

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/hes1d_slv/hes1d_slv_std/hes1d_slv_std_h_src.pdfhes1d_slv_std.h"
#include "
href../../common/enums_h_src.pdfenums.h"

#include "pnl/pnl_matrix.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_finance.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(MC_GuyonLabordere)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_GuyonLabordere)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/**
 * Characteristics of a product
 */
typedef struct _Product  Product;
struct _Product
{
    double divid;
    double r; /*!< interest rate */
    double s0; /*!< spot */
    double v0; /*!< volatilité initiale */
    double kappa; /*!< parametre modele heston */
    double gamma; /*!< parametre modele heston */
    double vbarre; /*!< parametre modele heston */
    double rho; /*!< parametre modele heston */
    double T;
```

```

};

static double psi(double x)
{
    return sqrt(MAX(x, 0));
}

static double psi_carre(double x)
{
    return pow(psi(x), 2);
}

static double av(double t, double x, const Product *P)
{
    return P->kappa * (P->vbarre - x);
}

static double bv(double t, double x, const Product *P)
{
    return P->gamma * sqrt(MAX(x, 0));
}

//Algo 1, page 8 du papier
//input L: nb de bins = L+1 //Attention L doit diviser M-1
//input Si : vecteur de taille M représentant M valeurs de S(t_i)
//input Vi : vecteur de taille M représentant M valeurs de V(t_i)
//output Bi : vecteur de taille L+1 contenant le bord des intervalles
//output Ei : vecteur de taille L contenant la valeur de l'esp. Cond. sur
//chaque intervalle

static void Algo1(PnlVect *Bi, PnlVect *Ei, PnlVect *Si, PnlVect *Vi, int L)
{
    int l, k;
    double S;
    int M = Si->size;
    int size_bin = floor((double)(M - 1) / L);
    PnlVectInt *index = pnl_vect_int_create(M);
    pnl_vect_map_inplace(Vi, psi_carre);
    pnl_vect_qsort_index(Si, index, 'i');
    pnl_vect_permute_inplace(Vi, index); //on range Vi comme Si
    pnl_vect_set(Bi, 0, pnl_vect_get(Si, 0));
}

```

```

pnl_vect_set(Bi, L, pnl_vect_get(Si, M - 1));

for (l = 1; l < L; l++)
{
    pnl_vect_set(Bi, l, pnl_vect_get(Si, l * size_bin));
}
for (l = 0; l < L; l++)
{
    S = 0;
    for (k = 1; k < size_bin + 1; k++)
    {
        S = S + pnl_vect_get(Vi, l * size_bin + k);
    }
    pnl_vect_set(Ei, l, (double)L / M * S);
}
pnl_vect_int_free(&index);
}

```

```

//conditional expectation
//input : matrices de gaussiennes Z1 et Z2
//output : matrice B donnant les bins de taille (L+1)*M
//output : matrice E donnant les valeurs de l'esp. cond. de taille L*M
static double algo_particle(NumFunc_1 *p, int sig_t, PnlMat *S, PnlMat *V, PnlMat *B, PnlMat *E)
{
    int N = Z1->n; //nb de pas de discretisation
    int M = Z1->m; //nb de tirages MC
    PnlVect *Bj = pnl_vect_create_from_double(L + 1, 0);
    PnlVect *Ej = pnl_vect_create_from_double(L, 0);
    PnlVect *Vj = pnl_vect_create_from_double(M, 0);
    PnlVect *Sj = pnl_vect_create_from_double(M, 0);
    double delta = P->T / N;
    double vij, sij, EC;
    int i, j;
    double s = 0;
    double prix;

    for (j = 0; j < N; j++) //tous les pas de temps
    {
        pnl_mat_get_col(Vj, V, j);
    }
}

```

```

    pnl_mat_get_col(Sj, S, j);

    Algo1(Bj, Ej, Sj, Vj, L); //Sj et Vj ressortent triés

    for (i = 0; i < M; i++) //tous les tirages MC
    {
        vij = pnl_mat_get(V, i, j);
        sij = pnl_mat_get(S, i, j);

        if (i == 0) EC = pnl_vect_get(Ej, 0);
        else EC = pnl_vect_get(Ej, floor((double)(i - 1) * L / (M - 1)));
        pnl_mat_set(V, i, j + 1, vij + av(j * delta, vij, P)*delta + bv(j * de
        pnl_mat_set(S, i, j + 1, sij + (P->r - P->divid)*delta * sij + premia_
    }
}
for (i = 0; i < M; i++)
{
    s += p->Compute(p->Par, MGET(S, i, N));
}
prix = exp(-P->r * P->T) * s / M;

pnl_vect_free(&Bj);
pnl_vect_free(&Ej);
pnl_vect_free(&Vj);
pnl_vect_free(&Sj);

return prix;
}

```

```

static double MCGuyonLabordere(const Product *P, PnlMat *S, PnlMat *V, PnlMat *Z
{
    // M must be a multiple of bins
    div_t d = div(M, bins);
    if (d.rem != 0)
    {
        M = (d.quot + 1) * bins;
    }
    M++;
}

```

