

[Help](#)

```
/*
 * File written by Jérôme Lelong <jerome.lelong@gmail.com>
 * for Premia release 11
 * February 2009
 */

#include <stdlib.h>
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_vector.h"

#include "
href../../../../mod/bsnd/bsnd_stdnd/bsnd_stdnd_h_src.pdfbsnd_stdnd.h"
#include "
href../../../../common/enums_h_src.pdfenums.h"
#include "
href../../../../common/math/bsnd_math/bsnd_path_h_src.pdfmath/bsnd_math/bsnd_path.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2018+2) //The "#els
static int CHK_OPT(MC_ML_LocalBermuda)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_ML_LocalBermuda)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/**
 * Characteristics of a product
 */
typedef struct _Product Product;
struct _Product
{
    double r; /*!< interest rate */
    PnlVect *spot; /*!< spot */
}
```

```

double T; /*!< maturity */
PnlVect *sigma; /*!< volatility */
PnlVect *divid; /*!< dividende */
double kappa; /*!< strike */
double corr; /*!< correlation */
int dim; /*!< dimension de l'actif */
NumFunc_nd *payoff;
};

/**
 * Construit la factorisée de Cholesky (triangulaire inférieure) de la
 * matrice de corrélation (avec des corr partout et une diagonale de 1
 *
 * @param correl (out) factorisée de Cholesky de la matrice de corrélation
 * @param d dimension du modèle
 * @param corr corrélation entre les composantes
 */
static void init_correl(PnlMat **correl, const Product *P)
{
    *correl = pnl_mat_create_from_double(P->dim, P->dim, P->corr);
    pnl_mat_set_diag(*correl, 1, 0.);
    pnl_mat_chol((*correl));
}

//renvoie une matrice Z contenant l'asset de taille N*(J+1)
//N : nb de tirages MC
//J : nb de points de discrétisation

static void simul_asset(PnlMat **Z, int N, int J, const Product *P, PnlMat *corr)
{
    double h = P->T / J;
    int i, j, d;
    PnlMat **G;
    double Zj_1, sig, divid;
    PnlVect *Gij_1;

    G = malloc((P->dim) * sizeof(PnlMat *));
    for (d = 0; d < P->dim; d++)
    {
        G[d] = pnl_mat_create_from_double(N, J, 0);
        pnl_mat_rng_normal(G[d], N, J, rng);
    }
}

```

```

    }
    Gij_1 = pnl_vect_create_from_double(P->dim, 0);
    for (i = 0; i < N; i++)
    {
        for (d = 0; d < P->dim; d++)
        {
            pnl_mat_set(Z[d], i, 0, pnl_vect_get(P->spot, d));
        }
    }
    for (j = 1; j < J + 1; j++)
    {
        for (i = 0; i < N; i++)
        {
            for (d = 0; d < P->dim; d++)
            {
                Zj_1 = pnl_mat_get(Z[d], i, j - 1);
                sig = GET(P->sigma, d);
                divd = GET(P->divid, d);
                PnlVect Ld = pnl_vect_wrap_mat_row(correl, d);
                pnl_vect_set(Gij_1, d, pnl_mat_get(G[d], i, j - 1));
                pnl_mat_set(Z[d], i, j, Zj_1 * exp((P->r - divd - 0.5 * sig * sig)
            }
        }
    }
    pnl_vect_free(&Gij_1);
    for (d = 0; d < P->dim; d++)
    {
        pnl_mat_free(&G[d]);
    }
    free(G);
}

//définition du payoff
static double g(double t, PnlVect *x, const Product *P)
{
    NumFunc_nd *payoff = P->payoff;
    return exp(-P->r * t) * payoff->Compute(payoff->Par, x);
}

//vecteurs Kg et Ng définis page 11 du papier
static void vecteurs_k_et_n(PnlVectInt *Kg, PnlVectInt *Ng, int L, int k0)

