

## Help

```
#include "lmm1d_std.h"
#include "pnl/pnl_basis.h"
#include "math/mc_lmm_glassermanzhao.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
static int CHK_OPT(MC_LongstaffSchwartz_BermudanSwaption)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_LongstaffSchwartz_BermudanSwaption)(void *Opt, void *Mod, PricingMet
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Price of bermudan swaption using Longstaff-Schwartz algorithm
 * @param LS_Price price by Longstaff-Schwartz algorithm on exit.
 * @param Nominal nominal of swaption
 * @param NbrMCsimulation the number of samples
 * @param ptLib Libor structure contains initial value of libor rates
 * @param ptBermSwpt Swaption structure contains bermudan swaption information
 * @param ptVol Volatility structure contains libor volatility deterministic fun
 * @param generator the index of the random generator to be used
 * @param basis_name regression basis
 * @param DimApprox dimension of regression basis
 * @param NbrStepPerTenor number of steps of discretization between T(i) and T(i
 * @param flag_numeraire measure under wich simulation is done.
 * flag_numeraire=0->Terminal measure, flag_numeraire=1->Spot measure
 * Rmk: Libor rates are simulated using the method proposed by Glasserman-Zhao.
 */
static void MC_BermSwaption_LongstaffSchwartz(double *LS_Price, double Nominal,
{
    int alpha, beta, m, k, N, NbrExerciseDates, time_index, save_brownian, save_al
    double tenor, regressed_value, payoff, numeraire_0;
    double *VariablesExplicatives;

    Libor *ptL_current;
    Swaption *ptSwpt;
```

```

PnlMat *LiborPathsMatrix, *BrownianMatrixPaths;
PnlMat *ExplicativeVariables;
PnlVect *OptimalPayoff;
PnlVect *RegressionCoeffVect;
PnlBasis *basis;

//Nfac = ptVol->numberOfFactors;
N = ptLib->numberOfMaturities;
tenor = ptBermSwpt->tenor;
alpha = (int)(ptBermSwpt->swaptionMaturity / tenor); // T(alpha) is the swapti
beta = (int)(ptBermSwpt->swapMaturity / tenor); // T(beta) is the swap maturi
NbrExerciseDates = beta - alpha;
start_index = 0;
end_index = beta - 1;
Nsteps = end_index - start_index;

save_brownian = 0;
save_all_paths = 1;
nbr_var_explicatives = 2;

VariablesExplicatives = malloc(nbr_var_explicatives * sizeof(double));
ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
OptimalPayoff = pnl_vect_create(NbrMCsimulation);
RegressionCoeffVect = pnl_vect_new();
LiborPathsMatrix = pnl_mat_new(); // LiborPathsMatrix contains all the traject
BrownianMatrixPaths = pnl_mat_new(); // We store also the brownian values to b

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

mallocLibor(&ptL_current, N, tenor, 0.1);

// ptSwpt := contains the information about the swap to be be exerced at each
// The maturity of the swap stays the same.
mallocSwaption(&ptSwpt, ptBermSwpt->swaptionMaturity, ptBermSwpt->swapMaturity

numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

// Simulation the "NbrMCsimulation" paths of Libor rates. We also store browni
Sim_Libor_Glasserman(start_index, end_index, ptLib, ptVol, generator, NbrMCsim

ptSwpt->swaptionMaturity = ptBermSwpt->swapMaturity - tenor; // Last exerice d

```

```

time_index = end_index;

// At the last exercise date, price of the option = payoff.
for (m = 0; m < NbrMCsimulation; m++)
{
    pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m * Nst
    LET(OptimalPayoff, m) = Swaption_Payoff_Discounted(ptL_current, ptSwpt, p,
}

for (k = NbrExerciseDates - 1; k >= 1; k--)
{
    ptSwpt->swaptionMaturity -= tenor; // k'th exercise date
    time_index -= 1;

    for (m = 0; m < NbrMCsimulation; m++)
    {
        pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m *
        MLET(ExplicativeVariables, m, 0) = computeSwapRate(ptL_current, time_i
        MLET(ExplicativeVariables, m, 1) = GET(ptL_current->libor, time_index)
    }
    // Least square fitting
    pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, Optimal

    // Equation de programmation dynamique.
    for (m = 0; m < NbrMCsimulation; m++)
    {
        pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, time_index + m *
        payoff = Swaption_Payoff_Discounted(ptL_current, ptSwpt, p, flag_numer

        // If the payoff is null, the OptimalPayoff doesn't change.
        if (payoff > 0)
        {
            VariablesExplicatives[0] = computeSwapRate(ptL_current, time_index
            VariablesExplicatives[1] = GET(ptL_current->libor, time_index);

            regressed_value = pnl_basis_eval(basis, RegressionCoeffVect, Varia

            if (payoff > regressed_value)
            {
                LET(OptimalPayoff, m) = payoff;
            }
        }
    }
}

