```
   Help
#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)
#else

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h" // To use the function "inta
    pprox"
#include "TreeHW1dGeneralized.h"

// Return the voltility of the short rate
double Current_VolatilityHW1dG(ModelHW1dG* HW1dG, double t)
{
    int i, N;

    i=0;
    N = (HW1dG->TimeGrid)->size;

    if(HW1dG->TimeGrid==NULL) {printf("FATALE ERREUR, PAS
    DE GRILLE DE TEMPS !");}

    else
    {
        while(GET(HW1dG->TimeGrid, i)<t && i<N-1)
        {
            i++;
        }
    }

    return GET(HW1dG->ShortRateVolGrid, i);
}

// Construction of the time grid
int SetTimeGridHW1dG(TreeHW1dG *Meth, int n, double date,
    double T)
{
    int i;
```

```
    double delta_time;
    int i_date;

    Meth->Ngrid=n;
    Meth->Tf=T;

    Meth->t = pnl_vect_create(n+2);

    delta_time = T/n;

    for(i=0; i<=n+1; i++)
    {
        LET(Meth->t, i) = i * delta_time;
    }

    i_date = (int) floor(date / delta_time);

    if ( (i_date > 0) && ((GET(Meth->t, i_date+1)-date) >
    delta_time*INC))
    {
        LET(Meth->t, i_date) = date;
    }

    return i_date;
}

int SetTimeGrid_TenorHW1dG(TreeHW1dG *Meth, int NtY,
    double T0, double S0, double periodicity)
{
    int i;
    double delta_time, delta_time1;
    int n, m;

    delta_time = periodicity/NtY;

    n = (int) ((S0-T0)/periodicity + 0.1);
    m = (int) floor(T0/delta_time);

    delta_time1 = T0/m;

    Meth->Tf = S0;
```

```
    Meth->Ngrid = m + n*NtY;

    Meth->t = pnl_vect_create(Meth->Ngrid+2);

    for(i=0; i<=m; i++)
    {
        LET(Meth->t, i) = i * delta_time1; // Discretizati
    on of [0, T0]
    }

    for(i=m + 1; i<=m + n*NtY+1; i++)
    {
        LET(Meth->t, i) = T0 + (i-m) * delta_time; // Discr
    etization of ]T0, S0]
    }

    return i;
}

void SetTreeHW1dG(TreeHW1dG* Meth, ModelHW1dG* ModelParam,
    ZCMarketData* ZCMarket)
{
    double mean_reversion ;
    double sigma ;

    double Pdown, Pmiddle, Pup;
    double Q2Value, sum_alpha;

    double delta_x1, delta_x2;
    double delta_t1, delta_t2;

    double current_rate;
    double beta_x;
    int jminprev, jmaxprev;
    int jmin,jmax;
    int i, j, h;

    PnlVect* Q1; // Quantity used to calibrate the tree to
    the initial yield curve (see the book Brigo&Mercurio p.80)
    PnlVect* Q2; // Quantity used to calibrate the tree to
    the initial yield curve (see the book Brigo&Mercurio p.80)
```

```
mean_reversion = (ModelParam->MeanReversion);

Meth->Jminimum = pnl_vect_int_create(Meth->Ngrid + 1);
Meth->Jmaximum = pnl_vect_int_create(Meth->Ngrid + 1);

pnl_vect_int_set(Meth->Jminimum,0,0);
pnl_vect_int_set(Meth->Jmaximum,0,0);

pnl_vect_int_set(Meth->Jminimum, 1,-1);
pnl_vect_int_set(Meth->Jmaximum, 1, 1);

// Calcul de alpha(0) et alpha(1)
Meth->alpha = pnl_vect_create(Meth->Ngrid + 1);
Q1 = pnl_vect_create(3);
Q2 = pnl_vect_create(1);

delta_t1 = GET(Meth->t, 1) - GET(Meth->t,0); // = t[1]
- t[0]
delta_t2 = GET(Meth->t, 2) - GET(Meth->t,1); // = t[2]
- t[1]

LET(Meth->alpha, 0) = -log(BondPrice(GET(Meth->t, 1),
ZCMarket))/delta_t1; // alpha(0) = -log(Pm(0,t1))/t1

Pup = 1.0/6.0;
Pmiddle = 2.0/3.0;
Pdown = 1- Pmiddle - Pup;

LET(Q1, 0) = Pdown   * exp(- GET(Meth->alpha,0) * delt
