Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h" // To use the function "pnl_iround"
#include "TreeCIRpp1D.h"

#define INC 1.0e-5

// Construction of a time grid for a Cap/Floor
// For a Cap/Floor with first resest date T0, payements at T1, T2,..., Tn, with
// The TimeGrid contains NtY steps in each interval [Ti, Ti+1] and an equivalent
int SetTimegridCapCIRpp1D(TreeCIRpp1D *Meth, int NtY, double current_date, doubl
{
  int i;
  double delta_time, delta_time1;
  int i_current_date, n, m;

  delta_time = periodicity / NtY;

  n = (int)((S0 - T0) / periodicity + 0.1);
  m = (int)(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; // Discretization of [0, T0]
    }

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

```
    {
      LET(Meth->t, i) = T0 + (i - m) * delta_time; // Discretization of ]T0, S0]
    }

  i_current_date = (int) floor(current_date / delta_time);

  if ((i_current_date > 0) && ((GET(Meth->t, i_current_date + 1) - current_date)
    {
      LET(Meth->t, i_current_date) = current_date;
    }

  return i_current_date;
}

//Construction of the time grid
int SetTimegridZCbondCIRpp1D(TreeCIRpp1D *Meth, int n, double current_date, doub
{
  int i;
  double delta_time;
  int i_current_date, i_T;

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

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

  delta_time = S / n;

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

  i_current_date = (int) ceil(current_date / delta_time);

  if ((i_current_date > 0) && (i_current_date < n) && ((GET(Meth->t, i_current_d
    {
      LET(Meth->t, i_current_date) = current_date;
    }

  i_T = (int) ceil(T / delta_time);
```

```
  if ((i_T > 0) && (i_T < n) && ((GET(Meth->t, i_T - 1) - T) > delta_time * INC)
    {
      LET(Meth->t, i_T) = T;
    }

  return i_current_date;
}


// Construction of the time grid
int SetTimegridCIRpp1D(TreeCIRpp1D *Meth, int n, double current_date, double T)
{
  int i;
  double delta_time;
  int i_current_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_current_date = (int) floor(current_date / delta_time);

  if ((i_current_date > 0) && ((GET(Meth->t, i_current_date + 1) - current_date)
    {
      LET(Meth->t, i_current_date) = current_date;
    }

  return i_current_date;
}


void SetTreeCIRpp1D(TreeCIRpp1D *Meth, ModelCIRpp1D *ModelParam, ZCMarketData *Z
```

```
{
  double a, b, sigma;

  double sum_alpha;

  double delta_t, sqrt_delta_t;

  double current_rate, current_x, next_x, R_x, x_middle;
  int i, h;
  int NumberNode1, NumberNode2, index;

  double bc, be;

  PnlVect *Probas;
  PnlVect *Q1; // Quantity used to calibrate the tree to the initial yield curve
  PnlVect *Q2; // Quantity used to calibrate the tree to the initial yield curve

  Meth->alpha = pnl_vect_create(Meth->Ngrid + 1);
  Meth->Xmin  = pnl_vect_create(Meth->Ngrid + 1);
  Meth->Xmax  = pnl_vect_create(Meth->Ngrid + 1);

  Probas = pnl_vect_create(3);
  Q1 = pnl_vect_create(3);
  Q2 = pnl_vect_create(1);

  ///********* Model parameters *********///
  a = (ModelParam->MeanReversion);
  b = (ModelParam->LongTermMean);
  sigma = (ModelParam->Volatility);
  current_x = ModelParam->Initialx0; // x(0)

  // Calcul de alpha(0) et alpha(1)
  delta_t = GET(Meth->t, 1) - GET(Meth->t, 0); // = t[1] - t[0]
  sqrt_delta_t = sqrt(delta_t);

  be = current_x / (sqrt_delta_t *floor(current_x / sqrt(1.5 * delta_t)));
  bc = current_x / (sqrt_delta_t *floor(current_x / sqrt(1.5 * delta_t + 1)));

  if (fabs(bc - sqrt(1.5)) < fabs(be - sqrt(1.5)))
    {
      (Meth->bb) = bc;
```

```
  }
else
  {
    (Meth->bb) = be;
  }

Meth->delta_x = (Meth->bb) * sqrt_delta_t;

LET(Meth->Xmin, 0) = current_x;
LET(Meth->Xmax, 0) = current_x;

current_rate = -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t;

LET(Meth->alpha, 0) = current_rate - R(current_x, sigma); // alpha(0) = -log(P

