

## [Help](#)

```
#include "
href../../../../mod/lmm1d/lmm1d_std/ldmm1d_std_h_src.pdfldmm1d_std.h"
#include "
href../../../../common/math/mc_ldmm_glassermanzhao_h_src.pdfmath/mc_ldmm_glassermanzh
#include "
href../../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(MC_GZ)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_GZ)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// Compute the swaption Price under terminal measure using MonteCarlo simulation
static double lmm_european_swaption_pricer(double Nominal, long NbrMCsimulation,
{
    int m, N, save_all_paths, save_brownian, start_index, end_index, alpha;
    double tenor, price, payoff, variance, numeraire_0;

    PnlMat *LiborPathsMatrix;
    Libor *ptL_current;

    LiborPathsMatrix = pnl_mat_create(0, 0);

    N = ptLib->numberOfMaturities;
    tenor = ptSwpt->tenor;
    alpha = (int)(ptSwpt->swaptionMaturity / tenor); // T(alpha) is the swaption m
    start_index = 0;
    end_index = alpha;

    numeraire_0 = Numeraire(0, ptLib, flag_numeraire);

    save_brownian = 0;
```

```

save_all_paths = 0;

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

Sim_Libor_Glasserman(start_index, end_index, ptLib, ptVol, generator, NbrMCsim

variance = 0.;
price = 0.;

for (m = 0; m < NbrMCsimulation; m++)
{
    pnl_mat_get_row(ptL_current->libor, LiborPathsMatrix, m);
    payoff = Swaption_Payoff_Discounted(ptL_current, ptSwpt, p, flag_numeraire

    price += payoff;
    variance += SQR(payoff);
}

price = numeraire_0 * Nominal * price / NbrMCsimulation;
variance = SQR(numeraire_0 * Nominal) * variance / NbrMCsimulation;
variance = sqrt(variance - SQR(price));
freeLibor(&ptL_current);
pnl_mat_free(&LiborPathsMatrix);

return price;
}

static int mc_eurswaption_glassermanzhao_lmm1d(NumFunc_1 *p, double l0, double s
{
    Volatility *ptVol;
    Libor *ptLib;
    Swaption *ptSwpt;
    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(&ptSwpt, swaption_maturity, swap_maturity, 0.0, swaption_strike

```

```

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

*swaption_price = lmm_european_swaption_pricer(Nominal, NbrMCsimulation, p, pt

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

return init_mc;
}

int CALC(MC_GZ)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return mc_eurswaption_glassermanzhao_lmm1d(
        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_ENUM.value,
        Met->Par[1].Val.V_PINT,
        Met->Par[2].Val.V_LONG,
        Met->Par[3].Val.V_ENUM.value,
        &(Met->Res[0].Val.V_DOUBLE));
}

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

```

```

}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "MC_GlassermanZhao_swaption";
        Met->Par[0].Val.V_ENUM.value = 0;
        Met->Par[0].Val.V_ENUM.members = &PremiaEnumRNGs;
        Met->Par[1].Val.V_INT = 2;
        Met->Par[2].Val.V_LONG = 10000;
        Met->Par[3].Val.V_ENUM.value = 0;
        Met->Par[3].Val.V_ENUM.members = &PremiaEnumAfd;
    }
    return OK;
}

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

```