Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_mathtools.h" // To use the function "pnl_iround"
#include "math/read_market_zc/InitialYieldCurve.h"
#include "TreeHW2D.h"

int SetTimegridHW2D_Cap(TreeHW2D *Meth, int NtY, double T_intermediate, double T
{
  int i;
  double delta_time, delta_time1;
  int n, m;

  delta_time = periodicity / NtY;

  n = (int)((T_final - T_intermediate) / periodicity + 0.1);

  m = (int) floor(T_intermediate / delta_time);

  delta_time1 = 0.;
  if (m != 0) delta_time1 = T_intermediate / m;

  Meth->Tf = T_final;
  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; // Discretization of [0, T_intermediate
    }

  for (i = m + 1; i <= m + n * NtY + 1; i++)
    {
```

```
      LET(Meth->t, i) = T_intermediate + (i - m) * delta_time; // Discretization
    }

  return OK;
}

// Construction of the time grid
int SetTimegridHW2D(TreeHW2D *Meth, int NbrTimeSteps, double T)
{
  int i;
  double delta_time;

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

  Meth->t = pnl_vect_create(NbrTimeSteps + 2); // time discretisation from 0 to

  delta_time = T / (double)NbrTimeSteps;

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

  return OK;
}

// Build the matrix 3x3 of probabilities
void BuildProbasMatrixHW2D(TreeHW2D *Meth, double eta_over_deltau, double eta_ov
{
  double pu, pm, pd, qu, qm, qd, epsilon;

  pu = ProbaUpHW2D(eta_over_deltay);
  pm = ProbaMiddleHW2D(eta_over_deltay);
  pd = 1 - pu - pm;

  qu = ProbaUpHW2D(eta_over_deltau);
  qm = ProbaMiddleHW2D(eta_over_deltau);
  qd = 1 - qu - qm;

  if (correlation_u_y <= 0)
```

```
{
  epsilon = -correlation_u_y / 36;
  if (epsilon > pd * qm / 4)
    {
      epsilon = pd * qm / 4;
    }
  if (epsilon > pm * qu / 4)
    {
      epsilon = pm * qu / 4;
    }
  if (epsilon > pm * qd / 4)
    {
      epsilon = pm * qd / 4;
    }
  if (epsilon > pu * qm / 4)
    {
      epsilon = pu * qm / 4;
    }
  if (epsilon > pu * qu)
    {
      epsilon = pu * qu;
    }
  if (epsilon > pd * qd)
    {
      epsilon = pd * qd;
    }

  MLET(Meth->ProbasMatrix, 0, 0) = pd * qd - epsilon;
  MLET(Meth->ProbasMatrix, 1, 0) = pm * qd - 4 * epsilon;
  MLET(Meth->ProbasMatrix, 2, 0) = pu * qd + 5 * epsilon;

  MLET(Meth->ProbasMatrix, 0, 1) = pd * qm - 4 * epsilon;
  MLET(Meth->ProbasMatrix, 1, 1) = pm * qm + 8 * epsilon;
  MLET(Meth->ProbasMatrix, 2, 1) = pu * qm - 4 * epsilon;

  MLET(Meth->ProbasMatrix, 0, 2) = pd * qu + 5 * epsilon;
  MLET(Meth->ProbasMatrix, 1, 2) = pm * qu - 4 * epsilon;
  MLET(Meth->ProbasMatrix, 2, 2) = pu * qu - epsilon;

}
```

```
  else
    {
      epsilon = correlation_u_y / 36;
      if (epsilon > pd * qm / 4)
        {
          epsilon = pd * qm / 4;
        }
      if (epsilon > pm * qd / 4)
        {
          epsilon = pm * qd / 4;
        }
      if (epsilon > pm * qu / 4)
        {
          epsilon = pm * qu / 4;
        }
      if (epsilon > pu * qm / 4)
        {
          epsilon = pu * qm / 4;
        }
      if (epsilon > pd * qu)
        {
          epsilon = pd * qu;
        }
      if (epsilon > pu * qd)
        {
          epsilon = pu * qd;
        }

      MLET(Meth->ProbasMatrix, 0, 0) = pd * qd + 5 * epsilon;
      MLET(Meth->ProbasMatrix, 1, 0) = pm * qd - 4 * epsilon;
      MLET(Meth->ProbasMatrix, 2, 0) = pu * qd - epsilon;

      MLET(Meth->ProbasMatrix, 0, 1) = pd * qm - 4 * epsilon;
      MLET(Meth->ProbasMatrix, 1, 1) = pm * qm + 8 * epsilon;
      MLET(Meth->ProbasMatrix, 2, 1) = pu * qm - 4 * epsilon;