/// Passage de i=0 a i=1
LET(Meth->Xmin, 1) = MiddleNode(Meth, 0, a, b, sigma, current_x, sqrt_delta_t,
LET(Meth->Xmax, 1) = MiddleNode(Meth, 0, a, b, sigma, current_x, sqrt_delta_t,

LET(Q1, 0) = GET(Probas, 0) * exp(- current_rate * delta_t); // Q(1,-1) Down
LET(Q1, 1) = GET(Probas, 1) * exp(- current_rate * delta_t); // Q(1, 0) Middle
LET(Q1, 2) = GET(Probas, 2) * exp(- current_rate * delta_t); // Q(1,-2) Up


sum_alpha = 0;
for (h = 0 ; h <= 2 ; h++)
  {
    next_x = x_value(1, h, Meth);
    R_x = R(next_x, sigma);
    sum_alpha += GET(Q1, h) * exp(-R_x * delta_t);
  }

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

for (i = 1; i < Meth->Ngrid ; i++)
  {
    LET(Meth->Xmin, i + 1) = MiddleNode(Meth, i, a, b, sigma, GET(Meth->Xmin,
    LET(Meth->Xmax, i + 1) = MiddleNode(Meth, i, a, b, sigma, GET(Meth->Xmax,

    NumberNode1 = (int)((GET(Meth->Xmax, i) - GET(Meth->Xmin, i)) / (Meth->del
    NumberNode2 = (int)((GET(Meth->Xmax, i + 1) - GET(Meth->Xmin, i + 1)) / (M
```

```
      pnl_vect_resize(Q2, NumberNode2 + 1); // Q1 :=Q(i,.) et Q2 : =Q(i+1,.)
      pnl_vect_set_double(Q2, 0);

      for (h = 0 ; h <= NumberNode1 ; h++)
        {
          current_x = x_value(i, h, Meth);
          current_rate = R(current_x, sigma) + GET(Meth->alpha, i);

          x_middle = MiddleNode(Meth, i, a, b, sigma, current_x, sqrt_delta_t, P

          index = (int)((x_middle - GET(Meth->Xmin, i + 1)) / (Meth->delta_x) +

          LET(Q2, index + 1) += GET(Q1, h) * GET(Probas, 2) * exp(-current_rate

          LET(Q2, index)   += GET(Q1, h) * GET(Probas, 1) * exp(-current_rate *

          LET(Q2, index - 1) += GET(Q1, h) * GET(Probas, 0) * exp(-current_rate

        } //END loop over h

      sum_alpha = 0;
      for (h = 0 ; h <= NumberNode2 ; h++)
        {
          next_x = x_value(i + 1, h, Meth);
          R_x = R(next_x, sigma);
          sum_alpha += GET(Q2, h) * exp(-R_x * delta_t);
        }


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

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

    }

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

}// FIN de la fonction SetTreeCIRpp1D
```

```
double x_value(int i, int h, TreeCIRpp1D *Meth)
{
  return (GET(Meth->Xmin, i) + h * (Meth->delta_x));
}

double R(double x, double sigma)
{
  if (x <= 0)
    {
      return 0;
    }
  else
    {
      return SQR(x * sigma) / 4;
    }
}


double MiddleNode(TreeCIRpp1D *Meth, int i, double a, double b, double sigma, do
{
  int j;

  double x_m, mean, x_up, epsilon;

  epsilon = 1e-10;

  if (current_x <= epsilon)
    {
      j = (int) ceil(2 * sqrt(a * b) / (sigma * (Meth->bb)) - 1);
      if (j < 1)
        {
          j = 1;
        }

      //j = 1;
      x_up = current_x + (j + 1) * (Meth->delta_x);

      LET(Probas, 2) = a * b * SQR(sqrt_delta_t) / R(x_up, sigma); // Up
      LET(Probas, 1) = 0;
      LET(Probas, 0) = 1 - GET(Probas, 2);
```

```
    }

  else
    {
      mean = (0.5 * a * (4 * b / SQR(sigma) - SQR(current_x)) - 0.5) / current_x

      j = (int) floor(mean * sqrt_delta_t / (Meth->bb) + 1 / SQR((Meth->bb)));

      LET(Probas, 2) = 1 / (2 * SQR((Meth->bb))) - 0.5 * j + mean * sqrt_delta_t
      LET(Probas, 1) = 1 / (2 * SQR((Meth->bb))) + 0.5 * j - mean * sqrt_delta_t
      LET(Probas, 0) = 1 - GET(Probas, 1) - GET(Probas, 2);
    }

  x_m = current_x + j * (Meth->delta_x);

  return x_m;
}

int indiceTimeCIRpp1D(TreeCIRpp1D *Meth, double s) // To locate the date s inf t
{
  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;
}


int DeleteTimegridCIRpp1D(struct TreeCIRpp1D *Meth)
{
  pnl_vect_free(&(Meth->t));
  return 1;
```

```
}

int DeleteTreeCIRpp1D(struct TreeCIRpp1D *Meth)
{

  pnl_vect_free(&(Meth->Xmax));
  pnl_vect_free(&(Meth->Xmin));

  pnl_vect_free(&(Meth->alpha));

  DeleteTimegridCIRpp1D(Meth);
  return 1;
}


#endif //PremiaCurrentVersion
```