```
   Help
#include "hes1d_std.h"
#include "enums.h"
#include "pnl/pnl_interpolation.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+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(TR_VELLEKOOPNIEUWENHUIS_Heston)(void *
    Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_VELLEKOOPNIEUWENHUIS_Heston)(void *Opt,void *
    Mod,PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static void linestep(PnlVect *xgrid, double xmin, double dx
    , int nx)
{
    int i;
    pnl_vect_resize(xgrid, nx);

    for (i=0; i<nx; i++)
    {
        LET(xgrid,i) = xmin+i*dx;
    }
}

static double MGET_OptionValue(PnlMat *OptionValue, int i,
    int j)
{
    int i1, j1;

    i1 = MIN(i, OptionValue->m-1);
    j1 = MIN(j, OptionValue->n-1);

    i1 = MAX(0, i1);
```

```
    j1 = MAX(0, j1);

    return MGET(OptionValue, i1, j1);
}

static int HestonTreeAmericanPut(double kappa, double thet
    a, double omega, double rho, double V0, double S0, double
    maturity, double r, double divid, int n, int mv, int mz, int
    UseBS, int Eur_or_Am, NumFunc_1  *p, double *ptprice,
    double *ptdelta)
{
    double dt, dv, dz, dv_next, dz_next;
    double S, z, v, v_old, z_next, v_next, strike;
    double min_z, max_z, min_v, max_v;
    double min_z_old, max_z_old, min_v_old, max_v_old;
    double min_z_next, max_z_next, min_v_next, max_v_next;
    double discount_step,proba_lv_lz, payoff, option_price1
    , option_price2;
    int last_index, i, j, k, l, lv, lz, i_v, i_z, outmode,
    type;
    double *C;

    PnlMat *OptionValue, *OptionValue_next, *OptionValue_
    interp;
    PnlVect *min_v_vect, *max_v_vect, *min_z_vect, *max_z_
    vect;
    PnlVect *v_next_grid, *z_next_grid, *v_next_to_interp,
    *z_next_to_interp;

    PnlVect *sign_move_v;
    PnlVect *sign_move_z;
    PnlVect *proba_move_zv;

    n++;
    mz++;
    mv++;

    v_next_grid = pnl_vect_create(mv);
    z_next_grid = pnl_vect_create(mz);
    v_next_to_interp = pnl_vect_create(mv*mz);
    z_next_to_interp = pnl_vect_create(mv*mz);
```

```
OptionValue_next = pnl_mat_create(mv, mz);
OptionValue = pnl_mat_create(mv, mz);
OptionValue_interp = pnl_mat_create(mv, mz);

min_v_vect = pnl_vect_create(n);
max_v_vect = pnl_vect_create(n);
min_z_vect = pnl_vect_create(n);
max_z_vect = pnl_vect_create(n);

C = malloc(4*4*(mv-1)*(mz-1)*sizeof(double));

strike = p->Par[0].Val.V_DOUBLE;
r = r - divid;
type = NATURAL; // type of bicubic spline to compute
outmode = NATURAL; //set the behavior to evaluate the
bicubic outside the grid

dt = maturity/(n-1);
discount_step = exp(-r*dt);

sign_move_v = pnl_vect_create_from_list(4, -1.,-1.,1.,1
.);
sign_move_z = pnl_vect_create_from_list(4, -1.,1.,1.,-1
.);
proba_move_zv = pnl_vect_create_from_list(4, 0.25*dis
count_step*(1.+rho),0.25*discount_step*(1.-rho),0.25*discoun
t_step*(1.+rho),0.25*discount_step*(1.-rho));

v = V0;
z = log(S0);

LET(min_v_vect, 0) = v;
LET(max_v_vect, 0) = v;
LET(min_z_vect, 0) = z;
LET(max_z_vect, 0) = z;

for (i=1; i<n; i++)
{
    min_v_old = GET(min_v_vect, i-1);
    max_v_old = GET(max_v_vect, i-1);
```

```
    min_z_old = GET(min_z_vect, i-1);
    max_z_old = GET(max_z_vect, i-1);

    dv = (max_v_old-min_v_old)/(double)(mv-1);
    dz = (max_z_old-min_z_old)/(double)(mz-1);

    max_v = 0.;
    min_v = 100.;
    v_old = min_v_old;
    for (j=0; j<mv; j++) // Compute max_v and min_v
    {
        v = v_old + kappa*(theta-v_old)*dt + omega*sq
rt(MAX(0,v_old)*dt);
        max_v = MAX(max_v, v);

        v = v_old + kappa*(theta-v_old)*dt - omega*sq
rt(MAX(0,v_old)*dt);
        min_v = MIN(min_v, v);
        v_old += dv;
    }
    LET(min_v_vect, i) = min_v;
    LET(max_v_vect, i) = max_v;

    max_z = max_z_old + (r-0.5*max_v_old)*dt + sqrt(MAX
(0,max_v_old)*dt);
    LET(max_z_vect, i) = max_z;
    min_z = min_z_old + (r-0.5*max_v_old)*dt - sqrt(MAX
(0,max_v_old)*dt);
    LET(min_z_vect, i) = min_z;

    //printf("i=%i, min_v=%f, max_v=%f, max_S=%f, max_
S=%f {n", i, min_v, max_v, exp(min_z), exp(max_z));
}

///********************* Dynamic Programing

// We start by initialise the price of the option at
maturity.