      MLET(Meth->ProbasMatrix, 0, 2) = pd * qu - epsilon;
      MLET(Meth->ProbasMatrix, 1, 2) = pm * qu - 4 * epsilon;
      MLET(Meth->ProbasMatrix, 2, 2) = pu * qu + 5 * epsilon;
    }
}
```

```
// Set the value of uIndexMin, uIndexMax, yIndexMin, yIndexMax and alpha after c
void SetTreeHW2D(TreeHW2D *Meth,  ModelHW2D *ModelParam, ZCMarketData *ZCMarket)
{

  double a , sigma1, b, sigma2, rho, sigma3, rho_y_u;

  double eta_over_delta_u, eta_over_delta_y;

  double delta_y1, delta_y2;
  double delta_u1, delta_u2;
  double delta_t1, delta_t2;

  double beta_u, beta_y, sum_alpha, current_rate;
  int jmin, jmax, jminprev, jmaxprev;
  int kmin, kmax, kminprev, kmaxprev;
  int i, j, k, h, l;
  int index_j, index_k;

  PnlMat *Q1; // Quantity used in the calibration of the tree to the initial yie
  PnlMat *Q2; // Quantity used in the calibration of the tree to the initial yie
  // Q1(j,k) : Price at t=0 of a security that pays 1 if r(i) = r(j,k) the value
  // Q2(j,k) : Price at t=0 of a security that pays 1 if r(i+1) = r(j,k) the val
  Q1 = pnl_mat_create(0, 0);
  Q2 = pnl_mat_create(0, 0);

  Meth->ProbasMatrix = pnl_mat_create(3, 3);
  Meth->alpha = pnl_vect_create(Meth->Ngrid + 1);

  Meth->uIndexMin = pnl_vect_int_create(Meth->Ngrid + 1);
  Meth->uIndexMax = pnl_vect_int_create(Meth->Ngrid + 1);

  Meth->yIndexMin = pnl_vect_int_create(Meth->Ngrid + 1);
  Meth->yIndexMax = pnl_vect_int_create(Meth->Ngrid + 1);

  ///*******************Parameters of the processes r, u and y ****************
  a = (ModelParam->rMeanReversion);
  sigma1 = (ModelParam->rVolatility);

  b = (ModelParam->uMeanReversion);
  sigma2 = (ModelParam->uVolatility);
```

```
rho = (ModelParam->correlation);

sigma3 = sqrt(sigma1 * sigma1 + sigma2 * sigma2 / ((b - a) * (b - a)) + 2 * rh
rho_y_u = (rho * sigma1 + sigma2 / (b - a)) / sigma3; // correlation between u

///*************** Initialisation of the vectors yIndexMin, yIndexMax, uIndexM
pnl_vect_int_set(Meth->uIndexMin, 0, 0);
pnl_vect_int_set(Meth->uIndexMax, 0, 0);

pnl_vect_int_set(Meth->yIndexMin, 0, 0);
pnl_vect_int_set(Meth->yIndexMax, 0, 0);

///********************** Calcul des alpha(i) **********************///
// Computation of alpha(0)
delta_t2 = GET(Meth->t, 1) - GET(Meth->t, 0); // = t[1] - t[0]
LET(Meth->alpha, 0) = -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t2; /

pnl_mat_resize(Q1, 1, 1);
MLET(Q1, 0, 0) = 1.;

kmin = 0;
kmax = 0;
jmin = 0;
jmax = 0;
delta_t1 = 0.;
for (i = 0; i < Meth->Ngrid ; i++)
  {
    delta_t2 = GET(Meth->t, i + 1) - GET(Meth->t, i);

    delta_y1 = delta_xHW2D(delta_t1, a, sigma3); // delta_y (i)
    delta_y2 = delta_xHW2D(delta_t2, a, sigma3); // delta_y (i+1)

    delta_u1 = delta_xHW2D(delta_t1, b, sigma2); // delta_u (i)
    delta_u2 = delta_xHW2D(delta_t2, b, sigma2); // delta_u (i+1)

    beta_y   = exp(-a * delta_t2) * delta_y1 / delta_y2;
    beta_u   = exp(-b * delta_t2) * delta_u1 / delta_u2;

