

[Help](#)

```

extern "C" {
#include "mer1d_lim.h"
#include "enums.h"
}
#include "math/levy_fd.h"
extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
    static int CHK_OPT(FD_ImpExpUpOut)(void *Opt, void *Mod)
    {
        return NONACTIVE;
    }
    int CALC(FD_ImpExpUpOut)(void *Opt, void *Mod, PricingMethod *Met)
    {
        return AVAILABLE_IN_FULL_PREMIA;
    }
#else
    static int ImpExpUpOut(int am, double S0, NumFunc_1 *p, double l_up, double r
    {

        double price0, delta0;
        int flag_callput, flag_stdbarrier;

        double delta = sqrt(gamma2);
        double luplog = log(l_up / S0);
        if (dx > fabs(luplog) / 2.)
            dx = fabs(luplog) / 2.;
        int N1 = (int)ceil(fabs(luplog) / dx);
        dx = fabs(luplog) / N1;

        double Ar = luplog - dx;
        /*Construction of the model*/
        Merton_measure measure(mu, delta, lambda, sigma, dx);
        double k = 3;
        double A1 = log(2. / 3) + T * measure.espX1 - k * sqrt(T * measure.varX1);

        if (A1 < -30) A1 = -30;
        int N = (int) ceil((luplog - A1) / dx);
        A1 = luplog - N * dx;

```

```

double K = p->Par[0].Val.V_DOUBLE;

flag_stdbarrier = 2;

/*Price Computation*/
if ((p->Compute) == &Put)
{
    flag_callput = 2;
    if (flag_scheme == 1)
        vector<double> u = price2(am, measure, flag_callput, flag_stdbarrier,
    else
        vector<double> u = price2c(am, measure, flag_callput, flag_stdbarrier,
    /*Price */
    *ptprice = price0;

    /*Delta */
    *ptdelta = delta0;
}
else if ((p->Compute) == &Call)
{
    /*Price */
    flag_callput = 1;
    if (flag_scheme == 1)
        vector<double> u = price2(am, measure, flag_callput, flag_stdbarrier,
    else
        vector<double> u = price2c(am, measure, flag_callput, flag_stdbarrier,
    *ptprice = price0;

    /*Delta */
    *ptdelta = delta0;
}

return OK;
}

int CALC(FD_ImpExpUpOut)(void *Opt, void *Mod, PricingMethod *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)->Compute)((ptOpt->Limit.Val.V_NUMFUN
    rebate = ((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt->Rebate.Val.V_NUM

    return ImpExpUpOut(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
                       ptOpt->PayOff.Val.V_NUMFUNC_1, limit, rebate, ptOpt->Matu
}

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

    if ((opt->OutOrIn).Val.V_BOOL == OUT)
        if ((opt->DownOrUp).Val.V_BOOL == UP)
            if ((opt->EuOrAm).Val.V_BOOL == EURO)
                if ((opt->Parisian).Val.V_BOOL == FALSE)
                    return OK;

    return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    static int first = 1;

    if (first)
    {
        Met->HelpFilenameHint = "FD_ImpExpUpOut_mer";
        Met->Par[0].Val.V_PDOUBLE = 0.001;
        Met->Par[1].Val.V_INT2 = 400;
        Met->Par[2].Val.V_ENUM.value = 1;
        Met->Par[2].Val.V_ENUM.members = &PremiaEnumExpPart;
        first = 0;
    }

    return OK;
}

```

```
PricingMethod MET(FD_ImpExpUpOut) =
{
    "FD_ImpExpUpOut",
    { {"Space Discretization Step", DOUBLE, {500}, ALLOW}, {"TimeStepNumber", IN
        {"Explicit Part", ENUM, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_ImpExpUpOut),
    {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID}, {" ", P
    CHK_OPT(FD_ImpExpUpOut),
    CHK_split,
    MET(Init)
};
}
```