

## [Help](#)

```
#include <stdlib.h>
#include "
href../../mod/merhes1d/merhes1d_pad/merhes1d_pad_h_src.pdfmerhes1d_pad.h"
#include "pnl/pnl_basis.h"
#include "
href../../common/math/alfonsi_h_src.pdfmath/alfonsi.h"
#include "
href../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates)(void *Opt, void *M
{
    return NONACTIVE;
}
int CALC(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates)(void *Opt, void *Mod, Pricin
{
    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_LoSc(NumFunc_2 *p, double S0, double Maturity, double
{
    int j, m, nbr_var_explicatives;
    int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
    double regressed_value, discounted_payoff, S_t, V_t, A_t, discount, discount_s
    double exercise_date, european_price, european_delta, V_mean;
    double *VariablesExplicatives;

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

    european_price = 0.;
    european_delta = 0.;

    step = Maturity / (NbrExerciseDates - 1);
```

```

discount_step = exp(-r * step);
discount = exp(-r * Maturity);

nbr_var_explicatives = 2;

/* We store Spot and Variance*/
flag_SpotPaths = 1;
flag_VarPaths = 1;
flag_AveragePaths = 1;

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

pnl_mat_resize(RegressionCoeffMat, NbrExerciseDates - 2, DimApprox);

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

ExplicativeVariables = pnl_mat_create(NbrMCsimulation, nbr_var_explicatives);
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulation); // Payoff if follo

RegressionCoeffVect = pnl_vect_create(0); // Regression coefficient.
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); // Matrix of the whole trajectories of th

// 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
    A_t = MGET(AveragePaths, NbrExerciseDates - 1, m); // Simulated Value of t
    LET(DiscountedOptimalPayoff, m) = discount * (p->Compute)(p->Par, S_t, A_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
    A_t = MGET(AveragePaths, j, m);

    // Regression basis contains price and delta of european asian option
    // As BS volatility, we take sqrt of expectation of V(Maturity) knowing
    V_mean = theta + (V_t - theta) * exp(-k * (Maturity - exercise_date));
    Ap_FixedAsian_BlackScholes(S_t, A_t, exercise_date, p, Maturity, r, di

    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

pnl_mat_set_row(RegressionCoeffMat, RegressionCoeffVect, j - 1); // Save r

/** 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
    A_t = MGET(AveragePaths, j, m);

    discounted_payoff = discount * (p->Compute)(p->Par, S_t, A_t); // Pay

    if (discounted_payoff > 0) // If the discounted_payoff is null, the Op
    {
        V_mean = theta + (V_t - theta) * exp(-k * (Maturity - exercise_dat
        Ap_FixedAsian_BlackScholes(S_t, A_t, exercise_date, p, Maturity, r

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

        regressed_value = pnl_basis_eval(basis, RegressionCoeffVect, Varia

        if (discounted_payoff > regressed_value)
        {

```

```

        LET(DiscountedOptimalPayoff, m) = discounted_payoff;
    }
}
}

// At initial date, no need for regression, conditional expectation is just a
*ContinuationValue_0 = pnl_vect_sum(DiscountedOptimalPayoff) / NbrMCsimulation

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;
}

/** Upper bound for american option using Andersen and Broadie algorithm.
* @param AmOptionUpperPrice upper bound for the price on exit.
* @param NbrMCsimulationDual number of outer simulation in Andersen and Broadie
* @param NbrMCsimulationDualInternal number of inner simulation in Andersen and
* @param NbrMCsimulationPrimal number of simulation in Longstaff-Schwartz algorithm
*/
static int MC_Am_Alfonsi_AnBr(double S0, double Maturity, double r, double divid
{
    int m, m_i, i, nbr_var_explicatives, ExerciceOrContinuation, init_mc;
    int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
    double discounted_payoff, discounted_payoff_inner, ContinuationValue, LowerPri

    double DoobMeyerMartingale, MaxVariable, S_t, V_t, A_t, S_t_inner, V_t_inner,
    double discount_step, discount, step, exercise_date, CondExpec_inner, Delta_0,
    double *VariablesExplicatives;

    PnlMat *RegressionCoeffMat;
    PnlMat *SpotPaths, *SpotPaths_inner;
    PnlMat *VarPaths, *VarPaths_inner;

