

[Help](#)

```
#include "bs1d_doublim.h"

/*Initial Mesh*/
static double initial_dmesh(double refinement,
    double x_min,double x_max,double x0,int upordown)
{

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

    inref=1./refinement;
    x=(x0-x_min)/(x_max-x_min)-0.5;

    temp=x;

    if (inref>=0.2)
    {

        acinq = 8*(2.0*inref + 1.0/inref - 3.0);
        atrois = 2*(5.0 - 4.0*inref - 1.0/inref);
        if (upordown)
            x = x/2.0+0.25;
        else
            x = x/2.0-0.25;
        temp = inref*x + atrois*x*x*x + acinq*x*x*x*x*x
            ;
        if (upordown)
            temp = 2.0*temp-0.5;
        else
            temp = 2.0*temp+0.5;
    }
    return (temp+0.5)*(x_max-x_min)+x_min;
}

/*New Mesh*/
static void new_dmesh(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 am,double s,NumFunc_1 *p,
    NumFunc_1 *L, NumFunc_1 *U,double rebate,double t,
    double r,double divid,double sigma,int N,int M,double
    theta,double refinement,double *ptprice,double *
    ptdelta)
{
    int i,TimeIndex,upordown;
    double vv,z,Dir_low,Dir_up,sigma2;
    double time_mesh,x_min,x_max,x0;
    double *alpha,*beta,*gamma,*alpha1,*beta1,*g
        amma1,*old_x,h;
    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;

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

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

/*Terminal Values*/
x_min=log(((L->Compute)(L->Par,t))/s)-z*t;
x_max=log(((U->Compute)(U->Par,t))/s)-z*t;
h=(x_max-x_min)/N;

for(i=0;i<N/2;i++)
{
    x0=x_min+((double)i)*(x_max-x_min)/(double)N;
    upordown=1;
    old_x[i]=initial_dmesh(refinement,x_min,x_min+(
        x_max-x_min)/2.,x0,upordown);
    P_Old[i]=exp(-r*t)*(p->Compute)(p->Par,s*exp(ol
        d_x[i]+z*t));
}
for(i=N/2;i<=N;i++)
{
    x0=x_min+((double)i)*(x_max-x_min)/(double)N;
    upordown=0;
    old_x[i]=initial_dmesh(refinement,x_min+(x_max-
        x_min)/2.,x_max,x0,upordown);
    P_Old[i]=exp(-r*t)*(p->Compute)(p->Par,s*exp(ol
        d_x[i]+z*t));
}
P_Old[0]=exp(-r*t)*rebate;
P_Old[N]=exp(-r*t)*rebate;

/*Finite Difference Cycle*/
for(TimeIndex=1;TimeIndex<=M;TimeIndex++)

```

```

{
/*New Mesh Computing*/
x_min=log(((L->Compute)(L->Par,t-(double)
    TimeIndex*time_mesh))/s)-z*(t-(double)TimeIndex*time_
    mesh);
x_max=log(((U->Compute)(U->Par,t-(double)
    TimeIndex*time_mesh))/s)-z*(t-(double)TimeIndex*time_
    mesh);
new_dmesh(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/(ol
        d_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]-ol
        d_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];

/*Dirichlet Boundary Condition*/
Dir_low=exp(-r*(t-(double)TimeIndex*time_mesh))*rebate;

V[1]-=alpha[1]*Dir_low;

Dir_up=exp(-r*(t-(double)TimeIndex*time_mesh))*rebate;

V[N-1]-=gamma[N-1]*Dir_up;

/*Gauss method*/
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]=Dir_up;
P_New[0]=Dir_low;

```

```

    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;

    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);

    return Fem_Out(ptOpt->EuOrAm.Val.V_BOOL,ptMod->
        SO.Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,
        ptOpt->LowerLimit.Val.V_NUMFUNC_1,ptOpt->UpperLim
        it.Val.V_NUMFUNC_1,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_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_DOUBLE=1.5;

        first=0;
    }

    return OK;
}

PricingMethod MET(FD_Fem_Out)=
{
    "FD_Fem_Out",
    {{"SpaceStepNumber",INT2,100,ALLOW    },{"TimeS
        tepNumber",INT2,100,ALLOW},
        {"Theta",RGDOUBLE051,100,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