```
   Help
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+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 "TreeShortRate.h"

// Construction of the time grid
int SetTimeGrid(TreeShortRate *Meth, int n, double T)
{
    int i;
    double delta_time;

    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;
    }

    return OK;
}

int SetTimeGrid_Tenor(TreeShortRate *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 = 0;
    if(m!=0) 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;
}

/*Newton algorithm*/
double PhiAlpha(double alpha_i, double delta_t_i, double de
    lta_x_i, int jmin, int jmax, PnlVect* Q, double ZCbondpric
    e, double (*func_model) (double), double (*func_model_der)
    (double))
{
    int j;
    double sum, sum_der, current_rate, current_rate_der;

    sum=0.;
    sum_der=0.;
```

```
    for(j=jmin; j<=jmax; j++)
    {
        current_rate = func_model(alpha_i + j*delta_x_i);
        current_rate_der = func_model_der(alpha_i + j*delt
    a_x_i);

        sum += GET(Q, j-jmin) * exp(-current_rate*delta_t_
    i);
        sum_der += GET(Q, j-jmin) * exp(-current_rate*delt
    a_t_i) * current_rate_der * delta_t_i;
    }

    return ((ZCbondprice-sum)/sum_der);
}

/*Computation of alpha[i] with numerical search*/
double FindAlpha(double alpha_init, double delta_t_i,
    double delta_x_i, int jmin, int jmax, PnlVect* Q, double     ZCbondprice, do
    er) (double))
{
    const double precision = 0.00001;
    double current_alpha, current_phi;
    int j;

    current_alpha = alpha_init;
    j = 0;

    current_phi = PhiAlpha(current_alpha, delta_t_i, delta_
    x_i, jmin, jmax, Q, ZCbondprice, func_model, func_model_d
    er);

    while((fabs(current_phi) > precision) && (j < 50))
    {
        j++;

        current_alpha = current_alpha - current_phi;

        current_phi   = PhiAlpha(current_alpha, delta_t_i,
    delta_x_i, jmin, jmax, Q, ZCbondprice, func_model, func_
    model_der);
    }
```

```
    return current_alpha;
}

void SetTreeShortRate(TreeShortRate* Meth, ModelParameters*
     ModelParam, ZCMarketData* ZCMarket, double (*func_model)
    (double), double (*func_model_der) (double), double (*
    func_model_inv) (double))
{

  double a ;
  double sigma ;

  double Pdown, Pmiddle, Pup, eta_over_deltax;
  double Q2Value;

  double delta_x1, delta_x2;
  double delta_t1, delta_t2;

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

  PnlVect* Q1; // Quantity used to calibrate the tree to th
    e initial yield curve (see the book Brigo&Mercurio page 80)
  PnlVect* Q2; // Quantity used to calibrate the tree to th
    e initial yield curve (see the book Brigo&Mercurio page 80)


  ///********* Model parameters *********///
  a = (ModelParam->MeanReversion);
  sigma = (ModelParam->RateVolatility);

  ///************  Construction of the vector index Jminimu
    m et Jmaximum and calibration of the tree to the zc term
    structure ************///
  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);

// Compute alpha(0) and 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]

current_rate = -log(BondPrice(GET(Meth->t, 1), ZCMarket))
  /delta_t1;

LET(Meth->alpha, 0) = func_model_inv(current_rate);

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

LET(Q1, 0) = Pdown   * exp(- current_rate * delta_t1); //
   Q(1,-1)
LET(Q1, 1) = Pmiddle * exp(- current_rate * delta_t1); //
   Q(1,0)
LET(Q1, 2) = Pup     * exp(- current_rate * delta_t1); //
   Q(1,-2)

delta_x1 = SpaceStep(delta_t1, a, sigma);

jmin =-1; jmax = 1;

ZCbondprice = BondPrice(GET(Meth->t, 2), ZCMarket);

LET(Meth->alpha, 1) = FindAlpha(GET(Meth->alpha, 0), delt
  a_t2, delta_x1, jmin, jmax, Q1, ZCbondprice, func_model,
  func_model_der);
```

```
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]

    delta_x1 = SpaceStep(delta_t1, a, sigma); // SpaceS
  tep (i)
    delta_x2 = SpaceStep(delta_t2, a, sigma); // SpaceS
  tep (i+1)

    beta = exp(-a * delta_t2) * delta_x1 / delta_x2;

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

    jmin = intapprox(jminprev * beta) - 1; // jmin := jm
  in[i+1]
    jmax = intapprox(jmaxprev * beta) + 1; // jmax := jm
  ax[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,.) and
  Q2 : =Q(i+1,.)