```

```

{
    int i;
    double S = 0;
    double eps0 = 0.6;
    pnl_vect_int_resize(Kg, L + 1);
    pnl_vect_int_resize(Ng, L + 1);
    if (L == 0)
    {
        pnl_vect_int_set(Kg, 0, (int)pow(1.2 / eps0, 6));
        pnl_vect_int_set(Ng, 0, (int)pow(1.2 / eps0, 2));
    }
    else
    {
        for (i = 0; i < L + 1; i++)
        {
            pnl_vect_int_set(Kg, i, k0 * pow(2, i));
            if (i > 0) S = S + pow(pnl_vect_int_get(Kg, i), 11.0 / 24);
        }
        for (i = 0; i < L + 1; i++)
        {
            pnl_vect_int_set(Ng, i, (int)10.0 * pow(2, L / 3.0) * pow(k0, 1.0 / 3)
        }
    }
}

//fonction indicatrice servant à définir  $w^k_{ij}$  (page 13 du papier)
static double ind(PnlVect *z, PnlVect *Z, double delta)
{
    pnl_vect_minus_vect(z, Z);
    if (pnl_vect_norm_two(z) <= delta)
        return 1.0;
    else
        return 0;
}

static double fxi(int j, PnlVect *Z, double c, int J, const Product *P)
{
    double h = P->T / J;
    return fmax(g(j * h, Z, P), c);
}

```

```

//vecteur de k composantes (calculé à j fixé et l fixé, où l représente la l
//ème trajectoire MC, xi étant calculé en  $z=Z^l_j$ ). défini page 6 du papier
//comme une fonction de z
static void vecteur_xi(PnlVect *xi, PnlMat **Z, PnlMat *C, int j, int k, int l,
{
    int m, d;
    PnlVect *Zmj;
    Zmj = pnl_vect_create_from_double(P->dim, 0);
    pnl_vect_resize(xi, k);
    for (m = 0; m < k; m++)
    {
        for (d = 0; d < P->dim; d++)
        {
            pnl_vect_set(Zmj, d, pnl_mat_get(Z[d], m, j));
        }
        pnl_vect_set(xi, m, fxi(j, Zmj, pnl_mat_get(C, m, l), J, P));
    }
}

```

```

//vecteur de k composantes, calculé à j fixé, i fixé, où i est la i ème
//trajectoire MC.  $w_{ij}^k$  est calculé au point  $z=Z^i_j$  (voir définition page
//6 du papier)
static void vecteur_w(PnlVect *w, PnlMat **Z, int i, int j, int k, const Product
{
    int l;
    int m, d;
    PnlVect *Zij, *Zmj, *Zlj;
    double S;
    double delta_k = 100.0 * pow(k, -1.0 / (P->dim + 2));
    Zij = pnl_vect_create_from_double(P->dim, 0);
    Zmj = pnl_vect_create_from_double(P->dim, 0);
    Zlj = pnl_vect_create_from_double(P->dim, 0);
    pnl_vect_resize(w, k);

    for (d = 0; d < P->dim; d++)
    {
        pnl_vect_set(Zij, d, pnl_mat_get(Z[d], i, j));
    }
    S = 0;
    for (m = 0; m < k; m++)
    {

```

```

    for (d = 0; d < P->dim; d++)
    {
        pnl_vect_set(Zmj, d, pnl_mat_get(Z[d], m, j));
    }
    S = S + ind(Zmj, Zij, delta_k);
}
S = S / k;
for (l = 0; l < k; l++)
{
    for (d = 0; d < P->dim; d++)
    {
        pnl_vect_set(Zlj, d, pnl_mat_get(Z[d], l, j));
    }
    pnl_vect_set(w, l, ind(Zlj, Zij, delta_k) / S);
}
pnl_vect_free(&Zmj);
pnl_vect_free(&Zij);
pnl_vect_free(&Zlj);
}

