```
   Help
#include <stdlib.h>
#include  "hes1d_std.h"
#include "pnl/pnl_basis.h"
#include  "math/alfonsi.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_AM_Alfonsi_MLSM)(void *Opt, void *
    Mod)
{
    return NONACTIVE;
}
int CALC(MC_AM_Alfonsi_MLSM)(void *Opt,void *Mod,Pricing
    Method *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/** Price of american put/call option using Longstaff-Schwa
    rtz algorithm **/
/** Heston model is simulated using the method proposed by
    Alfonsi **/
// Exercice dates are : T(0), T(1), ..., T(NbrExerciseDate
    s-1).
// with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
static int MC_AM_Alfonsi_MLSM(NumFunc_1 *p, double S0,
    double Maturity, double r, double divid, double V0, double k,
    double theta, double sigma, double rho, long NbrMCsimulation,
    int NbrExerciseDates, int NbrStepPerPeriod, int generator,
    int basis_name, int DimApprox, int flag_cir, double *ptPriceA
    m, double *ptDeltaAm)
{
    int j, m, m_in_money, nbr_var_explicatives, init_mc;
    int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
    double S_init, continuation_value, discounted_payoff,
    S_t, V_t, alpha;
    double discount_step, discount, time_step, exercise_da
```

```
te;
double *VariablesExplicatives;

PnlMat *OneSpotPaths, *OneVarPaths, *SpotPaths, *VarP
aths, *AveragePaths, *ExplicativeVariables;
PnlVect *OptimalPayoff, *RegressionCoeffVect;
PnlVect *VectToRegress, *InititSpotPaths;
PnlBasis *basis;

init_mc=pnl_rand_init(generator, NbrExerciseDates*Nb
rStepPerPeriod, NbrMCsimulation);
if (init_mc != OK) return init_mc;

alpha = 0.1/Maturity;
nbr_var_explicatives = 2;

basis = pnl_basis_create(basis_name, DimApprox, nbr_
var_explicatives);

VariablesExplicatives = malloc(nbr_var_explicatives*si
zeof(double));

ExplicativeVariables = pnl_mat_create(NbrMCsimulation,
nbr_var_explicatives);
OptimalPayoff = pnl_vect_create(NbrMCsimulation);  //
Payoff if following optimal strategy.
InititSpotPaths = pnl_vect_create_from_double(NbrMCsimu
lation, S0);
VectToRegress = pnl_vect_create(NbrMCsimulation);

RegressionCoeffVect = pnl_vect_create(0); // Regression
 coefficient.
SpotPaths = pnl_mat_create(NbrExerciseDates, NbrMCsimu
lation); // Matrix of the whole trajectories of the spot
VarPaths = pnl_mat_create(NbrExerciseDates, NbrMCsimu
lation); // Matrix of the whole trajectories of the variance
AveragePaths = pnl_mat_create(0, 0);
OneSpotPaths = pnl_mat_create(0, 0);
OneVarPaths = pnl_mat_create(0, 0);

time_step = Maturity / (double)(NbrExerciseDates-1);
```

```
discount_step = exp(-r*time_step);
discount = exp(-r*Maturity);

/* We store Spot and Variance*/
flag_SpotPaths = 1;
flag_VarPaths  = 1;
flag_AveragePaths = 0;

HestonSimulation_Alfonsi(flag_SpotPaths, OneSpotPaths,
flag_VarPaths, OneVarPaths, flag_AveragePaths, AveragePaths,
 S0*exp(-r*Maturity*alpha), Maturity*alpha, r, divid, V0,
k, theta, sigma, rho, NbrMCsimulation, 2, 2, generator, fla
g_cir);

for (m=0; m<NbrMCsimulation; m++)
{
    LET(InititSpotPaths, m) = MGET(OneSpotPaths, 1, m);
}

// Simulation of the whole paths
for (m=0; m<NbrMCsimulation; m++)
{
    S_init = GET(InititSpotPaths, m);
    HestonSimulation_Alfonsi(flag_SpotPaths, OneSpotP
aths, flag_VarPaths, OneVarPaths, flag_AveragePaths, Avera
gePaths, S_init, Maturity, r, divid, V0, k, theta, sigma, rh
o, 1, NbrExerciseDates, NbrStepPerPeriod, generator, flag_
cir);

    for (j=0; j<NbrExerciseDates; j++)
    {
        MLET(SpotPaths, j, m) = MGET(OneSpotPaths, j, 0
);
        MLET(VarPaths, j, m)  = MGET(OneVarPaths, j, 0)
;
    }
}

// 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); // Si
mulated value of the spot at the maturity T
    LET(OptimalPayoff, m) = discount * (p->Compute)(p->
Par, S_t)/S0; // Discounted payoff
}

for (j=NbrExerciseDates-2; j>=0; j--)
{
    /** Least square fitting **/
    exercise_date -= time_step;
    discount /= discount_step;

    m_in_money=0;
    pnl_mat_resize(ExplicativeVariables, NbrMCsimulatio
n, nbr_var_explicatives);
    pnl_vect_resize(VectToRegress, NbrMCsimulation);
    for (m=0; m<NbrMCsimulation; m++)
    {
        V_t = MGET(VarPaths, j, m); // Simulated value
of the variance at t=exercise_date
        S_t = MGET(SpotPaths, j, m); // Simulated value
 of the spot at t=exercise_date

        discounted_payoff = discount * (p->Compute)(p->
Par, S_t)/S0;

        if (discounted_payoff>0)
        {
            MLET(ExplicativeVariables, m_in_money, 0) =
 S_t/S0;
            MLET(ExplicativeVariables, m_in_money, 1) =
 V_t/V0;

            LET(VectToRegress, m_in_money) = GET(Optim
alPayoff, m);
            m_in_money++;
        }
    }
    pnl_mat_resize(ExplicativeVariables, m_in_money, nb
r_var_explicatives);
    pnl_vect_resize(VectToRegress, m_in_money);

    pnl_basis_fit_ls(basis, RegressionCoeffVect, Explic
ativeVariables, VectToRegress);

    /** Dynamical programming equation **/
    for (m=0; m<NbrMCsimulation; m++)
    {
        V_t = MGET(VarPaths, j, m);
        S_t = MGET(SpotPaths, j, m);
        discounted_payoff = discount * (p->Compute)(p->
Par, S_t)/S0; // Discounted payoff

        if (discounted_payoff>0.) // If the payoff is
null, the OptimalPayoff doesnt change.