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

```
jmin     = pnl_iround(jminprev * beta_y) - 1; // jmin := jmin[i+1]
jmax     = pnl_iround(jmaxprev * beta_y) + 1; // jmax := jmax[i+1]

kminprev = kmin; // kminprev := kmin[i]
kmaxprev = kmax; // kmaxprev := kmax[i]
kmin     = pnl_iround(kminprev * beta_u) - 1; // kmin := kmin[i+1]
kmax     = pnl_iround(kmaxprev * beta_u) + 1; // kmax := kmax[i+1]

pnl_vect_int_set(Meth->uIndexMin, i + 1, kmin);
pnl_vect_int_set(Meth->uIndexMax, i + 1, kmax);
pnl_vect_int_set(Meth->yIndexMin, i + 1, jmin);
pnl_vect_int_set(Meth->yIndexMax, i + 1, jmax);

pnl_mat_resize(Q2, jmax - jmin + 1, kmax - kmin + 1); // Q1 :=Q(i,.) et Q2

pnl_mat_set_double(Q2, 0);

for (h = jminprev ; h <= jmaxprev ; h++) //(i,h,l) -> (i+1, j +/- 1, k +/-
  {
    for (l = kminprev ; l <= kmaxprev ; l++)
      {
        current_rate = GET(Meth->alpha, i) + h * delta_y1 - l * delta_u1 /

        j = pnl_iround(h * beta_y); // j : index of the middle node moving
        k = pnl_iround(l * beta_u); // k : index of the middle node moving

        eta_over_delta_y = h * beta_y - j;
        eta_over_delta_u = l * beta_u - k;

        BuildProbasMatrixHW2D(Meth, eta_over_delta_u, eta_over_delta_y, rh

        for (index_j = -1 ; index_j <= 1 ; index_j++) // loop over the 9 n
          {
            for (index_k = -1 ; index_k <= 1 ; index_k++)
              {
                MLET(Q2, j - jmin + index_j, k - kmin + index_k) += MGET(Q
              }
          }
      }

  } // End computation of Q(i+1,.,.) used in the calculation of alpha(i+1)
```

```
      delta_t1 = delta_t2;

      delta_t2 = GET(Meth->t, i + 2) - GET(Meth->t, i + 1);
      sum_alpha = 0;
      for (j = jmin ; j <= jmax ; j++)
        {
          for (k = kmin; k <= kmax ; k++)
            {
              sum_alpha += MGET(Q2, j - jmin, k - kmin) * exp((-j * delta_y2 + k
            }
        }

      LET(Meth->alpha, i + 1) = log(sum_alpha / BondPrice(GET(Meth->t, i + 2), Z

      pnl_mat_clone(Q1, Q2); // Copy Q2 in Q1 (ie : copy Q(i+1) in Q(i))

    } // FIN boucle sur i

  pnl_mat_free(&Q1);
  pnl_mat_free(&Q2);

}// END of the function SetTreeHW2D

void BackwardIterationHW2D(TreeHW2D *Meth, ModelHW2D *ModelParam, ZCMarketData *
{
  double a , sigma1, b, sigma2, rho, sigma3, rho_y_u;

  int jmin, kmin; // jmin[i+1], jmax[i+1]
  int jminprev, jmaxprev, kminprev, kmaxprev; // jmin[i], jmax [i]
  int i, j, k, h, l; // i = represents the time index. h, l, j, k represents the
  int index_j, index_k; // represents the nine nodes emanating from every node.

  double eta_over_delta_u, eta_over_delta_y;
  double delta_y1, delta_y2; // delta_y1 = space step of the process y at time i
  double delta_u1, delta_u2; // delta_u1 = space step of the process u at time i
  double delta_t1, delta_t2; // time step
  double beta_u, beta_y;     // Quantity used in the compuation of the probabili

  double current_rate;
  double Q2Value;
```

```
///*******************Parameters of the processes r, u and y; y = (r-alpha) +
a = (ModelParam->rMeanReversion);
sigma1 = (ModelParam->rVolatility);

b = (ModelParam->uMeanReversion);
sigma2 = (ModelParam->uVolatility);

rho = (ModelParam->correlation);

sigma3 = sqrt(sigma1 * sigma1 + sigma2 * sigma2 / ((b - a) * (b - a)) + 2 * rh
rho_y_u = (rho * sigma1 + sigma2 / (b - a)) / sigma3; // correlation between y

jminprev = pnl_vect_int_get(Meth->yIndexMin, index_last);  // jmin(index_last)
jmaxprev = pnl_vect_int_get(Meth->yIndexMax, index_last);  // jmax(index_last)