a_t1); // Q(1,-1)
LET(Q1, 1) = Pmiddle * exp(- GET(Meth->alpha,0) * delt
a_t1); // Q(1,0)
LET(Q1, 2) = Pup     * exp(- GET(Meth->alpha,0) * delt
a_t1); // Q(1,-2)

sigma = Current_VolatilityHW1dG(ModelParam, GET(Meth->
t, 1)); // sigma(t1)

delta_x1 = SpaceStepHW1dG(delta_t1, sigma);//SpaceStep    HW1dG(delta_t1, a,
```

```
sum_alpha = GET(Q1, 0) * exp(delta_x1 * delta_t2) + GET
(Q1, 1) + GET(Q1, 2) * exp(-delta_x1 * delta_t2);

LET(Meth->alpha, 1) = log(sum_alpha/BondPrice(GET(Meth-
>t, 2), ZCMarket))/delta_t2;

jmin =-1; jmax = 1;


for ( i =1; i<Meth->Ngrid ; i++)
{
    delta_t1 = GET(Meth->t, i) - GET(Meth->t,i-1); // =
 t[i] - t[i-1]
    delta_t2 = GET(Meth->t, i+1) - GET(Meth->t,i); // =
 t[i+1] - t[i]

    sigma = Current_VolatilityHW1dG(ModelParam, GET(
Meth->t, i)); // sigma(ti)
    delta_x1 = SpaceStepHW1dG(delta_t1, sigma);// delt
a_x[i]
    sigma = Current_VolatilityHW1dG(ModelParam, GET(
Meth->t, i+1)); // sigma(ti+1)
    delta_x2 = SpaceStepHW1dG(delta_t2, sigma);// delt
a_x[i+1]

    beta_x = delta_x1 / delta_x2;

    jminprev = jmin; // jminprev := jmin[i]
    jmaxprev = jmax; // jmaxprev := jmax[i]

    jmin = intapprox(jminprev * beta_x * exp(-delta_t2
* mean_reversion)) - 1; // jmin := jmin[i+1]
    jmax = intapprox(jmaxprev * beta_x * exp(-delta_t2
* mean_reversion)) + 1; // jmax := jmax[i+1]

    pnl_vect_int_set(Meth->Jminimum, i+1, jmin);
    pnl_vect_int_set(Meth->Jmaximum, i+1, jmax);

    pnl_vect_resize(Q2, jmax-jmin+1); // Q1 :=Q(i,.) et
 Q2 : =Q(i+1,.)
```

```
    pnl_vect_set_double(Q2, 0);

    for (h= jminprev ; h <=jmaxprev ; h++)
    {
        current_rate = GET(Meth->alpha, i) + h*delta_x1
;

        j = intapprox(h*beta_x*exp(-delta_t2 * mean_rev
ersion)); //j index of the middle node emanating from (i,h)

        // Probability to go from (i,h) to (i+1,j+1)
with an UP movement
        Pup = ProbaUpHW1dG(h, j, delta_t2, beta_x, mea
n_reversion);
        // Probability to go from (i,h) to (i+1,j) wit
h a Middle movement
        Pmiddle = ProbaMiddleHW1dG(h, j, delta_t2, bet
a_x, mean_reversion);
         // Probability to go from (i,h) to (i+1,j-1)
with a Down movement
        Pdown = 1 - Pup - Pmiddle;

        Q2Value = GET(Q2, j+1-jmin) + GET(Q1, h-jminp
rev) * Pup * exp(-current_rate*delta_t2);
        LET(Q2, j+1-jmin) = Q2Value;

        Q2Value = GET(Q2, j-jmin) + GET(Q1, h-jminprev)
 * Pmiddle * exp(-current_rate*delta_t2);
        LET(Q2, j-jmin) = Q2Value;

        Q2Value = GET(Q2, j-1-jmin) + GET(Q1, h-jminp
rev) * Pdown * exp(-current_rate*delta_t2);
        LET(Q2, j-1-jmin) = Q2Value;

    } //END loop over h

    delta_t2 = GET(Meth->t, i+2) - GET(Meth->t,i+1);
    sum_alpha =0;
    for (j= jmin ; j <=jmax ; j++) { sum_alpha += GET(
Q2, j-jmin) * exp(-j*delta_x2*delta_t2); }
```

```
        LET(Meth->alpha, i+1) = log(sum_alpha/BondPrice(GET
    (Meth->t, i+2), ZCMarket))/delta_t2;

        pnl_vect_clone(Q1, Q2);

    } // End Loop on i

    pnl_vect_free(&Q1);
    pnl_vect_free(&Q2);

}// FIN de la fonction SetTreeHW1dG

// Backward computation of the price of a product from
    time Meth->t[index_last] to Meth->t[index_first].
// The initial value of the product (at time Meth->t[index_
    last]) is the vector OptionPriceVect2.
