Help

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

#include <stdio.h>
#include <stdlib.h>
#include <
href../../../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_mathtools.h"
#include "
href../../../../common/math/InterestRateModelTree/TreeLRS1D/TreeLRS1D_h_src.pdfT


// 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 SetTimegridCapLRS1D(TreeLRS1D *Meth, int NtY, double current_date, double T0
{
  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) 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; // 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 SetTimegridZCbondLRS1D(TreeLRS1D *Meth, int n, double current_date, double T
{
  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;
}

int SetTimegridLRS1D(TreeLRS1D *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;
}


double phi_value(TreeLRS1D *Meth, int i, int h, int j) // i>=1 , j=0,1,2
{
```

```
  if (h == 0 || h == 1)
    {
      return GET(Meth->phi, SQR(i - 1) + j);
    }

  else
    {
      return GET(Meth->phi, SQR(i - 1) + (h - 2 + j));
    }
}

/// i>1
/// h number of the box
double Interpolation(TreeLRS1D *Meth, int i, int h, PnlVect *OptionPriceVect2, d
{
  double Phi0, Phi1, Phi2, theta0, theta1, theta2;

  double epsilon;

  epsilon = 1e-6;

  if (h == 0)
    {
      return GET(OptionPriceVect2, 0);
    }

  else if (h == 2 * i)
    {
      return GET(OptionPriceVect2, OptionPriceVect2->size - 1);
    }

  else if (h == 1)
    {
      Phi1 = phi_value(Meth, i, h, 0);
      Phi0 = phi_value(Meth, i, h, 1);

      if (fabs(Phi1 - Phi0) < epsilon)
        {
          return GET(OptionPriceVect2, 1);
        }
```

```
    theta1 = (phi_next - Phi0) / (Phi1 - Phi0);
    theta0 = (phi_next - Phi1) / (Phi0 - Phi1);

    return theta1 * GET(OptionPriceVect2, 1) + theta0 * GET(OptionPriceVect2,
  }

else if (h == 2 * i - 1)
  {
    Phi1 = phi_value(Meth, i, h, 0);
    Phi0 = phi_value(Meth, i, h, 1);

    if (fabs(Phi1 - Phi0) < epsilon)
      {
        return GET(OptionPriceVect2, 1);
      }

    theta1 = (phi_next - Phi0) / (Phi1 - Phi0);
    theta0 = (phi_next - Phi1) / (Phi0 - Phi1);

    return theta1 * GET(OptionPriceVect2, 3 * (h - 1)) + theta0 * GET(OptionPr
  }

else
  {
    Phi2 = phi_value(Meth, i, h, 0);
    Phi1 = phi_value(Meth, i, h, 1);
    Phi0 = phi_value(Meth, i, h, 2);

    if (fabs(Phi2 - Phi0) < epsilon)
      {
        theta2 = 0;
        theta1 = 0;
        theta0 = 1;
      }

    else if (fabs(Phi1 - Phi0) < epsilon)
      {
        theta2 = (phi_next - Phi1) / (Phi2 - Phi1);
        theta1 = (phi_next - Phi2) / (Phi1 - Phi2);
        theta0 = 0;
      }
```

```
      else if (fabs(Phi2 - Phi1) < epsilon)
        {
          theta2 = 0;
          theta1 = (phi_next - Phi0) / (Phi1 - Phi0);
          theta0 = (phi_next - Phi1) / (Phi0 - Phi1);
        }

      else
        {
          theta2 = (phi_next - Phi1) * (phi_next - Phi0) / ((Phi2 - Phi1) * (Phi
          theta1 = (phi_next - Phi0) * (phi_next - Phi2) / ((Phi1 - Phi0) * (Phi
          theta0 = (phi_next - Phi1) * (phi_next - Phi2) / ((Phi0 - Phi1) * (Phi
        }

      return theta2 * GET(OptionPriceVect2, 3 * (h - 1)) + theta1 * GET(OptionPr
    }

}


double MeanPrice(TreeLRS1D *Meth, int i, int h, PnlVect *OptionPriceVect2)
{
  if (h == 0)
    {
      return GET(OptionPriceVect2, 0);
    }

  else if (h == 2 * i)
    {
      return GET(OptionPriceVect2, OptionPriceVect2->size - 1);
    }

  else if (h == 1)
    {
      return (GET(OptionPriceVect2, 1) + GET(OptionPriceVect2, 2)) / 2;
    }

  else if (h == 2 * i - 1)
    {
      return (GET(OptionPriceVect2, 3 * (h - 1)) + GET(OptionPriceVect2, 3 * (h
```