        {
            VariablesExplicatives[0] = S_t/S0;
            VariablesExplicatives[1] = V_t/V0;

            continuation_value = pnl_basis_eval(basis,
RegressionCoeffVect, VariablesExplicatives);

            if (discounted_payoff > continuation_value)
            {
                LET(OptimalPayoff, m) = discounted_payo
ff;
            }
        }
    }
}

pnl_mat_resize(ExplicativeVariables, NbrMCsimulation,
nbr_var_explicatives);
pnl_vect_resize(VectToRegress, NbrMCsimulation);
for (m=0; m<NbrMCsimulation; m++)
{
    V_t = MGET(VarPaths, 0, m);
    S_t = MGET(SpotPaths, 0, m);

    MLET(ExplicativeVariables, m, 0) = S_t/S0;
```

```
        MLET(ExplicativeVariables, m, 1) = V_t/V0;

        LET(VectToRegress, m) = GET(OptimalPayoff, m);
    }
    pnl_basis_fit_ls(basis, RegressionCoeffVect, Explicati
    veVariables, VectToRegress);

    VariablesExplicatives[0] = 1.;
    VariablesExplicatives[1] = 1.;

    *ptPriceAm = S0*pnl_basis_eval(basis, RegressionCoeffV
    ect, VariablesExplicatives);

    *ptDeltaAm = pnl_basis_eval_D(basis, RegressionCoeffVec
    t, VariablesExplicatives, 0);

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

    pnl_vect_free(&OptimalPayoff);
    pnl_vect_free(&RegressionCoeffVect);
    pnl_vect_free(&InititSpotPaths);
    pnl_vect_free(&VectToRegress);

    return OK;
}


int CALC(MC_AM_Alfonsi_MLSM)(void *Opt, void *Mod, Pricing
    Method *Met)
{
    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.);
    Met->Par[1].Val.V_INT = MAX(2, Met->Par[1].Val.V_INT);
    // At least two exercise dates.


    return MC_AM_Alfonsi_MLSM(ptOpt->PayOff.Val.V_NUMFUNC_1
    ,
                                   ptMod->S0.Val.V_PDOUBLE,
                                   ptOpt->Maturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                   r,
                                   divid,
                                   ptMod->Sigma0.Val.V_PDO
    UBLE,
                                   ptMod->MeanReversion.h
    al.V_PDOUBLE,
                                   ptMod->LongRunVariance.
    Val.V_PDOUBLE,
                                   ptMod->Sigma.Val.V_PDOUB
    LE,
                                   ptMod->Rho.Val.V_PDOUB
    LE,
                                   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_ENUM.
    value,
                                   &(Met->Res[0].Val.V_
    DOUBLE),
                                   &(Met->Res[1].Val.V_
    DOUBLE));
}

static int CHK_OPT(MC_AM_Alfonsi_MLSM)(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->Par[0].Val.V_LONG=50000;
        Met->Par[1].Val.V_INT=20;
        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_ENUM.value=2;
        Met->Par[6].Val.V_ENUM.members=&PremiaEnumCirOrder;
    }

    return OK;
}


PricingMethod MET(MC_AM_Alfonsi_MLSM)=
{
    "MC_AM_Alfonsi_MLSM",
    {
        {"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},
        {"Cir Order",ENUM,{100},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(MC_AM_Alfonsi_MLSM),
    {   {"Price",DOUBLE,{100},FORBID},
        {"Delta",DOUBLE,{100},FORBID},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(MC_AM_Alfonsi_MLSM),
    CHK_ok,
    MET(Init)
};
```

References