if (UseBS==1) // Use black Scholes option price as ini
tialization
{
```

```
    last_index = n-2;
}
else
{
    last_index = n-1;
}

min_v = GET(min_v_vect, last_index);
max_v = GET(max_v_vect, last_index);
min_z = GET(min_z_vect, last_index);
max_z = GET(max_z_vect, last_index);

dv = (max_v-min_v)/(double)(mv-1);
dz = (max_z-min_z)/(double)(mz-1);

z = min_z;
for (k=0; k<mz; k++)
{
    S = exp(z);
    payoff = (p->Compute)(p->Par, S);

    v = min_v;
    for (j=0; j<mv; j++)
    {
        if (UseBS==1) // Use black Scholes option
price as initialization
        {
            MLET(OptionValue_next, j, k) = pnl_bs_put(
S, strike, dt, r, 0., MAX(1e-10,sqrt(v)));
            if ((p->Compute)==&Call)
            {
                MLET(OptionValue_next, j, k) += S-stri
ke*exp(-r*dt);
            }
        }

        else
        {
            MLET(OptionValue_next, j, k) = payoff;
        }
```

```
            if (Eur_or_Am == 1)
            {
                MLET(OptionValue_next, j, k) = MAX(MGET(
    OptionValue_next, j, k), payoff);
            }

            v += dv;
        }
        z +=dz;
    }

    // Backward iteration
    for (i=last_index-1; i>=1; i--)
    {
        min_v = GET(min_v_vect, i);
        max_v = GET(max_v_vect, i);
        min_z = GET(min_z_vect, i);
        max_z = GET(max_z_vect, i);

        min_v_next = GET(min_v_vect, i+1);
        max_v_next = GET(max_v_vect, i+1);
        min_z_next = GET(min_z_vect, i+1);
        max_z_next = GET(max_z_vect, i+1);

        dv = (max_v-min_v)/(double)(mv-1);
        dz = (max_z-min_z)/(double)(mz-1);
        dv_next = (max_v_next-min_v_next)/(double)(mv-1);
        dz_next = (max_z_next-min_z_next)/(double)(mz-1);

        linestep(v_next_grid, min_v_next, dv_next, mv);
        linestep(z_next_grid, min_z_next, dz_next, mz);

        pnl_mat_set_double(OptionValue, 0.0);
        for (l=0;l<4;l++)
        {
            lv = GET(sign_move_v, l);
            lz = GET(sign_move_z, l);
            proba_lv_lz = GET(proba_move_zv, l);

            v = min_v;
            for (j=0; j<mv; j++)
```

```
        {
            //v = min_v + j*dv;
            v_next = v + kappa*(theta-v)*dt + lv*omega*
sqrt(MAX(0,v)*dt);
            z_next = min_z + (r-0.5*v)*dt + lz*sqrt(MAX
(0,v)*dt);
            for (k=0; k<mz; k++)
            {
                //z = min_z + k*dz;
                //z_next = z + (r-0.5*v)*dt + lz*sqrt(
MAX(0,v)*dt);
                LET(v_next_to_interp, k + j*mz) = v_nex
t;
                LET(z_next_to_interp, k + j*mz) = z_nex
t;
                z_next += dz;
            }
            v += dv;
        }

        pnl_bicubic_spline(z_next_grid, v_next_grid,
OptionValue_next, C, type);
        pnl_eval_bicubic(z_next_grid, v_next_grid, C,
z_next_to_interp, v_next_to_interp, OptionValue_interp, NUL
L, NULL,  outmode);

        //Max_Mat_Zero(OptionValue_interp);
        pnl_mat_axpy(proba_lv_lz, OptionValue_interp,
OptionValue);
    }

    if (Eur_or_Am == 1)
    {
        for (k=0; k<mz; k++)
        {
            z = min_z + k*dz;
            S = exp(z);
            payoff = (p->Compute)(p->Par, S);
            for (j=0; j<mv; j++)
            {
                MLET(OptionValue, j, k) = MAX(MGET(
```

```
OptionValue, j, k), payoff);
            }
        }
    }


    pnl_mat_clone(OptionValue_next, OptionValue);
}

// Initial date t=0.