```
    }

  else
    {
      return (GET(OptionPriceVect2, 3 * (h - 1)) + GET(OptionPriceVect2, 3 * (h
    }

}


int number_phi_in_box(int i, int h) // h : box number; i>1
{
  if (i == 1)
    {
      return 1;
    }

  else if (h == 0 || h == 2 * i)
    {
      return 1;
    }

  else if (h == 1 || h == 2 * i - 1)
    {
      return 2;
    }

  else
    {
      return 3;
    }

}

int index_tree(int i, int h, int j) // h : box number; i>1
{
  if (h == 0)
    {
      return 0;
    }
```

```
  else if (h == 2 * i)
    {
      return (6 * i - 4);
    }

  else if (h == 1)
    {
      return (h + j);
    }

  else
    {
      return (3 * (h - 1) + j);
    }

}

void SetTreeLRS1D(TreeLRS1D *Meth, ModelLRS1D *ModelParam, ZCMarketData *ZCMarke
{
  double sigma, rho, kappa, lambda;

  int i, h, iter;

  double delta_t, sqrt_delta_t, delta_y;

  double y_ih, y_00, phi_next;
  double conditional_expect_phi_ih, epsilon;
  PnlVect *proba_from_ih;
  PnlVect *proba_reaching_node_1;
  PnlVect *proba_reaching_node_2;

  PnlVect *conditional_expect_phi_1;
  PnlVect *conditional_expect_phi_2;



  ///********* Model parameters *********///
  kappa = (ModelParam->Kappa);
  sigma = (ModelParam->Sigma);
  rho = (ModelParam->Rho);
  lambda = (ModelParam->Lambda);
```

```
///************  Construction de l'arbre ************///
Meth->phi = pnl_vect_create(SQR(Meth->Ngrid));

delta_t = GET(Meth->t, 1) - GET(Meth->t, 0);
sqrt_delta_t = sqrt(delta_t);

y_00 = r_to_y(ModelParam, -log(BondPrice(GET(Meth->t, 1), ZCMarket)) / delta_t

proba_reaching_node_1 = pnl_vect_create(1);
proba_reaching_node_2 = pnl_vect_create(1);

conditional_expect_phi_1 = pnl_vect_create(1);
conditional_expect_phi_2 = pnl_vect_create(1);
proba_from_ih = pnl_vect_create(3);
epsilon = 1e-20;

LET(proba_reaching_node_1, 0) = 1.0;
LET(conditional_expect_phi_1, 0) = 0;

y_ih = y_00;

iter = 0;
for (i = 0; i < Meth->Ngrid ; i++)
  {
    pnl_vect_resize(proba_reaching_node_2, 2 * (i + 1) + 1);
    pnl_vect_resize(conditional_expect_phi_2, 2 * (i + 1) + 1);

    pnl_vect_set_double(proba_reaching_node_2, 0);
    pnl_vect_set_double(conditional_expect_phi_2, 0);

    delta_t = GET(Meth->t, i + 1) - GET(Meth->t, i);
    sqrt_delta_t = sqrt(delta_t);
    delta_y = lambda * sqrt_delta_t;

    for (h = 0; h <= 2 * i; h++)
      {
        y_ih = y_00 + (i - h) * delta_y; // y(i,h)
        conditional_expect_phi_ih = GET(conditional_expect_phi_1, h); // epera

        // Calcul des probabilites
```

```
          probabilities(GET(Meth->t, i), y_ih, conditional_expect_phi_ih, lambda

          // Calcul de proba_reaching_node (i+1)
          LET(proba_reaching_node_2, h)   += GET(proba_reaching_node_1, h) * GET
          LET(proba_reaching_node_2, h + 1) += GET(proba_reaching_node_1, h) * G
          LET(proba_reaching_node_2, h + 2) += GET(proba_reaching_node_1, h) * G

          // Calcul de conditional_expect_phi (i+1)
          phi_next = conditional_expect_phi_ih * (1 - 2 * kappa * delta_t) + SQR

          LET(Meth->phi, iter) = phi_next;
          iter++;

