

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/bs1d/bs1d_doublim/bs1d_doublim_h_src.pdfbs1d_doublim.h"
#include "
href../../common/error_msg_h_src.pdferror_msg.h"

/*Initial Mesh*/
static double initial_dmesh(double refinement, double x_min, double x_max, double x0)
{
    double atois;
    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);
        atois = 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 + atois * 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)
{
    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, *gamma1, *old_x;
    double *new_x, *V, *Vp, *beta_p, *P_New, *P_Old, *temp;

    /*Memory Allocation*/
    alpha = malloc((N + 1) * sizeof(double));
    if (alpha == NULL)
        return MEMORY_ALLOCATION_FAILURE;

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

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

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

```

```

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

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

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

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

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

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

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

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

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

temp = 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;

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.,
    P_Old[i] = exp(-r * t) * (p->Compute)(p->Par, s * exp(old_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,
    P_Old[i] = exp(-r * t) * (p->Compute)(p->Par, s * exp(old_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)
    x_max = log(((U->Compute)(U->Par, t - (double)TimeIndex * time_mesh)) / s)
    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

```

```

        - 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] -
        + (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] -
        + 1.0 / (old_x[i] - old_x[i - 1]))));
    gamma1[i] = (-vv * (1.0 - theta) * time_mesh * (1. - 2.0 / (old_x[i + 1] -
        - (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_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) *
            (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 *

/*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, PricingMethod *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)->Compute)((ptOpt->Rebate.Val.V_NUMFU

    return Fem_Out(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOf
}

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

    if ((opt->TwoDoubleStep).Val.V_BOOL == FALSE)
        if ((opt->OutOrIn).Val.V_BOOL == OUT)
            if ((opt->Parisian).Val.V_BOOL == FALSE)
                return OK;

    return WRONG;
}

```

```

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "fd_fem_updownout_bs";

        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;
    }
    return OK;
}

PricingMethod MET(FD_Fem_Out) =
{
    "FD_Fem_Out",
    { {"SpaceStepNumber", INT2, {100}, ALLOW    }, {"TimeStepNumber", INT2, {100},
        {"Theta", RGDOUBLE051, {100}, ALLOW}, {"Refinement", RGDOUBLE14, {100}, ALLO
    },
    CALC(FD_Fem_Out),
    {"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
    CHK_OPT(FD_Fem_Out),
    CHK_split,
    MET(Init)
};

```