

[Help](#)

```
#include <stdlib.h>
#include "mer1d_std.h"
#include "enums.h"

static int expl_1d(int am, PARAM p, DENSITY g, MESH m, WEIGHT w, IMESH Im, NumFu
{
    int j, i;
    double integral, *weight, *sol_a, *sol_b, *boundary, *Obst;

    /* vector allocation */
    sol_a = malloc(m.N * sizeof(double));
    Obst = malloc(m.N * sizeof(double));
    if (Obst == NULL) return MEMORY_ALLOCATION_FAILURE;
    if (sol_a == NULL) return MEMORY_ALLOCATION_FAILURE;
    memset(sol_a, 0, m.N * sizeof(double));
    sol_b = malloc(m.N * sizeof(double));
    if (sol_b == NULL) return MEMORY_ALLOCATION_FAILURE;
    memset(sol_b, 0, m.N * sizeof(double));
    weight = malloc(Im.N * sizeof(double));
    if (weight == NULL) return MEMORY_ALLOCATION_FAILURE;
    memset(weight, 0, Im.N * sizeof(double));
    boundary = malloc(m.N * sizeof(double));
    if (boundary == NULL) return MEMORY_ALLOCATION_FAILURE;
    memset(boundary, 0, m.N * sizeof(double));

    /* set integral weights */
    d1_intcomp(Im.N, m.h, weight, g.d, SIMP);

    /*Terminal Values*/
    for (j = 0; j < m.N; j++)
    {
        sol_a[j] = (p_func->Compute)(p_func->Par, exp(m.xmin + j * m.h));
        Obst[j] = sol_a[j];
    }

    /* set boundary */
    set_boundaryAA(bound, m, p, Im, sol_a, boundary);

    /*Finite Difference Cycle*/
```

```

for (i = 1; i <= m.M; i++)
{
    for (j = 0; j < Im.min; j++)
    {
        sol_b[j] = boundary[j]; /* boundary */
    }
    for (j = Im.max; j < m.N; j++)
    {
        sol_b[j] = boundary[j]; /* boundary */
    }

    for (j = Im.min; j < Im.max; j++)
    {
        integral = calc_int(Im.N, weight, &sol_a[j - Im.min]);
        sol_b[j] = m.k * w.p1 * sol_a[j - 1] + (1.0 - m.k * w.p2) * sol_a[j] +
    }

    for (j = 0; j < m.N; j++)
    {
        sol_a[j] = sol_b[j];

        if (am)
            sol_a[j] = MAX(Obst[j], sol_a[j]);
    }

}

/* Price */
*ptprice = sol_a[m.Index];
/*Delta*/
*ptdelta = (sol_a[m.Index + 1] - sol_a[m.Index - 1]) / (2.0 * p.s * m.h);

/* Memory Desallocation */
free(sol_a);
free(sol_b);
free(weight);
free(boundary);
free(Obst);

return RETURNOK;
}

```

```

static int Explicit(int am, double s, NumFunc_1 *p_func, double t, double r, do
{
    MESH m;
    WEIGHT w;
    IMESH Im;
    PARAM p;
    DENSITY g;
    EQ eq;
    double K;

    K = p_func->Par[0].Val.V_DOUBLE;
    Gaussian_data(mu, gamma2, &g);
    set_parameter(s, K, t, r, sigma, divid, lambda, g.Eu, &p);
    equation(p, &eq);

    if (N % 2 == 1) N++;

    initgrid_1Dbis(p, g, eq, N, &m, &Im);
    set_weights_espl(p.T, eq, &m, &w);
    Gaussian_vect(0, Im.N, m.h, &g);
    expl_1d(am, p, g, m, w, Im, p_func, bound, ptprice, ptdelta);
    freeDensity(&g);
    return OK;
}

int CALC(FD_Explicit)(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 Explicit(ptOpt->EuOrAm.Val.V_BOOL,
                    ptMod->S0.Val.V_PDOUBLE,
                    ptOpt->PayOff.Val.V_NUMFUNC_1,
                    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                    r,
                    divid,

```

```

        ptMod->Sigma.Val.V_PDOUBLE,
        ptMod->Lambda.Val.V_PDOUBLE,
        ptMod->Mean.Val.V_PDOUBLE,
        ptMod->Variance.Val.V_PDOUBLE,
        Met->Par[0].Val.V_INT,
        Met->Par[1].Val.V_ENUM.value,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE));
    }

static int CHK_OPT(FD_Explicit)(void *Opt, void *Mod)
{
    return OK;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "FD_Explicit_mer";
        Met->Par[0].Val.V_INT2 = 1000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumBoundaryCond;
    }

    return OK;
}

PricingMethod MET(FD_Explicit) =
{
    "FD_Explicit",
    { {"SpaceStepNumber", INT2, {500}, ALLOW },
      {"Boundary Condition", ENUM, {1}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_Explicit),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    }
};

```

```
    },  
    CHK_OPT(FD_Explicit),  
    CHK_split,  
    MET(Init)  
};
```