```

//vecteur de taille J de matrices de taille (Kg(L),L+1), définie page 5 du
//papier. Ici évaluée en $z=Z^i_j$. Pour chaque point de la grille j, on calcule
//la matrice C. Les lignes de la matrice représentent le nb de tirages MC. Les
//colonnes de la matrice représentent les L+1 valeurs prises par Kg

```

static void matrice_C(PnlMat **C, PnlMat **Z, PnlVectInt *Kg, PnlVectInt *Ng, in
{
    int j, i, l;
    PnlVect *xi;
    PnlVect *w;
    double S;
    xi = pnl_vect_create_from_double(pnl_vect_int_get(Kg, L), 0);
    w = pnl_vect_create_from_double(pnl_vect_int_get(Kg, L), 0);
    for (j = J - 1; j >= 0; j--)
    {
        for (l = 0; l < L + 1; l++)
        {
            vecteur_xi(xi, Z, C[j + 1], j + 1, pnl_vect_int_get(Kg, l), l, J, P);
            for (i = 0; i < pnl_vect_int_get(Ng, l); i++)
            {
                vecteur_w(w, Z, i, j, pnl_vect_int_get(Kg, l), P);
                S = pnl_vect_scalar_prod(w, xi);
            }
        }
    }
}

```

```

        S = S / pnl_vect_int_get(Kg, l);
        pnl_mat_set(C[j], i, l, S);
    }
}
}
pnl_vect_free(&xi);
pnl_vect_free(&w);
}

```

//fonction renvoyant le 1er instant où l'on doit exercer (défini page 8 du papier)

```

static double tau_f(PnlMat **Z, PnlMat **C, int rr, int k, int J, const Product
{

```

```

    int j, d;
    double h = P->T / J;
    j = 0;
    PnlVect *Zrrj;
    Zrrj = pnl_vect_create_from_double(P->dim, 0);
    for (d = 0; d < P->dim; d++)
    {
        pnl_vect_set(Zrrj, d, pnl_mat_get(Z[d], rr, j));
    }
    while (g(j * h, Zrrj, P) < pnl_mat_get(C[j], rr, k) && (j < J + 1))
    {
        j++;
        for (d = 0; d < P->dim; d++)
        {
            pnl_vect_set(Zrrj, d, pnl_mat_get(Z[d], rr, j));
        }
    }
    pnl_vect_free(&Zrrj);
    return j;
}

```

//calcul de $V_0^{n,k}$ défini page 8 du papier

```

static double prix(PnlMat **Z, PnlMat **C, PnlVectInt *Kg, PnlVectInt *Ng, int L
{
    PnlMat *tau;
    PnlVect *Zrt;
    PnlVect *Zrt_1;
    int l, rr, d;
    double S, Sl, tau_k0, n0, nl, tau_kl, tau_kl_1;

```

```

double h = P->T / J;
Zrt = pnl_vect_create_from_double(P->dim, 0);
Zrt_1 = pnl_vect_create_from_double(P->dim, 0);
S = 0;
tau = pnl_mat_create_from_double(pnl_vect_int_get(Ng, 0), L + 1, 0);
for (l = 0; l < L + 1; l++)
{
    for (rr = 0; rr < pnl_vect_int_get(Ng, l); rr++)
    {
        pnl_mat_set(tau, rr, l, tau_f(Z, C, rr, l, J, P));
    }
}
n0 = pnl_vect_int_get(Ng, 0);
for (rr = 0; rr < n0; rr++)
{
    tau_k0 = pnl_mat_get(tau, rr, 0);
    for (d = 0; d < P->dim; d++)
    {
        pnl_vect_set(Zrt, d, pnl_mat_get(Z[d], rr, (int)tau_k0));
    }
    S = S + g(tau_k0 * h, Zrt, P);
}
S = S / n0;
for (l = 1; l < L + 1; l++)
{
    S1 = 0;
    n1 = pnl_vect_int_get(Ng, l);
    for (rr = 0; rr < n1; rr++)
    {
        tau_kl = pnl_mat_get(tau, rr, l);
        tau_kl_1 = pnl_mat_get(tau, rr, l - 1);
        for (d = 0; d < P->dim; d++)
        {
            pnl_vect_set(Zrt, d, pnl_mat_get(Z[d], rr, (int)tau_kl));
            pnl_vect_set(Zrt_1, d, pnl_mat_get(Z[d], rr, (int)tau_kl_1));
        }
        S1 = S1 + g(tau_kl * h, Zrt, P) - g(tau_kl_1 * h, Zrt_1, P);
    }
    S = S + S1 / n1;
}
pnl_mat_free(&tau);

