

[Help](#)

```

#include "mer1d_std.h"

static int MCMerton(double s, NumFunc_1 *p,
    double t, double r, double divid, double sigma, double
    lambda, double mu, double gamma2, long N, int genera
    tor, double inc, double confidence, double *ptpric
    e, double *ptdelta, double *pterror_price,
    double *pterror_delta , double *inf_price, double *su
    p_price, double *inf_delta, double *sup_delta)
{
    int flag;
    long i;
    double mean_price, mean_delta, var_price, var_
        delta, forward, forward_stock, price_sample_plus,
        price_sample_minus, exp_sigmaxwt, S_T, U_T, pric
        e_sample, delta_sample=0., s_plus, s_minus, sigma_
        sqrt;
    double g;
    int init_mc, mc_or_qmc;
    int simulation_dim= 1;
    double alpha, z_alpha;
    double eps=1.0;

    long nj,j;
    double poisson_jump,mm,Eu;

    Eu= exp(mu+0.5*gamma2)-1.;
    mm = r-divid-lambda*Eu;
    /* Value to construct the confidence interval *
    /
    alpha= (1.- confidence)/2.;
    z_alpha= Inverse_erf(1.- alpha);

    /*Initialisation*/
    flag= 0;
    s_plus= s*(1.+inc);
    s_minus= s*(1.-inc);
    mean_price= 0.0;
    mean_delta= 0.0;

```

```
var_price= 0.0;
var_delta= 0.0;

/*Median forward stock and delta values*/
sigma_sqrt=sigma*sqrt(t);
forward= exp((mm-SQR(sigma)/2.0)*t);
forward_stock= s*forward;

/*MC sampling*/
init_mc= InitGenerator(generator,simulation_dim
,N);

/* Test after initialization for the generator
*/
if(init_mc == OK)
{
    mc_or_qmc= Rand_Or_Quasi(generator);

    /* Begin N iterations */
    for(i=1 ; i<=N ; i++)
    {
        /* Simulation of a gaussian variable according to the generator type,
        that is Monte Carlo or Quasi Monte Carlo.
        */
        g= Gaussians[mc_or_qmc](1, CREATE, 0, generator);
        exp_sigmaxwt=exp(sigma_sqrt*g);

        /* Jump */
        nj = Poisson(lambda*t,generator);
        poisson_jump = 1.;
        for (j=1;j<=nj;j++){
            g = Gaussians[mc_or_qmc](1, CREATE, 0, generator);
            poisson_jump *= (exp(mu+sqrt(gamma2)*g));
        }

        S_T= forward_stock*exp_sigmaxwt*poisson_jump;
```

```

    U_T= forward*exp_sigmaxwt*poisson_jump;

    /*Price*/
    price_sample=(p->Compute)(p->Par,S_T);

    /*Delta*/
    price_sample_plus= (p->Compute)(p->Par, U_T*
    s_plus);
    price_sample_minus= (p->Compute)(p->Par, U_
    T*s_minus);
    delta_sample= (price_sample_plus-price_samp
    le_minus)/(2.*s*inc);
    /*Sum*/
    mean_price+= price_sample;
    mean_delta+= delta_sample;

    /*Sum of squares*/
    var_price+= SQR(price_sample);
    var_delta+= SQR(delta_sample);
}

    /* End N iterations */

    /* Price */
    *ptprice=exp(-r*t)*(mean_price/(double) N);
    *pterror_price=sqrt(exp(-2.0*r*t)*var_pric
    e/(double)N - SQR(*ptprice))/sqrt(N-1);

    /*Delta*/
    *ptdelta=exp(-r*t)*mean_delta/(double) N;
    *pterror_delta= sqrt(exp(-2.0*r*t)*(var_de
    lta/(double)N-SQR(*ptdelta)))/sqrt((double)N-1);

    /* Price Confidence Interval */
    *inf_price= *ptprice - z_alpha*(*pterror_p
    rice);
    *sup_price= *ptprice + z_alpha*(*pterror_p
    rice);

    /* Delta Confidence Interval */
    *inf_delta= *ptdelta - z_alpha*(*pterror_d

```

```

    elta);
    *sup_delta= *ptdelta + z_alpha*(*pterror_d
    elta);
}
return init_mc;
}

```

```

int CALC(MC_Merton)(void *Opt, void *Mod, Pricing
    Method *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 MCMerton(ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,
        ptOpt->Maturity.Val.V_DATE-pt
        Mod->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_LONG,
        Met->Par[1].Val.V_INT,
        Met->Par[2].Val.V_PDOUBLE,
        Met->Par[3].Val.V_DOUBLE,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE),
        &(Met->Res[2].Val.V_DOUBLE),
        &(Met->Res[3].Val.V_DOUBLE),
        &(Met->Res[4].Val.V_DOUBLE),
        &(Met->Res[5].Val.V_DOUBLE),
        &(Met->Res[6].Val.V_DOUBLE),
        &(Met->Res[7].Val.V_DOUBLE));
}

```

```

}

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

    if ((opt->EuOrAm).Val.V_BOOL==EURO)
        return OK;

    return WRONG;
}

static int MET(Init)(PricingMethod *Met)
{
    static int first=1;
    int type_generator;

    type_generator= Met->Par[1].Val.V_INT;

    if (first)
    {
        Met->Par[0].Val.V_LONG=50000;
        Met->Par[1].Val.V_INT=0;
        Met->Par[2].Val.V_PDOUBLE=0.01;
        Met->Par[3].Val.V_DOUBLE= 0.95;
        Met->Par[4].Val.V_PDOUBLE=0.1;
        Met->Par[5].Val.V_DOUBLE= 0.0;
        Met->Par[6].Val.V_DOUBLE= 0.16;

        first=0;
    }
    if (Rand_Or_Quasi(type_generator)==QMC)
    {
        Met->Res[2].Viter=IRRELEVANT;
        Met->Res[3].Viter=IRRELEVANT;
        Met->Res[4].Viter=IRRELEVANT;
        Met->Res[5].Viter=IRRELEVANT;
        Met->Res[6].Viter=IRRELEVANT;
        Met->Res[7].Viter=IRRELEVANT;
    }
}

```

```

    }
else
{
    Met->Res[2].Viter=ALLOW;
    Met->Res[3].Viter=ALLOW;
    Met->Res[4].Viter=ALLOW;
    Met->Res[5].Viter=ALLOW;
    Met->Res[6].Viter=ALLOW;
    Met->Res[7].Viter=ALLOW;
}
return OK;
}

PricingMethod MET(MC_Merton)=
{
    "MC_Merton",
    {"N iterations",LONG,100,ALLOW},
    {"RandomGenerator",GENER,100,ALLOW},
    {"Delta Increment Rel (Digit)",PDOUBLE,100,ALLOW},
    {"Confidence Value",DOUBLE,100,ALLOW},
    {" ",END,0,FORBID}},
    CALC(MC_Merton),
    {"Price",DOUBLE,100,FORBID},
    {"Delta",DOUBLE,100,FORBID} ,
    {"Error Price",DOUBLE,100,FORBID},
    {"Error Delta",DOUBLE,100,FORBID} ,
    {"Inf Price",DOUBLE,100,FORBID},
    {"Sup Price",DOUBLE,100,FORBID} ,
    {"Inf Delta",DOUBLE,100,FORBID},
    {"Sup Delta",DOUBLE,100,FORBID} ,
    {" ",END,0,FORBID}},
    CHK_OPT(MC_Merton),
    CHK_mc,
    MET(Init)
};

```

References