

## Help

```

#include <stdlib.h>
#include "merhes1d_std.h"
#include "pnl/pnl_basis.h"
#include "math/alfonsi.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(MC_AM_Alfonsi_LongstaffSchwartz_Bates)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_AM_Alfonsi_LongstaffSchwartz_Bates)(void *Opt, void *Mod, PricingMet
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Lower bound for american option using Longstaff-Schwartz algorithm */
// Exercice dates are : T(0), T(1), ..., T(NbrExerciseDates-1).
// with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
static int MC_Am_Alfonsi_LoS_Bates(NumFunc_1 *p, double S0, double Maturity, do
{
    int j, m, nbr_var_explicatives, init_mc;
    int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
    double continuation_value, discounted_payoff, european_price, european_delta,
    double *VariablesExplicatives;

    PnlMat *SpotPaths, *VarPaths, *AveragePaths, *ExplicativeVariables;
    PnlVect *DiscountedOptimalPayoff, *RegressionCoeffVect;
    PnlBasis *basis;

    /* Value to construct the confidence interval */
    z_alpha = pnl_inv_cdfnor((1. + confidence) / 2.);

    step = Maturity / (double)(NbrExerciseDates - 1);
    discount_step = exp(-r * step);
    discount = exp(-r * Maturity);

    /* We store Spot and Variance*/

```

```

flag_SpotPaths = 1;
flag_VarPaths  = 1;
flag_AveragePaths = 0;

european_price = 0.;
european_delta = 0.;

nbr_var_explicatives = 2;

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

VariablesExplicatives = malloc(nbr_var_explicatives * sizeof(double));

ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulation); // Continuation Va

RegressionCoeffVect = pnl_vect_create(0);
SpotPaths = pnl_mat_create(0, 0); // Matrix of the whole trajectories of the s
VarPaths = pnl_mat_create(0, 0); // Matrix of the whole trajectories of the va
AveragePaths = pnl_mat_create(0, 0);

init_mc = pnl_rand_init(generator, NbrExerciseDates * NbrStepPerPeriod, NbrMCs
if (init_mc != OK) return init_mc;

// Simulation of the whole paths
BatesSimulation_Alfonsi(flag_SpotPaths, SpotPaths, flag_VarPaths, VarPaths, fl

// At maturity, the price of the option = discounted_payoff
exercise_date = Maturity;
for (m = 0; m < NbrMCsimulation; m++)
{
    S_t = MGET(SpotPaths, NbrExerciseDates - 1, m); // Simulated Value of the
    LET(DiscountedOptimalPayoff, m) = discount * (p->Compute)(p->Par, S_t); //
}

for (j = NbrExerciseDates - 2; j >= 1; j--)
{
    /** Least square fitting */
    exercise_date -= step;
    discount /= discount_step;

```

```

for (m = 0; m < NbrMCsimulation; m++)
{
    V_t = MGET(VarPaths, j, m); // Simulated value of the variance
    S_t = MGET(SpotPaths, j, m); // Simulated value of the spot
    ApAlosHeston(S_t, p, Maturity - exercise_date, r, divid, V_t, k, theta)

    MLET(ExplicativeVariables, m, 0) = discount * european_price / S0;
    MLET(ExplicativeVariables, m, 1) = discount * european_delta * S_t * s
}

pnl_basis_fit_ls(basis, RegressionCoeffVect, ExplicativeVariables, Discoun

/** Dynamical programming equation */
for (m = 0; m < NbrMCsimulation; m++)
{
    V_t = MGET(VarPaths, j, m); // Simulated value of the variance
    S_t = MGET(SpotPaths, j, m); // Simulated value of the spot
    discounted_payoff = discount * (p->Compute)(p->Par, S_t); // Discounte

    if (discounted_payoff > 0.) // If the payoff is null, the OptimalPayof
    {
        ApAlosHeston(S_t, p, Maturity - exercise_date, r, divid, V_t, k, t

        VariablesExplicatives[0] = discount * european_price / S0;
        VariablesExplicatives[1] = discount * european_delta * S_t * sqrt(

        continuation_value = pnl_basis_eval(basis, RegressionCoeffVect, Va

        if (discounted_payoff > continuation_value)
        {
            LET(DiscountedOptimalPayoff, m) = discounted_payoff;
        }
    }
}

}

discount /= discount_step;

// At initial date, no need for regression, continuation value is just a plain
continuation_value = pnl_vect_sum(DiscountedOptimalPayoff) / (double)NbrMCsimu
discounted_payoff = discount * (p->Compute)(p->Par, S0);

