

[Help](#)

```

/* European options in 2 dimensions.
   Standard Monte Carlo simulation for a
   CallMaximum - PutMinimum - Exchange or BestOf option.
   In the case of Monte Carlo simulation, the pr
   ogram provides estimations for price and delta w
   ith a confidence interval.
   In the case of Quasi-Monte Carlo simulation,
   the program just provides estimations for price
   and delta. */

#include "bs2d_std2d.h"

static int Standard2DMC(double s1, double s2,
    NumFunc_2 *p, double t, double r, double divid1,
    double divid2, double sigma1, double sigma2, double rh
    o, long N, int generator, double confidence,
    double *ptprice, double *ptdelta1, double *ptdelta2,
    double *pterror_price, double *pterror_delta1, double
    *pterror_delta2, double *inf_price, double *sup_
    price, double *inf_delta1, double *sup_delta1,
    double *inf_delta2, double *sup_delta2)
{
    int i;
    double sigma11, sigma12, sigma21, sigma22, sig
    ma11_sqrt, sigma12_sqrt, sigma21_sqrt, sigma22_sq
    rt, forward1, forward_stock1, forward_delta1, fo
    rward2, forward_stock2, forward_delta2, exp_sig
    maxwt1, exp_sigmaxwt2, g_1, g_2;
    double mean_price, mean_delta1, mean_delta2,
        var_price, var_delta1, var_delta2;
    double price_sample, delta1_sample=0., delta2_
        sample=0.;
    double U_T1, U_T2, S_T1, S_T2;
    double K1, K2, ratio;

    int init_mc, mc_or_qmc;
    int simulation_dim= 2;
    double alpha, z_alpha;

```

```

/* Value to construct the confidence interval */
/
alpha= (1.- confidence)/2.;
z_alpha= Inverse_erf(1.- alpha);

/* Initialisation */
mean_price= 0.0;
mean_delta1= 0.0;
mean_delta2= 0.0;
var_price= 0.0;
var_delta1= 0.0;
var_delta2= 0.0;

/* Covariance Matrix */
/* Coefficients of the matrix A such that A(tA)
   =Gamma */
sigma11= sigma1;
sigma12= 0.0;
sigma21= rho*sigma2;
sigma22= sigma2*sqrt(1.0-SQR(rho));
sigma11_sqrt=sigma11*sqrt(t);
sigma12_sqrt=sigma12*sqrt(t);
sigma21_sqrt=sigma21*sqrt(t);
sigma22_sqrt=sigma22*sqrt(t);

/* Median forward stock and delta values */
forward1= exp(((r-divid1)-SQR(sigma1)/2.0)*t);
forward_stock1= s1*forward1;
forward_delta1= exp(-SQR(sigma1)/2.0*t);
forward2= exp(((r-divid2)-SQR(sigma2)/2.0)*t);
forward_stock2= s2*forward2;
forward_delta2= exp(-SQR(sigma2)/2.0*t);

if ( (p->Compute) == &BestOf)
{
    K1= (p->Par[0].Val.V_PDOUBLE);
    K2= (p->Par[1].Val.V_PDOUBLE);
}
if ( (p->Compute) == &Exchange)
    ratio= (p->Par[0].Val.V_PDOUBLE);

```

```

/* 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++)
    {
        /*Gaussian Random Variables*/
        g_1= Gaussians[mc_or_qmc](simulation_dim, CR
            EATE, 0, generator);
        g_2= Gaussians[mc_or_qmc](simulation_dim, RE
            TRIEVE, 1, generator);

        exp_sigmaxwt1= exp(sigma11_sqrt*g_1);
        S_T1= forward_stock1*exp_sigmaxwt1;
        U_T1= forward1*exp_sigmaxwt1;
        exp_sigmaxwt2= exp(sigma21_sqrt*g_1 + sigma2
            2_sqrt*g_2);
        S_T2= forward_stock2*exp_sigmaxwt2;
        U_T2= forward2*exp_sigmaxwt2;

        /*Price*/
        price_sample=(p->Compute) (p->Par,S_T1, S_T2
        );

        /*Delta*/
        if ( price_sample > 0)
        {
            /*Call on on the Maximum*/
            if ( p->Compute == &CallMax)
            {
                if (S_T1 >= S_T2)
                {
                    delta2_sample= 0.;
                    delta1_sample= U_T1;
                }
            }
        }
    }
}