          LET(conditional_expect_phi_2, h)   += phi_next * GET(proba_reaching_no
          LET(conditional_expect_phi_2, h + 1) += phi_next * GET(proba_reaching_
          LET(conditional_expect_phi_2, h + 2) += phi_next * GET(proba_reaching_
        }

      for (h = 1; h < proba_reaching_node_2->size; h++)
        {
          if (GET(proba_reaching_node_2, h) > epsilon) LET(conditional_expect_ph
          else LET(conditional_expect_phi_2, h) = GET(conditional_expect_phi_2,
        }
      //pnl_vect_div_vect_term(conditional_expect_phi_2, proba_reaching_node_2);

      pnl_vect_clone(conditional_expect_phi_1, conditional_expect_phi_2); // Cop
      pnl_vect_clone(proba_reaching_node_1, proba_reaching_node_2); // Copier Q2

    }

  pnl_vect_free(&conditional_expect_phi_1);
  pnl_vect_free(&conditional_expect_phi_2);
  pnl_vect_free(&proba_reaching_node_1);
  pnl_vect_free(&proba_reaching_node_2);
  pnl_vect_free(&proba_reaching_node_2);
  pnl_vect_free(&proba_from_ih);

}// FIN de la fonction SetTreeLRS1D

double r_to_y(ModelLRS1D *ModelParam, double r)
{
```

```
  if (ModelParam->Rho == 1)
    {
      return log(r) / (ModelParam->Sigma);
    }

  else
    {
      return pow(r, (1 - ModelParam->Rho)) / ((ModelParam->Sigma) * (1 - ModelPa
    }
}

double y_to_r(ModelLRS1D *ModelParam, double y)
{
  if (ModelParam->Rho == 1)
    {
      return exp(y * ModelParam->Sigma);
    }

  else
    {
      return pow(y * (1 - ModelParam->Rho) * ModelParam->Sigma, 1 / (1 - ModelPa
    }
}

/*Compute m, mean of Y=log(r/sigma)*/
double mean(double time, double Y, double Phi, ZCMarketData *ZCMarket, ModelLRS1
{
  double d_f0_t, f0_t, P0_t, P0_t_plus, P0_t_minus;

  double sigma, kappa, rho;

  double m, r, dt;

  double epsilon;

  kappa = (ModelParam->Kappa);
  sigma = (ModelParam->Sigma);
  rho = (ModelParam->Rho);

  r = y_to_r(ModelParam, Y);
```

```
  /*Computation pure discount bond*/

  P0_t = BondPrice(time, ZCMarket);

  epsilon = 1e-5;
  /*Computation of Forward rate and derivate*/
  if (time == 0)
    {
      time = epsilon;
      dt = epsilon;
    }

  else
    {
      dt = epsilon * time;
    }

  P0_t = BondPrice(time, ZCMarket);

  P0_t_plus  = BondPrice(time + dt, ZCMarket);

  P0_t_minus = BondPrice(time - dt, ZCMarket);

  f0_t = -(log(P0_t_plus) - log(P0_t_minus)) / (2 * dt);

  d_f0_t = -(log(P0_t_plus) - 2 * log(P0_t) + log(P0_t_minus)) / (dt * dt);

  m = (kappa * (f0_t - r) + Phi + d_f0_t) / (sigma * pow(r, rho)) - rho * sigma

  return m;
}

void probabilities(double date, double y_ij, double phi_ij, double lambda, doubl
{
  double m_ij, proba_up, proba_middle, epsilon;

  epsilon = 1e-10;
  m_ij = mean(date, y_ij, phi_ij, ZCMarket, ModelParam);

  proba_up  = 1 / (2 * SQR(lambda)) + (m_ij * sqrt_delta_t) / (2 * lambda); // +
```

```
  proba_middle = 1 - 1 / (SQR(lambda)); // - SQR((m_ij*sqrt_delta_t)/lambda);

  if (proba_up < epsilon)
    {
      proba_up = proba_middle;
    }

  if (proba_up > 1)
    {
      proba_up = proba_middle;
    }


  LET(proba_from_ij, 0) = proba_up;

  LET(proba_from_ij, 1) = proba_middle;

  LET(proba_from_ij, 2) = 1 - proba_up - proba_middle;
}

int indiceTimeLRS1D(TreeLRS1D *Meth, double s) // To locate the date s inf the 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;
}

int DeleteTimegridLRS1D(struct TreeLRS1D *Meth)
{
  pnl_vect_free(&(Meth->t));
```

```
  return 1;
}

int DeleteTreeLRS1D(struct TreeLRS1D *Meth)
{

  pnl_vect_free(&(Meth->phi));


  DeleteTimegridLRS1D(Meth);
  return 1;
}


#endif //PremiaCurrentVersion
```