```

```

PnlMat *AveragePaths, *AveragePaths_inner;
PnlVect *RegressionCoeffVect;
PnlBasis *basis;

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);
AveragePaths_inner = pnl_mat_create(0, 0);
SpotPaths_inner = pnl_mat_create(0, 0);
VarPaths_inner = pnl_mat_create(0, 0);
RegressionCoeffVect = pnl_vect_create(0);
RegressionCoeffMat = pnl_mat_create(0, 0);

/* We store Spot and Variance*/
flag_SpotPaths = 1;
flag_VarPaths = 1;
flag_AveragePaths = 1;

ContinuationValue_0 = 0.;
CondExpec_inner = 0;

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

nbr_var_explicatives = 2;
VariablesExplicatives = malloc(nbr_var_explicatives * sizeof(double));

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

/* Compute the lower price with Longstaff-Schwartz algorithm and save the regr
MC_Am_Alfonsi_LoSc(p, S0, Maturity, r, divid, V0, k, theta, sigma, rho, mu_jum

discounted_payoff = discount * (p->Compute)(p->Par, S0, S0);
LowerPrice_0 = MAX(discounted_payoff, ContinuationValue_0); // Price of am.opt

/* Simulation of the whole paths. These paths are independants of those used i
BatesSimulation_Alfonsi(flag_SpotPaths, SpotPaths, flag_VarPaths, VarPaths, fl

basis = pnl_basis_create(basis_name, DimApprox, nbr_var_explicatives);

```

```

Delta_0 = 0;

for (m = 0; m < NbrMCsimulationDual; m++)
{
    exercise_date = 0.;
    MaxVariable = 0.;
    discount = 1.;
    S_t = S0;
    V_t = V0;
    A_t = S0;

    ContinuationValue = ContinuationValue_0;
    discounted_payoff = discount * (p->Compute)(p->Par, S_t, A_t);

    LowerPrice = MAX(discounted_payoff, ContinuationValue);
    LowerPriceOld = LowerPrice;
    DoobMeyerMartingale = LowerPrice;

    /* Initialization of the duale variable. */
    MaxVariable = MAX(MaxVariable, discounted_payoff - DoobMeyerMartingale);

    for (i = 1; i <= NbrExerciseDates - 2; i++)
    {
        discount *= discount_step;
        exercise_date += step;

        pnl_mat_get_row(RegressionCoeffVect, RegressionCoeffMat, i - 1);

        ExerciceOrContinuation = (discounted_payoff > ContinuationValue);

        // If ExerciceOrContinuation=Exercice, we estimate the conditionnal ex
        if (ExerciceOrContinuation)
        {
            CondExpec_inner = 0;

            BatesSimulation_Alfonsi(flag_SpotPaths, SpotPaths_inner, flag_VarP

            for (m_i = 0; m_i < NbrMCsimulationDualInternal; m_i++)
            {
                S_t_inner = MGET(SpotPaths_inner, 1, m_i);
                V_t_inner = MGET(VarPaths_inner, 1, m_i);
            }
        }
    }
}

```

```

        A_t_inner = MGET(AveragePaths_inner, 1, m_i);

        discounted_payoff_inner = discount * (p->Compute)(p->Par, S_t_inner);

        V_mean = theta + (V_t_inner - theta) * exp(-k * (Maturity - exercise_date));
        Ap_FixedAsian_BlackScholes(S_t_inner, A_t_inner, exercise_date, p, Maturity, r, discount);

        VariablesExplicatives[0] = discount * european_price / S0;
        VariablesExplicatives[1] = discount * european_delta * S_t_inner * sqrt(V_t_inner);

        ContinuationValue_inner = pnl_basis_eval(basis, RegressionCoeffVect, VariablesExplicatives);

        CondExpec_inner += MAX(discounted_payoff_inner, ContinuationValue_inner);
    }

    CondExpec_inner /= (double)NbrMCsimulationDualInternal;
}

S_t = MGET(SpotPaths, i, m);
V_t = MGET(VarPaths, i, m);
A_t = MGET(AveragePaths, i, m);
discounted_payoff = discount * (p->Compute)(p->Par, S_t, A_t);

V_mean = theta + (V_t - theta) * exp(-k * (Maturity - exercise_date));
Ap_FixedAsian_BlackScholes(S_t, A_t, exercise_date, p, Maturity, r, discount);

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

ContinuationValue = pnl_basis_eval(basis, RegressionCoeffVect, VariablesExplicatives);

LowerPrice = MAX(discounted_payoff, ContinuationValue);

/* Compute the martingale part in Doob Meyer decomposition of the lower price
if (ExerciceOrContinuation)
{
    DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - CondExpec_inner;
}
else