    pnl_vect_set_double(Q2, 0);

    /// Computation of the values of Q(i+1,.)
    for (h= jminprev ; h <=jmaxprev ; h++)
    {
        current_rate = func_model(GET(Meth->alpha, i) +
  h*delta_x1);

        j = intapprox(h*beta); //j index of the middle
  node emanating from (i,h)

        eta_over_deltax = h * beta - j;
```

```
      Pup = ProbaUp(eta_over_deltax);           // Prob
    ability to go from (i,h) to (i+1,j+1) with an UP movement
      Pmiddle = ProbaMiddle(eta_over_deltax);   // Prob
    ability to go from (i,h) to (i+1,j) with a Middle movement
      Pdown = 1 - Pup - Pmiddle;                    //
    Probability to go from (i,h) to (i+1,j-1) with a Down movem
    ent

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

     } //END Loop over h

     /// Computation of alpha(i+1)
     delta_t2 = GET(Meth->t, i+2) - GET(Meth->t,i+1);

     ZCbondprice = BondPrice(GET(Meth->t, i+2), ZCMarket);

     LET(Meth->alpha, i+1) = FindAlpha(GET(Meth->alpha, i)
    , delta_t2, delta_x2, jmin, jmax, Q2, ZCbondprice, func_
    model, func_model_der);

     pnl_vect_clone(Q1, Q2);

  }


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

}
```

```
void BackwardIteration(TreeShortRate* Meth, ModelParameters
    * ModelParam, PnlVect* OptionPriceVect1, PnlVect* OptionP
    riceVect2, int index_last, int index_first, double (*func_
    model) (double))
{
    double a ,sigma;

    int jmin; // jmin[i+1], jmax[i+1]
    int jminprev, jmaxprev; // jmin[i], jmax [i]
    int i, j, k; // i = represents the time index. j, k rep
    resents the nodes index

    double eta_over_delta_x;
    double delta_x1, delta_x2; // delta_y1 = space step of
    the process y at time i ; delta_y2 same at time i+1.
    double delta_t1, delta_t2; // time step
    double beta_x;     // quantity used in the compuation
    of the probabilities. it depends only on i.

    double current_rate;

    double Pup, Pmiddle, Pdown;

    ///*******************Parameters of the processes r, u
    and y **********************////
    a = ModelParam->MeanReversion;
    sigma = ModelParam->RateVolatility;

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

    ///** Backward computation of the option price from "
    index_last-1" to "index_first", knowing those at "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); //
jminprev := jmin(i)
    jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i); //
jmaxprev := jmax(i)

    pnl_vect_resize(OptionPriceVect1, jmaxprev-jminprev
 +1);  // OptionPrice1 := price at time i,

    delta_t1 = GET(Meth->t, i) - GET(Meth->t,MAX(i-1, 0
)); // when i=0, delta1=0;
    delta_t2 = GET(Meth->t, i+1) - GET(Meth->t,i);

    delta_x1 = SpaceStep(delta_t1, a, sigma);  // Spac
eStep (i)
    delta_x2 = SpaceStep(delta_t2, a, sigma);  // Spac
eStep (i+1)

    beta_x =  (delta_x1 / delta_x2) * exp(-a*delta_t2);

    for(j = jminprev ; j<= jmaxprev ; j++)
    {
        k= intapprox(j * beta_x); // index of the midd
le node emanating from (i,j)
        eta_over_delta_x = j * beta_x - k; // quantity
used in the compuation of the probabilities Pup, Pmiddle an
d Pdown.

        Pup = ProbaUp(eta_over_delta_x); // Probability
 of an up move from (i,j)
        Pmiddle = ProbaMiddle(eta_over_delta_x); //
Probability of a middle move from (i,j)
        Pdown = 1 - Pup - Pmiddle; // Probability of a
down move from (i,j)

        current_rate = func_model(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));
```

```
        }

      // Copy OptionPrice1 in OptionPrice2
      pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);

    } // END of the loop on i
}

int IndexTime(TreeShortRate *Meth, double s) // To locate
    the date s inf the tree. t[IndexTime(s)-1]< s <= t[IndexT
    ime(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;
}

double SpaceStep(double delta_t, double a, double sigma) //
     Renvoie Delta_x(i)
{
    return sigma * sqrt(1.5 * (1- exp(-2 * a * delta_t)) /
    a );
}

double ProbaUp(double x) // x : eta_ijk/SpaceStep(i+1)
{
    return (1.0/6.0 + x*x/2 + x/2);
}

double ProbaMiddle(double x)
{
```

```
    return (2.0/3.0 - x*x);
}

double ProbaDown(double x)
{
    return (1.0/6.0 + x*x/2 - x/2);
}

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

int DeleteTreeShortRate(struct TreeShortRate* Meth)
{

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

  pnl_vect_free(&(Meth->alpha));

  DeleteTimeGrid(Meth);
  return 1;
}

///*************** Function that defines the model (HW=
    Hull&White, SG=Squared Gaussian, BK=Black&Karasinski)*********
    **********///

//*************** SG ******************//
double func_model_sg1d(double x)
{
    return 0.5*x*x;
}

double func_model_der_sg1d(double x)
{
    return x;
}
```

```
double func_model_inv_sg1d(double r)
{
    return sqrt(2*r);
}

//***************** HW ******************//
double func_model_hw1d(double x)
{
    return x;
}

double func_model_der_hw1d(double x)
{
    return 1;
}

double func_model_inv_hw1d(double r)
{
    return r;
}

//***************** BK ******************//
double func_model_bk1d(double x)
{
    return exp(x);
}

double func_model_der_bk1d(double x)
{
    return exp(x);
}

double func_model_inv_bk1d(double r)
{
    return log(r);
}


#endif //PremiaCurrentVersion
```

References