kminprev = pnl_vect_int_get(Meth->uIndexMin, index_last);  // kmin(index_last)
kmaxprev = pnl_vect_int_get(Meth->uIndexMax, index_last);  // kmax(index_last)

///** Backward computation of the option price from "index_last-1" to "index_f
for (i = index_last - 1; i >= index_first; i--)
  {
    jmin = jminprev; // jmin := jmin(i+1)
    kmin = kminprev; // jmin := jmin(i+1)

    jminprev = pnl_vect_int_get(Meth->yIndexMin, i);  // jmin(i)
    jmaxprev = pnl_vect_int_get(Meth->yIndexMax, i);  // jmax(i)
    kminprev = pnl_vect_int_get(Meth->uIndexMin, i);  // kmin(i)
    kmaxprev = pnl_vect_int_get(Meth->uIndexMax, i);  // kmax(i)

    pnl_mat_resize(OptionPriceMat1, jmaxprev - jminprev + 1, kmaxprev - kminpr

    delta_t1 = GET(Meth->t, i) - GET(Meth->t, MAX(i - 1, 0));
    delta_t2 = GET(Meth->t, i + 1) - GET(Meth->t, i);

    delta_y1 = delta_xHW2D(delta_t1, a, sigma3); // delta_y (i)
    delta_y2 = delta_xHW2D(delta_t2, a, sigma3); // delta_y (i+1)

    delta_u1 = delta_xHW2D(delta_t1, b, sigma2); // delta_u (i)
    delta_u2 = delta_xHW2D(delta_t2, b, sigma2); // delta_u (i+1)

    beta_y   = exp(-a * delta_t2) * delta_y1 / delta_y2;
```

```
      beta_u   = exp(-b * delta_t2) * delta_u1 / delta_u2;

      // Loop over the node at the time i
      for (h = jminprev ; h <= jmaxprev ; h++) /// (i,h,l) -> (i+1, j +/- 1, k +
        {
          for (l = kminprev ; l <= kmaxprev ; l++)
            {
              current_rate = GET(Meth->alpha, i) + h * delta_y1 - l * delta_u1 /

              j = pnl_iround(h * beta_y); // j index of the middle node emanatin
              k = pnl_iround(l * beta_u); // k index of the middle node emanatin

              eta_over_delta_y = h * beta_y - j;
              eta_over_delta_u = l * beta_u - k;

              BuildProbasMatrixHW2D(Meth, eta_over_delta_u, eta_over_delta_y, rh

              Q2Value = 0;
              // Loop over the nodes at i+1 emanating from (i,h,l)
              for (index_j = -1 ; index_j <= 1 ; index_j++)
                {
                  for (index_k = -1 ; index_k <= 1 ; index_k++)
                    {
                      Q2Value += MGET(OptionPriceMat2, j - jmin + index_j, k - k
                    }
                }

              MLET(OptionPriceMat1, h - jminprev, l - kminprev) = exp(-current_r
            }
        }
      // Copy OptionPriceMat1 in OptionPriceMat2
      pnl_mat_clone(OptionPriceMat2, OptionPriceMat1);

    } // END of the loop on i
}


int indiceTimeHW2D(TreeHW2D *Meth, double s)  // To locate the date s inf the tr
{
  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 - 1;
}

double delta_xHW2D(double delta_t, double a, double sigma) // Return the step (f
{
  return sigma * sqrt(1.5 * (1 - exp(-2 * a * delta_t)) / a);
}

double ProbaUpHW2D(double x)     // x : eta_ijk/delta_xHW2D(i+1) avec les notati
{
  return (1.0 / 6.0 + x * x / 2 + x / 2);
}

double ProbaMiddleHW2D(double x) // x : eta_ijk/delta_xHW2D(i+1) avec les notati
{
  return (2.0 / 3.0 - x * x);
}

double ProbaDownHW2D(double x)  // x : eta_ijk/delta_xHW2D(i+1) avec les notatio
{
  return (1.0 / 6.0 + x * x / 2 - x / 2);
}

int DeleteTimegridHW2D(TreeHW2D *Meth)
{
  pnl_vect_free(&(Meth->t));
  return 1;
}
```

```
int DeleteTreeHW2D(TreeHW2D *Meth)
{

  pnl_vect_int_free(&(Meth->uIndexMin));
  pnl_vect_int_free(&(Meth->uIndexMax));

  pnl_vect_int_free(&(Meth->yIndexMin));
  pnl_vect_int_free(&(Meth->yIndexMax));

  pnl_vect_free(&(Meth->alpha));
  pnl_mat_free(&(Meth->ProbasMatrix));

  DeleteTimegridHW2D(Meth);

  return 1;
}


#endif //PremiaCurrentVersion
```