void BackwardIterationHW1dG(TreeHW1dG* Meth, ModelHW1dG*
    ModelParam, PnlVect* OptionPriceVect1, PnlVect* OptionPriceVec
    t2, int index_last, int index_first)
{
    double mean_reversion;
    double sigma;

    int jmin; // jmin[i+1], jmax[i+1]
    int jminprev, jmaxprev; // jmin[i], jmax [i]
    int i, j, k;
    double Pup, Pdown, Pmiddle;
    double delta_x1, delta_x2, beta_x;
    double delta_t1, delta_t2;
    double current_rate;


    ///******************Parameters of the process r *****
    ****************////
    mean_reversion = (ModelParam->MeanReversion);

    jminprev = pnl_vect_int_get(Meth->Jminimum, index_last)
    ;  // jmin(index_last)
    jmaxprev = pnl_vect_int_get(Meth->Jmaximum, index_last)
    ;  // jmax(index_last)
```

```
for(i = index_last-1; i>=index_first; i--)
{
    jmin = jminprev; // jmin := jmin(i+1)

    jminprev = pnl_vect_int_get(Meth->Jminimum, i);  //
 jmin(i)
    jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i);  //
 jmax(i)

    pnl_vect_resize(OptionPriceVect1, jmaxprev-jminprev
 +1);  // OptionPriceVect1 := Price of the bond in the tre
e at time t(i)

    delta_t1 = GET(Meth->t, i) - GET(Meth->t,i-1); //
Time step between t[i] et t[i-1]
    delta_t2 = GET(Meth->t, i+1) - GET(Meth->t,i); //
Time step between t[i+1] et t[i]

    sigma = Current_VolatilityHW1dG(ModelParam, GET(
Meth->t, i)); // sigma(ti)
    delta_x1 = SpaceStepHW1dG(delta_t1, sigma);//SpaceS
tepHW1dG(delta_t1, a, sigma);

    sigma = Current_VolatilityHW1dG(ModelParam, GET(
Meth->t, i+1)); // sigma(ti+1)
    delta_x2 = SpaceStepHW1dG(delta_t2, sigma);//SpaceS
tepHW1dG(delta_t2, a, sigma);

    beta_x = delta_x1 / delta_x2;

    // Loop over the node at the time i
    for(j = jminprev ; j<= jmaxprev ; j++)
    {
        k = intapprox(j*beta_x*exp(-delta_t2 * mean_rev
ersion)); //h index of the middle node emanating from (i,j)

        // Probability to go from (i,j) to (i+1,k+1)
with an UP movement
        Pup = ProbaUpHW1dG(j, k, delta_t2, beta_x, mea
n_reversion);
        // Probability to go from (i,j) to (i+1,k) wit
```

```
    h a Middle movement
            Pmiddle = ProbaMiddleHW1dG(j, k, delta_t2, bet
    a_x, mean_reversion);
             // Probability to go from (i,j) to (i+1,k-1)
    with a Down movement
            Pdown = 1 - Pup - Pmiddle;

            current_rate = j * delta_x1 + GET(Meth->alpha,
    i); // r(i,j)

            LET(OptionPriceVect1,j-jminprev) = exp(-
    current_rate*delta_t2) * ( Pup * GET(OptionPriceVect2, k+1-jmin)
    + Pmiddle * GET(OptionPriceVect2, k-jmin) + Pdown * GET(
    OptionPriceVect2, k-1-jmin)); // Backward computation of the bo
    nd price
        }

        pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);
     // Copy OptionPriceVect1 in OptionPriceVect2

    } // END of the loop on i (time)
}

// To locate the date s inf the tree. t[IndexTime(s)-1]< s
    <= t[IndexTime(s)]
int IndexTimeHW1dG(TreeHW1dG *Meth, double s)
{
  int i=0;

  if(Meth->t==NULL) {printf("FATALE ERREUR, PAS DE GRILLE
    DE TEMPS !");}
  else
    {
      while(GET(Meth->t, i)<s && i<=Meth->Ngrid)
        {
          i++;
        }
    }
  return i;
}
```

```
// Return Delta_x(i)
double SpaceStepHW1dG(double delta_t, double sigma)
{
    return sigma * sqrt(3 * delta_t);
}

// Probability to go from (i, j) to (i, k+1) with an Up mov
    ement
double ProbaUpHW1dG(int j, int k, double delta_t2,double
    beta_x, double mean_reversion) // beta_x = deltax1/deltax2
{
    double alpha;

    alpha = j * beta_x * exp(-mean_reversion * delta_t2) -
    k;

    return 1./6. + alpha*(alpha + 1)/2;
}

// Probability to go from (i, j) to (i, k) with a Middle
    movement
double ProbaMiddleHW1dG(int j, int k, double delta_t2,
    double beta_x, double mean_reversion)
{
    double alpha;

    alpha = j * beta_x * exp(-mean_reversion * delta_t2) -
    k;

    return 2./3. - alpha*alpha;
}

// Probability to go from (i, j) to (i, k-1) with a Down
    movement
double ProbaDownHW1dG(int j, int k, double delta_t2,
    double beta_x, double mean_reversion)
{
    double alpha;

    alpha = j * beta_x * exp(-mean_reversion * delta_t2) -
    k;
```

```
    return 1./6. + alpha*(alpha - 1)/2;
}

int DeleteTimegridHW1dG(struct TreeHW1dG *Meth)
{
  pnl_vect_free(&(Meth->t));
  return 1;
}

int DeleteTreeHW1dG(struct TreeHW1dG* Meth)
{

  pnl_vect_int_free(&(Meth->Jmaximum));
  pnl_vect_int_free(&(Meth->Jminimum));

  pnl_vect_free(&(Meth->alpha));

  DeleteTimegridHW1dG(Meth);
  return 1;
}

int DeletModelHW1dG(struct ModelHW1dG* HW1dG)
{

  pnl_vect_free(&(HW1dG->TimeGrid));
  pnl_vect_free(&(HW1dG->ShortRateVolGrid));

  return 1;
}


#endif //PremiaCurrentVersion
```

References