```

```

/* Price */
mean_price = MAX(discounted_payoff, continuation_value);

/* Sum of squares */
var_price = SQR(pnl_vect_norm_two(DiscountedOptimalPayoff)) / (double)NbrMCsim;
var_price = MAX(var_price, SQR(discounted_payoff)) - SQR(mean_price);

/* Price estimator */
*ptPriceAm = mean_price;
*ptPriceAmError = sqrt(var_price / ((double)NbrMCsimulation - 1));

/* Price Confidence Interval */
*ptInfPriceAm = *ptPriceAm - z_alpha * (*ptPriceAmError);
*ptSupPriceAm = *ptPriceAm + z_alpha * (*ptPriceAmError);

free(VariablesExplicatives);
pnl_basis_free(&basis);
pnl_mat_free(&SpotPaths);
pnl_mat_free(&VarPaths);
pnl_mat_free(&AveragePaths);
pnl_mat_free(&ExplicativeVariables);

pnl_vect_free(&DiscountedOptimalPayoff);
pnl_vect_free(&RegressionCoeffVect);

return OK;
}

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

    double r, divid;

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

    return MC_Am_Alfonsi_LoSc_Bates(ptOpt->PayOff.Val.V_NUMFUNC_1,

```

```

    ptMod->S0.Val.V_PDOUBLE,
    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
    r,
    divid,
    ptMod->Sigma0.Val.V_PDOUBLE,
    ptMod->MeanReversion.Val.V_PDOUBLE,
    ptMod->LongRunVariance.Val.V_PDOUBLE,
    ptMod->Sigma.Val.V_PDOUBLE,
    ptMod->Rho.Val.V_PDOUBLE,
    ptMod->Mean.Val.V_PDOUBLE,
    ptMod->Variance.Val.V_PDOUBLE,
    ptMod->Lambda.Val.V_PDOUBLE,
    Met->Par[0].Val.V_LONG,
    Met->Par[1].Val.V_INT,
    Met->Par[2].Val.V_INT,
    Met->Par[3].Val.V_ENUM.value,
    Met->Par[4].Val.V_ENUM.value,
    Met->Par[5].Val.V_INT,
    Met->Par[6].Val.V_PDOUBLE,
    Met->Par[7].Val.V_ENUM.value,
    &(Met->Res[0].Val.V_DOUBLE),
    &(Met->Res[1].Val.V_DOUBLE),
    &(Met->Res[2].Val.V_DOUBLE),
    &(Met->Res[3].Val.V_DOUBLE));
}

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

    if ((opt->EuOrAm).Val.V_BOOL == AMER)
        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_am_alfonsi_longstaffschwartz_merhes";
        Met->Par[0].Val.V_LONG = 100000;
        Met->Par[1].Val.V_INT = 10;
        Met->Par[2].Val.V_INT = 1;
        Met->Par[3].Val.V_ENUM.value = 0;
        Met->Par[3].Val.V_ENUM.members = &PremiaEnumRNGs;
        Met->Par[4].Val.V_ENUM.value = 0;
        Met->Par[4].Val.V_ENUM.members = &PremiaEnumBasis;
        Met->Par[5].Val.V_INT = 10;
        Met->Par[6].Val.V_DOUBLE = 0.95;
        Met->Par[7].Val.V_ENUM.value = 2;
        Met->Par[7].Val.V_ENUM.members = &PremiaEnumCirOrder;
    }

    return OK;
}

PricingMethod MET(MC_AM_Alfonsi_LongstaffSchwartz_Bates) =
{
    "MC_AM_Alfonsi_LongstaffSchwartz_Bates",
    {
        {"N Simulations", LONG, {100}, ALLOW},
        {"N Exercise Dates", INT, {100}, ALLOW},
        {"N Steps per Period", INT, {100}, ALLOW},
        {"RandomGenerator", ENUM, {100}, ALLOW},
        {"Basis", ENUM, {100}, ALLOW},
        {"Dimension Approximation", INT, {100}, ALLOW},
        {"Confidence Value", DOUBLE, {100}, ALLOW},
        {"Cir Order", ENUM, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_AM_Alfonsi_LongstaffSchwartz_Bates),
    {
        {"Price", DOUBLE, {100}, FORBID},
        {"Error Price", DOUBLE, {100}, FORBID},
        {"Inf Price", DOUBLE, {100}, FORBID},
        {"Sup Price", DOUBLE, {100}, FORBID},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    }
}

```

```
    },  
    CHK_OPT(MC_AM_Alfonsi_LongstaffSchwartz_Bates),  
    CHK_ok,  
    MET(Init)  
};
```