*ptprice = GET(proba_move_zv,0)*MLET(OptionValue, 0, 0)
 + GET(proba_move_zv,1)*MLET(OptionValue, 0, mz-1) +
           GET(proba_move_zv,2)*MLET(OptionValue, mv-1,
 mz-1) + GET(proba_move_zv,3)*MLET(OptionValue, mv-1, 0);
*ptprice = exp(-divid*maturity) * (*ptprice);

// Option delta.
min_v = GET(min_v_vect, 1);
max_v = GET(max_v_vect, 1);
min_z = GET(min_z_vect, 1);
max_z = GET(max_z_vect, 1);

dv = (max_v-min_v)/(double)(mv-1);
dz = (max_z-min_z)/(double)(mz-1);

v = V0;
z = log(S0);

i_v = (int) floor((v-min_v)/dv);
i_z = (int) floor((z-min_z)/dz);

option_price1 = MGET_OptionValue(OptionValue, i_v, i_z)
;
option_price2 = MGET_OptionValue(OptionValue, i_v, i_z+
1);

*ptdelta = exp(-divid*maturity)*discount_step*(option_
price2-option_price1)/(exp(min_z + (i_z+1)*dz)-exp(min_z + (i_
z)*dz));

free(C);
```

```
    pnl_mat_free(&OptionValue);
    pnl_mat_free(&OptionValue_next);
    pnl_mat_free(&OptionValue_interp);

    pnl_vect_free(&min_v_vect);
    pnl_vect_free(&max_v_vect);
    pnl_vect_free(&min_z_vect);
    pnl_vect_free(&max_z_vect);

    pnl_vect_free(&v_next_grid);
    pnl_vect_free(&z_next_grid);
    pnl_vect_free(&v_next_to_interp);
    pnl_vect_free(&z_next_to_interp);

    pnl_vect_free(&sign_move_v);
    pnl_vect_free(&sign_move_z);
    pnl_vect_free(&proba_move_zv);

    return OK;
}

int CALC(TR_VELLEKOOPNIEUWENHUIS_Heston)(void *Opt, void *
    Mod, PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid;

    if (ptMod->Sigma.Val.V_PDOUBLE==0.0)
    {
        Fprintf(TOSCREEN,"BLACK-SHOLES MODEL{n{n{n");
        return WRONG;
    }
    else
    {
        r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
        divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);

        return HestonTreeAmericanPut(ptMod->MeanReversion.h
    al.V_PDOUBLE,
```

```
                                        ptMod->LongRunVarianc
    e.Val.V_PDOUBLE,
                                        ptMod->Sigma.Val.V_PDO
    UBLE,
                                        ptMod->Rho.Val.V_PDOUB
    LE,
                                        ptMod->Sigma0.Val.V_
    PDOUBLE,
                                        ptMod->S0.Val.V_PDOUB
    LE,
                                        ptOpt->Maturity.Val.V_
    DATE-ptMod->T.Val.V_DATE,
                                        r,
                                        divid,
                                        Met->Par[0].Val.V_INT,
                                        Met->Par[1].Val.V_INT,
                                        Met->Par[2].Val.V_INT,
                                        Met->Par[3].Val.V_
    ENUM.value,
                                        ptOpt->EuOrAm.Val.V_BO
    OL,
                                        ptOpt->PayOff.Val.V_
    NUMFUNC_1,
                                        &(Met->Res[0].Val.V_
    DOUBLE),
                                        &(Met->Res[1].Val.V_
    DOUBLE));
    }

}


static int CHK_OPT(TR_VELLEKOOPNIEUWENHUIS_Heston)(void *
    Opt, void *Mod)
{
    if ( (strcmp( ((Option*)Opt)->Name,"CallEuro")==0) || (
    strcmp( ((Option*)Opt)->Name,"PutEuro")==0||(strcmp( ((
    Option*)Opt)->Name,"CallAmer")==0) || (strcmp( ((Option*)Opt)->
    Name,"PutAmer")==0)))
        return OK;
    else
```

```
        return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
    if ( Met->init == 0)
    {
      Met->init=1;
        Met->HelpFilenameHint = "tr_vellekoop_heston";
        Met->Par[0].Val.V_INT=20;
        Met->Par[1].Val.V_INT=20;
        Met->Par[2].Val.V_INT=200;
        Met->Par[3].Val.V_ENUM.value=1;
        Met->Par[3].Val.V_ENUM.members=&PremiaEnumBool;
    }

    return OK;
}

PricingMethod MET(TR_VELLEKOOPNIEUWENHUIS_Heston)=
{
    "TR_VELLEKOOPNIEUWENHUIS_Heston",
    {
        {"N steps t",INT,{100},ALLOW},
        {"N steps V",INT,{100},ALLOW},
        {"N steps S",INT,{100},ALLOW},
        {"Smoothing with BS Formula",ENUM,{1},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(TR_VELLEKOOPNIEUWENHUIS_Heston),
    {
        {"Price",DOUBLE,{100},FORBID},
        {"Delta",DOUBLE,{100},FORBID} ,
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(TR_VELLEKOOPNIEUWENHUIS_Heston),
    CHK_ok,
    MET(Init)
};
```

References