```



```

    pnl_vect_free(&Zrt);
    pnl_vect_free(&Zrt_1);
    return S;
}

/**
 * @brief
 *
 * @param spot
 * @param sigma
 * @param rho
 * @param divid
 * @param r
 * @param T
 * @param J number of exercising dates
 * @param L number of levels
 * @param gen
 * @param payoff
 * @param price
 * @return int
 */
static void mcmlmesh(PnlVect *spot, PnlVect *sigma, double rho, PnlVect *divid,
{
    PnlVectInt *Kg;
    PnlVectInt *Ng;
    PnlVect *w;
    Product P;

    //product parameters
    P.r = r; //taux d'interet
    P.T = T; //maturité
    P.dim = spot->size;
    P.spot = spot;
    P.sigma = sigma;
    P.divid = divid;
    P.corr = rho;
    P.payoff = payoff;

    //parameters of the method
    int k0 = 100; //paramètres définissant le nb de tirages MC, donné page 11

```

```

PnlMat **C = malloc((J + 1) * sizeof(PnlMat *));
PnlMat **Z = malloc((P.dim) * sizeof(PnlMat *));
int j = 0;
int d = 0;

PnlRng *rng = pnl_rng_create(gen);
pnl_rng_sseed(rng, 0);
PnlMat *correl;
init_correl(&correl, &P);

Kg = pnl_vect_int_new();
Ng = pnl_vect_int_new();
vecteurs_k_et_n(Kg, Ng, L, k0);
for (j = 0; j < J + 1; j++) C[j] = pnl_mat_create_from_double(fmax(pnl_vect_in
for (j = 0; j < P.dim; j++) Z[j] = pnl_mat_create_from_double(fmax(pnl_vect_in

w = pnl_vect_new();
simul_asset(Z, fmax(pnl_vect_int_get(Kg, L), pnl_vect_int_get(Ng, 0)), J, &P,

matrice_C(C, Z, Kg, Ng, J, L, &P);

*price = prix(Z, C, Kg, Ng, L, J, &P);

/* Free memory */
for (d = 0; d < P.dim; d++)
{
    pnl_mat_free(&Z[d]);
}
for (j = 0; j < J + 1; j++)
{
    pnl_mat_free(&C[j]);
}

pnl_vect_free(&w);
pnl_vect_int_free(&Kg);
pnl_vect_int_free(&Ng);
pnl_rng_free(&rng);
pnl_mat_free(&correl);
}

```

```

int CALC(MC_ML_LocalBermuda)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r;
    int i, size;
    PnlVect *spot, *sig, *divid;
    double price;

    size = ptMod->Size.Val.V_PINT;
    divid = pnl_vect_create(size);
    spot = ptMod->S0.Val.V_PNLVECT;
    sig = ptMod->Sigma.Val.V_PNLVECT;

    for (i = 0; i < size; i++)
        pnl_vect_set(divid, i, log(1. + GET(ptMod->Divid.Val.V_PNLVECT, i) / 100.));

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

    mcmllmesh(spot, sig, ptMod->Rho.Val.V_DOUBLE, divid, r,
               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
               Met->Par[1].Val.V_PINT,
               Met->Par[2].Val.V_PINT,
               Met->Par[0].Val.V_ENUM.value,
               ptOpt->PayOff.Val.V_NUMFUNC_ND,
               &price);

    Met->Res[0].Val.V_DOUBLE = price;

    pnl_vect_free(&divid);
    return OK;
}

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

    if ((opt->EuOrAm).Val.V_BOOL == AMER)
        return OK;
    return WRONG;
}

```

```

}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_ENUM.value = 0;
        Met->Par[0].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[1].Val.V_PINT = 10;
        Met->Par[2].Val.V_PINT = 4;
    }
    return OK;
}

PricingMethod MET(MC_ML_LocalBermuda) =
{
    "MC_ML_LocalBermuda",
    {
        {"RandomGenerator", ENUM, {0}, ALLOW},
        {"Nb dates", PINT, {10000}, ALLOW},
        {"Nb levels", PINT, {10000}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_ML_LocalBermuda),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_ML_LocalBermuda),
    CHK_ok,
    MET(Init)
};

```