```

    pnl_mat_resize(Z1, M, N);
    pnl_mat_set_zero(Z1);
    pnl_mat_resize(Z2, M, N);
    pnl_mat_set_zero(Z2);
    pnl_mat_resize(V, M, N + 1);
    pnl_mat_set_all(V, P->v0);
    pnl_mat_resize(S, M, N + 1);
    pnl_mat_set_all(S, P->s0);
    pnl_mat_rng_normal(Z1, M, N, rng);
    pnl_mat_rng_normal(Z2, M, N, rng);

    return algo_particle(p, sig_t, S, V, Z1, Z2, P, bins);
}

int CALC(MC_GuyonLabordere)(void *Opt, void *Mod, PricingMethod *Met)
{
    Product P;
    PnlMat *Z1, *Z2, *V, *S;
    PnlRng *rng;
    double prix_up, prix_down, h;
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    P.r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    P.divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    P.s0 = ptMod->S0.Val.V_PDOUBLE;
    P.T = ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE;
    P.v0 = ptMod->Sigma0.Val.V_PDOUBLE;
    P.rho = ptMod->Rho.Val.V_PDOUBLE;
    P.kappa = ptMod->MeanReversion.Val.V_PDOUBLE;
    P.gamma = ptMod->Sigma.Val.V_PDOUBLE;
    P.vbarre = ptMod->LongRunVariance.Val.V_PDOUBLE;
    Z1 = pnl_mat_new();
    Z2 = pnl_mat_new();
    V = pnl_mat_new();
    S = pnl_mat_new();
    rng = PnlRngArray[Met->Par[2].Val.V_ENUM.value];
    pnl_rng_sseed(rng, 0);

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

```

```

    MCGuyonLabordere(&P, S, V, Z1, Z2,
                      ptOpt->PayOff.Val.V_NUMFUNC_1,
                      ptMod->SigmaLV.Val.V_ENUM.value,
                      Met->Par[0].Val.V_LONG,
                      Met->Par[1].Val.V_INT,
                      rng,
                      Met->Par[3].Val.V_INT);

h = 0.1;
P.s0 = ptMod->S0.Val.V_PDOUBLE * (1 + h);
prix_up =
    MCGuyonLabordere(&P, S, V, Z1, Z2,
                      ptOpt->PayOff.Val.V_NUMFUNC_1,
                      ptMod->SigmaLV.Val.V_ENUM.value,
                      Met->Par[0].Val.V_LONG,
                      Met->Par[1].Val.V_INT,
                      rng,
                      Met->Par[3].Val.V_INT);

P.s0 = ptMod->S0.Val.V_PDOUBLE * (1 - h);
prix_down =
    MCGuyonLabordere(&P, S, V, Z1, Z2,
                      ptOpt->PayOff.Val.V_NUMFUNC_1,
                      ptMod->SigmaLV.Val.V_ENUM.value,
                      Met->Par[0].Val.V_LONG,
                      Met->Par[1].Val.V_INT,
                      rng,
                      Met->Par[3].Val.V_INT);
Met->Res[1].Val.V_DOUBLE = (prix_up - prix_down) / (2 * h * ptMod->S0.Val.V_PD

pnl_mat_free(&Z1);
pnl_mat_free(&Z2);
pnl_mat_free(&V);
pnl_mat_free(&S);

return OK;
}

```

```

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

    return WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    //int type_generator;
    if (Met->init == 0)
    {
        Met->init = 1;

        Met->Par[0].Val.V_LONG = 5000;
        Met->Par[1].Val.V_INT = 100;
        Met->Par[2].Val.V_ENUM.value = 0;
        Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[3].Val.V_INT = 20;
    }

    return OK;
}

PricingMethod MET(MC_GuyonLabordere) =
{
    "MC_GuyonLabordere",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"N bins", INT, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_GuyonLabordere),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    }
}

```

```
},  
  CHK_OPT(MC_GuyonLabordere),  
  CHK_mc,  
  MET(Init)  
};
```