

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/bs2d/bs2d_std2d/bs2d_std2d_h_src.pdfbs2d_std2d.h"
#include "
href../../common/enums_h_src.pdfenums.h"

static int d = 2;
static int l = 1;
static long N_sim;
static double **X, * *W, * *Dw, * *ln, * *Z, *P, *Pn, *Qn, *Semi, *Obst, *P2, *
static double *drift, *diff_z;
static double *s, * *sigma, *divid, * *sigma_inv, * *sigma_transf;

static void memory_allocation()
{
    int i;

    sigma = (double **)calloc(d, sizeof(double *));
    for (i = 0; i < d; i++)
        sigma[i] = (double *)calloc(d, sizeof(double));

    sigma_inv = (double **)calloc(d, sizeof(double *));
    for (i = 0; i < d; i++)
        sigma_inv[i] = (double *)calloc(d, sizeof(double));

    sigma_transf = (double **)calloc(d, sizeof(double *));
    for (i = 0; i < d; i++)
        sigma_transf[i] = (double *)calloc(d, sizeof(double));

    s = malloc((d) * sizeof(double));
    divid = malloc((d) * sizeof(double));
    drift = malloc((d) * sizeof(double));
    diff_z = malloc((d) * sizeof(double));

    X = (double **)calloc(d, sizeof(double *));
    for (i = 0; i < d; i++)
        X[i] = (double *)calloc(N_sim, sizeof(double));

    W = (double **)calloc(d, sizeof(double *));
```

```

for (i = 0; i < d; i++)
    W[i] = (double *)calloc(N_sim, sizeof(double));

Dw = (double **)calloc(d, sizeof(double *));
for (i = 0; i < d; i++)
    Dw[i] = (double *)calloc(N_sim, sizeof(double));

ln = (double **)calloc(d, sizeof(double *));
for (i = 0; i < d; i++)
    ln[i] = (double *)calloc(N_sim, sizeof(double));

Z = (double **)calloc(d, sizeof(double *));
for (i = 0; i < d; i++)
    Z[i] = (double *)calloc(N_sim, sizeof(double));

Delta = (double **)calloc(d, sizeof(double *));
for (i = 0; i < d; i++)
    Delta[i] = (double *)calloc(N_sim, sizeof(double));

Pn = malloc((N_sim) * sizeof(double));
Qn = malloc((N_sim) * sizeof(double));
P = malloc((N_sim) * sizeof(double));
P2 = malloc((N_sim) * sizeof(double));
Semi = malloc((N_sim) * sizeof(double));
Obst = malloc((N_sim) * sizeof(double));

return;
}

/*Memory Desallocation*/
static void free_memory()
{
    int i;

    for (i = 0; i < d; i++)
        free(Delta[i]);
    free(Delta);

    for (i = 0; i < d; i++)
        free(sigma[i]);
    free(sigma);
}

```

```

for (i = 0; i < d; i++)
    free(sigma_inv[i]);
free(sigma_inv);

for (i = 0; i < d; i++)
    free(sigma_transf[i]);
free(sigma_transf);

for (i = 0; i < d; i++)
    free(X[i]);
free(X);

for (i = 0; i < d; i++)
    free(W[i]);
free(W);

for (i = 0; i < d; i++)
    free(Z[i]);
free(Z);

for (i = 0; i < d; i++)
    free(Dw[i]);
free(Dw);

for (i = 0; i < d; i++)
    free(ln[i]);
free(ln);

free(divid);
free(drift);
free(s);
free(diff_z);

free(Pn);
free(Qn);
free(P);
free(P2);
free(Semi);
free(Obst);

```

```

    return;
}

static double H(double x)
{
    double val;

    if (x >= 0.) val = 1.;
    else val = 0.;

    return val;
}

static double g1(double x, double lambda)
{
    double val;

    val = 0.5 * lambda * exp(-lambda * fabs(x));

    return val;
}

static double GH1(double x, double lambda)
{
    double val;

    if (x < 0.) val = 0.5 * exp(lambda * x);
    else val = 1 - 0.5 * exp(-lambda * x);

    return val;
}

static int LionsRegnier2DMC(double s1, double s2, NumFunc_2 *p, double t, double
{
    int simulation_dim = 1, /*fermeture=1,*/init_mc;
    int i, j, k, jz, TimeIndex, n, ii;
    double eps, sum, sum1, sum2, eps2, att, semi0, c_price, c_delta1, c_delta2, de
    double val, tmp1, tmp2;
    double prod1, prod2, lambda1[10], lambda2[10], K, prodT, prodT1, prodT_tot, pr

```

```

K = p->Par[0].Val.V_DOUBLE;
N_sim = N;
n = exercise_date_number;

