*(t-(double)TimeIndex*time_mesh)))));

P_New[N]=VN-P_New[N-1]*AN;
P_New[0]=V0-P_New[1]*A0;

beta[1]+=alpha[1]*A0;
beta[N-1]+=gamma[N-1]*AN;

for(i=0;i<=N;i++)
{
    temp[i]=P_Old[i];
    P_Old[i]=P_New[i];
    P_New[i]=temp[i];
    temp[i]=old_x[i];
}

```

```

        old_x[i]=new_x[i];
        new_x[i]=temp[i];
    }

    }/*End of Time Cycle*/

    i=0;
    while (old_x[i]<0) i++;

    /*Price*/
    *ptprice=((s-s*exp(old_x[i-1]))*P_Old[i]+(s*exp
        (old_x[i])-s)*P_Old[i-1])/
        (s*(exp(old_x[i])-exp(old_x[i-1]))));

    /*Delta*/
    *ptdelta=(1.0/(s*(s*(exp(old_x[i+1])-exp(old_x[
        i-1])))))*((s*(exp(old_x[i])-exp(old_x[i-1])))*((
        P_Old[i+1]-P_Old[i])/(old_x[i+1]-old_x[i]))+s*((
        exp(old_x[i+1])-exp(old_x[i]))*(P_Old[i]-P_Old[
        i-1])/(old_x[i]-old_x[i-1]))));

    /*Memory Desallocation*/
    free(alpha);
    free(beta);
    free(gamma);
    free(alpha1);
    free(beta1);
    free(gamma1);
    free(old_x);
    free(new_x);
    free(V);
    free(Vp);
    free(beta_p);
    free(P_New);
    free(P_Old);
    free(temp);

    return OK;
}

```

```

int CALC(FD_Fem_Out)(void *Opt,void *Mod,Pricing
    Method *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid,rebate;
    int upordown;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
    rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compu
        te)((ptOpt->Rebate.Val.V_NUMFUNC_1)->Par,ptMod->T.
        Val.V_DATE);
    if ((ptOpt->DownOrUp).Val.V_BOOL==DOWN)
        upordown=0;
    else upordown=1;

    return Fem_Out(upordown,ptOpt->EuOrAm.Val.V_BO
        OL,ptMod->S0.Val.V_PDOUBLE,ptOpt->PayOff.Val.V_
        NUMFUNC_1,ptOpt->Limit.Val.V_NUMFUNC_1,rebate, ptOpt->
        Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,r,divid,pt
        Mod->Sigma.Val.V_PDOUBLE,Met->Par[0].Val.V_INT2,
        Met->Par[1].Val.V_INT2, Met->Par[2].Val.V_RGDOUBLE0
        51,Met->Par[3].Val.V_RGDOUBLE,Met->Par[4].Val.V_
        RGDOUBLE,Met->Par[5].Val.V_RGDOUBLE14,&(Met->Res[
        0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}

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

    if ((opt->OutOrIn).Val.V_BOOL==OUT)
    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;
        Met->Par[3].Val.V_RGDOUBLE=1.;
        Met->Par[4].Val.V_RGDOUBLE=0.;
        Met->Par[5].Val.V_DOUBLE=1.5;

        first=0;
    }

    return OK;
}

PricingMethod MET(FD_Fem_Out)=
{
    "FD_Fem",
    {{"SpaceStepNumber",INT2,100,ALLOW    },{"TimeS
        tepNumber",INT2,100,ALLOW},
    {"Theta",RGDOUBLE051,100,ALLOW}, {"Dirichlet W
        eights",RGDOUBLE,100,ALLOW}, {"Neumann Weights",RG
        DOUBLE,100,ALLOW}, {"Refinement",RGDOUBLE14,100,ALLOW}
        ,{" ",END,0,FORBID}}},
    CALC(FD_Fem_Out),
    {{"Price",DOUBLE,100,FORBID}, {"Delta",DOUBLE,10
        0,FORBID} ,{" ",END,0,FORBID}}},
    CHK_OPT(FD_Fem_Out),
    CHK_split,
    MET(Init)
};

```

## References