```

```
    }
else
{
    delta1_sample= 0.0;
    delta2_sample= U_T2;
}
}

/*Put on on the Minimum*/
if ( (p->Compute) == &PutMin)
{
    if (S_T1 <= S_T2)
    {
        delta2_sample= 0.;
        delta1_sample= -U_T1;
    }
else
{
    delta1_sample= 0.;
    delta2_sample= -U_T2;
}
}

/*Best of*/
if ( (p->Compute) == &BestOf)
{
    if (S_T1- K1 >= S_T2 - K2 )
    {
        delta2_sample= 0.;
        delta1_sample= U_T1;
    }
else
{
    delta1_sample=0.0;
    delta2_sample= U_T2;
}
}

/*Exchange*/
if ( (p->Compute) == &Exchange)
{
```

```

        delta1_sample= U_T1;
        delta2_sample= -ratio*U_T2;
    }
    }
else
    {
        delta1_sample= 0.0;
        delta2_sample= 0.0;
    }

/*Sum*/
mean_price+= price_sample;
mean_delta1+= delta1_sample;
mean_delta2+= delta2_sample;

/*Sum of squares*/
var_price+= SQR(price_sample);
var_delta1+= SQR(delta1_sample);
var_delta2+= SQR(delta2_sample);
}

/* End N iterations */

/* Price estimator */
*ptprice= exp(-r*t)*(mean_price/(double) N)
;

*pterror_price= sqrt(exp(-2.0*r*t)*var_price/(double)N - SQR(*ptprice))/sqrt(N-1);
*inf_price= *ptprice - z_alpha*(*pterror_price);
*sup_price= *ptprice + z_alpha*(*pterror_price);

/* Delta1 estimator */
*ptdelta1= exp(-r*t)*mean_delta1/(double) N;
*pterror_delta1= sqrt(exp(-2.0*r*t)*var_delta1/(double)N-SQR(*ptdelta1))/sqrt((double)N-1);
*inf_delta1= *ptdelta1 - z_alpha*(*pterror_delta1);
*sup_delta1= *ptdelta1 + z_alpha*(*pterror_delta1);

```

```

    error_delta1);

    /* Delta2 estimator */
    *ptdelta2= exp(-r*t)*mean_delta2/(double)
N;
    *pterror_delta2= sqrt(exp(-2.0*r*t)*var_delta2/(double)N-SQR(*ptdelta2))/sqrt((double)N-1);
    *inf_delta2= *ptdelta2 - z_alpha*(pt
error_delta2);
    *sup_delta2= *ptdelta2 + z_alpha*(pt
error_delta2);
}
return init_mc;
}

```

```

int CALC(MC_Standard2D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid1,divid2;

    r= log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid1= log(1.+ptMod->Divid1.Val.V_DOUBLE/100.)
;
    divid2= log(1.+ptMod->Divid2.Val.V_DOUBLE/100.)
;

    return Standard2DMC(ptMod->S01.Val.V_PDOUBLE,
        ptMod->S02.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_2,
        ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_
DATE,
        r,
        divid1,
        divid2,
        ptMod->Sigma1.Val.V_PDOUBLE,
        ptMod->Sigma2.Val.V_PDOUBLE,
        ptMod->Rho.Val.V_RGDOUBLE,

```

```

    Met->Par[0].Val.V_LONG,
    Met->Par[1].Val.V_INT,
    Met->Par[2].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),
    &(Met->Res[8].Val.V_DOUBLE),
    &(Met->Res[9].Val.V_DOUBLE),
    &(Met->Res[10].Val.V_DOUBLE),
    &(Met->Res[11].Val.V_DOUBLE)
);
}

int CHK_OPT(MC_Standard2D)(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= 10000;
    Met->Par[1].Val.V_INT= 0;
    Met->Par[2].Val.V_DOUBLE= 0.95;
    first=0;
}

if(Rand_Or_Quasi(type_generator)==QMC)
{
    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;
    Met->Res[8].Viter=IRRELEVANT;
    Met->Res[9].Viter=IRRELEVANT;
    Met->Res[10].Viter=IRRELEVANT;
    Met->Res[11].Viter=IRRELEVANT;

}
else
{
    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;
    Met->Res[8].Viter=ALLOW;
    Met->Res[9].Viter=ALLOW;
    Met->Res[10].Viter=ALLOW;
    Met->Res[11].Viter=ALLOW;
}
return OK;
}
```

```
PricingMethod MET(MC_Standard2D)=
{
```



```

"MC_Standard2D",
{{"N Iterations",LONG,1000,ALLOW},
 {"RandomGenerator",GENER,100,ALLOW},
 {"Confidence Value",DOUBLE,100,ALLOW},
 {" ",END,0,FORBID}}},
CALC(MC_Standard2D),
{{"Price",DOUBLE,100,FORBID},
 {"Delta1",DOUBLE,100,FORBID} ,
 {"Delta2",DOUBLE,100,FORBID},
 {"Error Price",DOUBLE,100,FORBID},
 {"Error Delta1",DOUBLE,100,FORBID} ,
 {"Error Delta2",DOUBLE,100,FORBID} ,
 {"Inf Price",DOUBLE,100,FORBID} ,
 {"Sup Price",DOUBLE,100,FORBID} ,
 {"Inf Delta1",DOUBLE,100,FORBID} ,
 {"Sup Delta1",DOUBLE,100,FORBID} ,
 {"Inf Delta2",DOUBLE,100,FORBID} ,
 {"Sup Delta2",DOUBLE,100,FORBID} ,
 {" ",END,0,FORBID}}},
CHK_OPT(MC_Standard2D),
CHK_ok,
MET(Init)
};

```

References