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

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

    memory_allocation();
    eps = t / (double)n;
    eps2 = SQR(eps);
    att = exp(-r * eps);

    /* Covariance Matrix */
    /* Coefficients of the matrix A such that A(tA)=Gamma */
    sigma[0][0] = sigma1;
    sigma[0][1] = 0.0;
    sigma[1][0] = rho * sigma2;
    sigma[1][1] = sigma2 * sqrt(1.0 - SQR(rho));

    /*Sigma Transformed*/
    for (i = 0; i < d; i++)
        for (j = 0; j < d; j++)
            sigma_transf[i][j] = sigma[i][j] / sigma[j][j];

    /*Inverse of Sigma Transformed sub-triangular */
    for (i = 0; i < d; i++)
        sigma_inv[i][i] = 1.;

    for (j = 0; j < d; j++)
    {
        for (i = j + 1; i < d; i++)
        {
            sum = 0.;
            for (k = j; k < i; k++)
                sum += sigma_transf[i][k] * sigma_inv[k][j];
            sigma_inv[i][j] = -sum;
        }
    }
}

```

```

    }
}

/*Drift,Diffusion*/
s[0] = s1;
s[1] = s2;
divid[0] = divid1;
divid[1] = divid2;

for (i = 0; i < d; i++)
{
    sum1 = 0.;
    sum2 = 0.;
    for (j = 0; j <= i; j++)
    {
        sum1 += SQR(sigma[i][j]);
        sum2 += sigma[i][j];
    }
    drift[i] = (r - divid[i] - 0.5 * sum1) * eps;
    diff_z[i] = sqrt(eps) * sigma[i][i];
}

/*Brownian motion at the end*/
for (i = 0; i < d; i++)
    for (j = 0; j < N; j++)
        W[i][j] = pnl_rand_normal(generator) * sqrt(t);

/*Final Stock*/
for (i = 0; i < d; i++)
{
    for (j = 0; j < N; j++)
    {
        sum = 0.;
        for (k = 0; k <= i; k++)
        {
            sum += sigma[i][k] * W[k][j];
        }
        X[i][j] = s[i] * exp(drift[i] * (double)n + sum);
    }
}

```

```

/*Final Price*/
for (j = 0; j < N; j++)
    Pn[j] = 0.0;

/*Backward Cycle*/
for (TimeIndex = n - 1; TimeIndex > 0; TimeIndex--)
{
    tmp1 = (double)(TimeIndex) / (double)(TimeIndex + 1);
    tmp2 = sqrt(tmp1 * eps);

    /*X,ln,Z,DW*/
    for (i = 0; i < d; i++)
    {
        for (j = 0; j < N; j++)
        {
            sum = 0.;
            val = W[i][j];
            W[i][j] = W[i][j] * tmp1 + tmp2 * pnl_rand_normal(generator);
            for (k = 0; k <= i; k++)
            {
                sum += sigma[i][k] * W[k][j];
            }

            /*X*/
            X[i][j] = s[i] * exp(drift[i] * (double)TimeIndex + sum);

            /*ln*/
            if (l == 0)
                ln[i][j] = 0.;
            else
            {
                if (i == 0)
                    ln[0][j] = 0.;
                else
                {
                    sum1 = 0.;
                    for (k = 0; k < i; k++)
                        sum1 += sigma_inv[i][k] *
                            (drift[k] * eps - (1. / (eps * (double)(TimeIndex + 1)))) * W[k][j];
                    ln[i][j] = sum1;
                }
            }
        }
    }
}

```

```

    }
    /*Z*/
    Z[i][j] = s[i] * exp((double)TimeIndex * drift[i] + ln[i][j] *

    /*Dw*/
    Dw[i][j] = eps * W[i][j] - (val - W[i][j]) * ((double)TimeIndex
        + eps2 * (double)TimeIndex * sigma[i][i];
    }
}
for (i = 0; i < d; i++)
{
    lambda1[i] = 1. / sqrt(eps * (double)TimeIndex);
    lambda2[i] = 1. / sqrt(eps * (double)TimeIndex);
}

/*P,Semi*/
for (j = 0; j < N; j++)
{
    PutMinAn(X[0][j], X[1][j], K, t - (double)TimeIndex * eps, r, divi
    Obst[j] = (p->Compute)(p->Par, X[0][j], X[1][j]) - c_price;

    sum1 = 0.;
    sum2 = 0.;
    for (jz = 0; jz < N; jz++)
    {
        prod1 = 1.;
        prod2 = 1.;
        for (i = 0; i < d; i++)
        {
            prod1 *= g1(Z[i][jz] - Z[i][j], lambda1[i]) + (H(Z[i][jz]
            prod2 *= g1(Z[i][jz] - Z[i][j], lambda2[i]) + (H(Z[i][jz]
        }

        sum1 += prod1 * Pn[jz];
        sum2 += prod2;
    }
    Semi[j] = sum1 / sum2;

