

[Help](#)

```
/* European options in 2 dimensions.
   Standard Monte Carlo simulation for a CallMaximum - PutMinimum - Exchange or
   In the case of Monte Carlo simulation, the program provides estimations for p
   In the case of Quasi-Monte Carlo simulation, the program just provides estima

#include "
href../../mod/bs2d/bs2d_std2d/bs2d_std2d_h_src.pdfbs2d_std2d.h"
#include "
href../../common/enums_h_src.pdfenums.h"
#include "pnl/pnl_cdf.h"

static int Standard2DMC(double s1, double s2, NumFunc_2 *p, double t, double r,
{
    int i;
    double sigma11, sigma21, sigma22, sigma11_sqrt, sigma21_sqrt, sigma22_sqrt, f
    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 = 0., K2 = 0., ratio = 0.;

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

    /* Value to construct the confidence interval */
    alpha = (1. - confidence) / 2.;
    z_alpha = pnl_inv_cdfnor(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;
sigma21 = rho * sigma2;
sigma22 = sigma2 * sqrt(1.0 - SQR(rho));
sigma11_sqrt = sigma11 * 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 = pnl_rand_init(generator, simulation_dim, N);

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

    /* Begin N iterations */
    for (i = 1 ; i <= N ; i++)
    {
        /*Gaussian Random Variables*/
        g_1 = pnl_rand_gauss(simulation_dim, CREATE, 0, generator);
        g_2 = pnl_rand_gauss(simulation_dim, RETRIEVE, 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 + sigma22_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(*ptp
*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(*p
*inf_delta1 = *ptdelta1 - z_alpha * (*pterror_delta1);
*sup_delta1 = *ptdelta1 + z_alpha * (*pterror_delta1);

/* Delta2 estimator */
*ptdelta2 = exp(-r * t) * mean_delta2 / (double) N;
*pterror_delta2 = sqrt(exp(-2.0 * r * t) * var_delta2 / (double)N - SQR(*p
*inf_delta2 = *ptdelta2 - z_alpha * (*pterror_delta2);
*sup_delta2 = *ptdelta2 + z_alpha * (*pterror_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_ENUM.value,
        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)
    );
}

static 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, Option *Opt)
{
    int type_generator;
    if (Met->init == 0)
    {
        Met->init = 1;
    }
}

```

```

    Met->Par[0].Val.V_LONG = 10000;
    Met->Par[1].Val.V_ENUM.value = 0;
    Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;
    Met->Par[2].Val.V_DOUBLE = 0.95;

}

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

if (pnl_rand_or_quasi(type_generator) == PNL_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", ENUM, {100}, ALLOW},
    {"Confidence Value", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {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} ,
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_Standard2D),
  CHK_ok,
  MET(Init)
};

```