```

```

        {
            DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - LowerPriceOld;
        }

        MaxVariable = MAX(MaxVariable, discounted_payoff - DoobMeyerMartingale);

        LowerPriceOld = LowerPrice;
    }

    /** Last Exercice Date. The price of the option here is equal to the discounted payoff.
    discount *= discount_step;

    // Decision to exercise or not before the last exercise date.
    ExerciceOrContinuation = (discounted_payoff > ContinuationValue);

    if (ExerciceOrContinuation)
    {
        BatesSimulation_Alfonsi(flag_SpotPaths, SpotPaths_inner, flag_VarPaths, VarPaths_inner);

        CondExpec_inner = 0;
        for (m_i = 0; m_i < NbrMCsimulationDualInternal; m_i++)
        {
            S_t_inner = MGET(SpotPaths_inner, 1, m_i);
            A_t_inner = MGET(AveragePaths, 1, m_i);
            discounted_payoff_inner = discount * (p->Compute)(p->Par, S_t_inner, A_t_inner);
            CondExpec_inner += discounted_payoff_inner;
        }
        CondExpec_inner /= (double) NbrMCsimulationDualInternal;
    }

    S_t = MGET(SpotPaths, NbrExerciseDates - 1, m);
    A_t = MGET(AveragePaths, NbrExerciseDates - 1, m);
    discounted_payoff = discount * (p->Compute)(p->Par, S_t, A_t);
    LowerPrice = discounted_payoff;

    if (ExerciceOrContinuation)
    {
        DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - CondExpec_inner;
    }
    else

```



```

        {
            DoobMeyerMartingale = DoobMeyerMartingale + LowerPrice - LowerPriceOld;
        }

        MaxVariable = MAX(MaxVariable, discounted_payoff - DoobMeyerMartingale);

        Delta_0 += MaxVariable;
    }

    Delta_0 /= NbrMCsimulationDual;
    *AmOptionUpperPrice = LowerPrice_0 + 0.5 * Delta_0;

    free(VariablesExplicatives);
    pnl_mat_free(&SpotPaths);
    pnl_mat_free(&VarPaths);
    pnl_mat_free(&AveragePaths);
    pnl_mat_free(&AveragePaths_inner);
    pnl_mat_free(&SpotPaths_inner);
    pnl_mat_free(&VarPaths_inner);
    pnl_mat_free(&RegressionCoeffMat);
    pnl_vect_free(&RegressionCoeffVect);
    pnl_basis_free(&basis);

    return init_mc;
}

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

    double T, t_0, T_0;
    double r, divid, time_spent, pseudo_strike, true_strike, pseudo_spot;
    int return_value;

    Met->Par[3].Val.V_INT = MAX(2, Met->Par[3].Val.V_INT); // At least two exercis

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

```

```

T = ptOpt->Maturity.Val.V_DATE;
T_0 = ptMod->T.Val.V_DATE;
t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;
time_spent = (T_0 - t_0) / (T - t_0);

if (T_0 < t_0)
{
    Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
    return_value = WRONG;
}

/* Case t_0 <= T_0 */
else
{
    pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;
    pseudo_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE - ti

    true_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = pseudo_strike;

    return_value = MC_Am_Alfonsi_AnBr(pseudo_spot,
                                      T - T_0,
                                      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_LONG,
                                      Met->Par[2].Val.V_LONG,
                                      Met->Par[3].Val.V_INT,
                                      Met->Par[4].Val.V_INT,
                                      Met->Par[5].Val.V_ENUM.value,
                                      Met->Par[6].Val.V_ENUM.value,

```

```

Met->Par[7].Val.V_INT,
Met->Par[8].Val.V_ENUM.value,
ptOpt->PayOff.Val.V_NUMFUNC_2,
&(Met->Res[0].Val.V_DOUBLE));

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = true_strike;
}
return return_value;
}

static int CHK_OPT(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates)(void *Opt, void *M
{
    if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedAmer") == 0) || (strcmp(((Op
        return OK;

    return WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "mc_am_asian_alfonsi_andersenbroadie_merhes";
        Met->Par[0].Val.V_LONG = 50000;
        Met->Par[1].Val.V_LONG = 500;
        Met->Par[2].Val.V_LONG = 500;
        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 = &PremiaEnumRNGs;
        Met->Par[6].Val.V_ENUM.value = 0;
        Met->Par[6].Val.V_ENUM.members = &PremiaEnumBasis;
        Met->Par[7].Val.V_INT = 10;
        Met->Par[8].Val.V_ENUM.value = 2;
        Met->Par[8].Val.V_ENUM.members = &PremiaEnumCirOrder;
    }

    return OK;
}

```

```

PricingMethod MET(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates) =
{
    "MC_Am_Asian_Alfonsi_AndersenBroadie_Bates",
    {
        {"N Sim.Primal", LONG, {100}, ALLOW},
        {"N Sim.Dual", LONG, {100}, ALLOW},
        {"N Sim.Dual Internal", 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},
        {"Cir Order", ENUM, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates),
    {"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(MC_Am_Asian_Alfonsi_AndersenBroadie_Bates),
    CHK_ok,
    MET(Init)
};

```