    /*Options Values*/
    P[j] = MAX(Obst[j], att * Semi[j]);

```



```

if (TimeIndex == 2)
{
    P2[j] = P[j];
}

if (TimeIndex == 1)
{
    if (P[j] == Obst[j])
    {
        PutMinAn(X[0][j], X[1][j], K, t - (double)TimeIndex * eps,

        if ((Z[0][j] < Z[1][j]))
        {
            Delta[0][j] = -H(K - MIN(Z[0][j], Z[1][j])) - c_delta1;
            Delta[1][j] = -c_delta2;
        }
        else if ((Z[1][j] < Z[0][j]))
        {
            Delta[1][j] = -H(MIN(K - Z[0][j], Z[1][j])) - c_delta2;
            Delta[0][j] = -c_delta1;
        }

    }
    else
    {
        for (i = 0; i < d; i++)
        {
            lambdaT1 = 1. / sqrt(eps * (double)TimeIndex);
            lambdaT = 1. / sqrt(eps * (double)TimeIndex);
            lambdaR1 = 1. / sqrt(eps * (double)TimeIndex);
            lambdaR = 1. / sqrt(eps * (double)TimeIndex);

            sumT = 0.;
            sumT1 = 0.;
            sumTD = 0.;
            sumR = 0.;
            sumR1 = 0.;
            for (jz = 0; jz < N; jz++)
            {

```

```

        prodT_tot = 1.;
        prodT1_tot = 1.;
        for (ii = 0; ii < d; ii++)
        {
            prodT_tot *= g1(Z[ii][jz] - Z[ii][j], lambdaT)
            prodT1_tot *= g1(Z[ii][jz] - Z[ii][j], lambdaT1)
        }

        prodT = g1(Z[i][jz] - Z[i][j], lambdaT) + (H(Z[i][jz] - Z[i][j], lambdaT) / (2 * lambdaT));
        prodT1 = g1(Z[i][jz] - Z[i][j], lambdaT1) + (H(Z[i][jz] - Z[i][j], lambdaT1) / (2 * lambdaT1));
        prodR = -g1(Z[i][jz] - Z[i][j], lambdaR) * (Dw[i][jz] - Dw[i][j]);
        prodR1 = -g1(Z[i][jz] - Z[i][j], lambdaR1) * (Dw[i][jz] - Dw[i][j]);

        sumT += prodT_tot * P2[jz];
        sumT1 += prodT1_tot * P2[jz];

        sumR += prodR * (prodT_tot / prodT) * P2[jz];
        sumR1 += prodR1 * (prodT1_tot / prodT1) * P2[jz];
        sumTD += prodT1_tot;

    }
    Delta[i][j] = att * (sumR * sumT1 - sumT * sumR1) / (sumT * sumT1 - sumTD * sumTD);
}
}
}

for (j = 0; j < N; j++)
    Pn[j] = P[j];
}

/*Final Step*/
PutMinAn(s[0], s[1], K, t, r, divid[0], divid[1], sigma1, sigma2, rho, &c_);
sum = 0.;
for (jz = 0; jz < N; jz++)
    sum += Pn[jz];

semi0 = sum / (double)N;

```

```

    sum = 0.;
    sum1 = 0.;
    for (jz = 0; jz < N; jz++)
    {
        sum += Delta[0][jz];
        sum1 += Delta[1][jz];
    }

    delta1 = sum / (double)N + c_delta1;
    delta2 = sum1 / (double)N + c_delta2;

    *ptprice = MAX((p->Compute)(p->Par, s1, s2) - c_price, att * semi0) + c_pr
    *ptdelta1 = delta1;
    *ptdelta2 = delta2;
}

free_memory();
return init_mc;
}

int CALC(MC_LionsRegnier2D)(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 LionsRegnier2DMC(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_INT,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE),
        &(Met->Res[2].Val.V_DOUBLE));
    }

static int CHK_OPT(MC_LionsRegnier2D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PutMinimumAmer") == 0))
        return OK;

    return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_LONG = 1000;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[2].Val.V_INT = 10;

    }
    return OK;
}

PricingMethod MET(MC_LionsRegnier2D) =
{
    "MC_LionsRegnier2d",
    { {"N iterations", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Number of Exercise Dates ", INT, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },

```

```

CALC(MC_LionsRegnier2D),
{ {"Price", DOUBLE, {100}, FORBID},
  {"Delta1", DOUBLE, {100}, FORBID} ,
  {"Delta2", DOUBLE, {100}, FORBID},
  {" " , PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(MC_LionsRegnier2D),
CHK_mc,
MET(Init)
};

```