```

```

    }
}

// The price at date 0 is the conditional expectation of OptimalPayoff, ie it'
*LS_Price = pnl_vect_sum(OptimalPayoff) / NbrMCsimulation;

*LS_Price *= (double)(numeraire_0 * Nominal);

free(VariablesExplicatives);
pnl_basis_free(&basis);
pnl_mat_free(&LiborPathsMatrix);
pnl_mat_free(&ExplicativeVariables);

pnl_vect_free(&OptimalPayoff);
pnl_vect_free(&RegressionCoeffVect);
pnl_mat_free(&BrownianMatrixPaths);

freeSwaption(&ptSwpt);
freeLibor(&ptL_current);
}

static int MCLongstaffSchwartz(NumFunc_1 *p, double l0, double sigma_const, int
{
    Volatility *ptVol;
    Libor *ptLib;
    Swaption *ptBermSwpt;
    int init_mc;
    int Nbr_Maturities;

    Nbr_Maturities = (int)(swap_maturity / tenor);

    mallocLibor(&ptLib , Nbr_Maturities, tenor, l0);
    mallocVolatility(&ptVol , nb_factors, sigma_const);
    mallocSwaption(&ptBermSwpt, swaption_maturity, swap_maturity, 0.0, swaption_st

    init_mc = pnl_rand_init(generator, nb_factors, nb_MC);
    if (init_mc != OK) return init_mc;

    MC_BermSwaption_LongstaffSchwartz(swaption_price, Nominal, nb_MC, p, ptLib, pt

```

```

    freeLibor(&ptLib);
    freeVolatility(&ptVol);
    freeSwaption(&ptBermSwpt);

    return init_mc;
}

int CALC(MC_LongstaffSchwartz_BermudanSwaption)(void *Opt, void *Mod, PricingMet
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return MCLongstaffSchwartz(ptOpt->PayOff.Val.V_NUMFUNC_1,
                                ptMod->l0.Val.V_PDOUBLE,
                                ptMod->Sigma.Val.V_PDOUBLE,
                                ptMod->NbFactors.Val.V_ENUM.value,
                                ptOpt->BMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                ptOpt->OMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                ptOpt->Nominal.Val.V_PDOUBLE,
                                ptOpt->FixedRate.Val.V_PDOUBLE,
                                ptOpt->ResetPeriod.Val.V_DATE,
                                Met->Par[0].Val.V_LONG,
                                Met->Par[1].Val.V_ENUM.value,
                                Met->Par[2].Val.V_ENUM.value,
                                Met->Par[3].Val.V_INT,
                                Met->Par[4].Val.V_INT,
                                Met->Par[5].Val.V_ENUM.value,
                                &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(MC_LongstaffSchwartz_BermudanSwaption)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerBermudanSwaption") == 0) || (strcmp((
        return OK;
    else
        return WRONG;
}

#endif //PremiaCurrentVersion

```

```

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

        Met->Par[0].Val.V_LONG = 50000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;
        Met->Par[2].Val.V_ENUM.value = 0;
        Met->Par[2].Val.V_ENUM.members = &PremiaEnumBasis;
        Met->Par[3].Val.V_INT = 10;
        Met->Par[4].Val.V_INT = 1;
        Met->Par[5].Val.V_ENUM.value = 0;
        Met->Par[5].Val.V_ENUM.members = &PremiaEnumAfd;

    }

    return OK;
}

```

```

PricingMethod MET(MC_LongstaffSchwartz_BermudanSwaption) =
{
    "MC_LongstaffSchwartz_BermudanSwaption",
    {
        {"N Simulation", LONG, {100}, ALLOW},
        {"RandomGenerator", ENUM, {100}, ALLOW},
        {"Basis", ENUM, {100}, ALLOW},
        {"Dimension Approximation", INT, {100}, ALLOW},
        {"Nbr discretisation step per periode", INT, {100}, ALLOW},
        {"Martingale Measure", ENUM, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_LongstaffSchwartz_BermudanSwaption),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_LongstaffSchwartz_BermudanSwaption),
    CHK_ok,
    MET(Init)
}

```

};