```
   Help
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "dynamic_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_root.h"


#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)
static int CHK_OPT(Hedging_FreyBackhaus)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(Hedging_FreyBackhaus)(void *Opt, void *Mod, Prici
    ngMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/*
 * Couverture de CDO avec risque de spread et contagion
 * Frey & Backhaus(2007)
 */

typedef struct{
  double r;
  double psi;//paramètre de dépendance entre les défauts
  int m;//nb de noms
  double sp;//spread
  double R;//taux de recouvrement
  double lambda0;//modélise la viabilité des
```

```
  //entreprises du CDO
  double lambda1;//modélise la force de la contagion
  double lambda2;//modélise la tendance du modèle à
  //générer des défauts en cascade
} Param;

typedef struct{

  double r;
  double psi;//paramètre de dépendance entre les défauts
  int m;//nb de noms
  double sp;//spread
  double R;//taux de recouvrement
  double lambda0;//modélise la viabilité des
  //entreprises du CDO
  double lambda1;//modélise la force de la contagion
  double lambda2;//modélise la tendance du modèle à générer
     des défauts en cascade
  double l;
} Param_exp;


//intensite de défaut définie page 4, article de 2007
static double h(double t, int Mt, Param *P)
{
  int m = P-> m;
  double psi = P->psi;
  double sp = P->sp;
  double R = P->R;
  double lambda0=P->lambda0;
  double lambda1=P->lambda1;
  double lambda2=P->lambda2;

  double mu_t=m*(1-exp(-sp/(1-R)*t));
  double num =(Mt>=mu_t)?(Mt-mu_t):0;
  return lambda0*psi+lambda1/lambda2*(exp(lambda2*num/m)-1)
    ;
}

//intensite de défaut définie page 4, article de 2007
static double h_exp(double t, void *params)
```

```
{
  Param_exp *P = (Param_exp *) params;
  int l=P->l;
  int m = P-> m;
  double psi = P->psi;
  double sp = P->sp;
  double R = P->R;
  double lambda0=P->lambda0;
  double lambda1=P->lambda1;
  double lambda2=P->lambda2;

  double mu_t=m*(1-exp(-sp/(1-R)*t));
  double num =(l>=mu_t)?(l-mu_t):0;
  return lambda0*psi+lambda1/lambda2*(exp(lambda2*num/m)-1)
    ;
}


//nombre de dates de paiement jusqu'au temps t inclus, les
//dates de paiement sont données par le vecteur z, attention
//0 n'est pas considéré comme une date de paiement
static int n_t(double t, PnlVect *z)
{
  int i=1;
  int lz=z->size;
  if(t<pnl_vect_get(z,1)) return 0;
  else
    while ((pnl_vect_get(z,i)<=t) && (i<lz)) i++;
  return (i-1);
}

static int subsub(int i)
{
  return 4*i;
}

//on subdivise z en 2 (cette fonction marche)
static void create_sub(PnlVect *sub, double t, double T, Pn
    lVect *z)
{
  int lz=z->size;
```

```
  int nt=n_t(t,z);//nombre de dates de paiement jusqu'au
    temps t
  int M=2*(lz-nt-1);
  int k=0;
  pnl_vect_resize(sub,M+1);
  pnl_vect_set(sub,0,t);
  for(k=0;2*k+1<M;k++)
    {
      pnl_vect_set(sub,2*(k+1),pnl_vect_get(z,nt+1+k));
      pnl_vect_set(sub,2*k+1,(pnl_vect_get(sub,2*k)+pnl_vec
    t_get(z,nt+k+1))/2);
    }

  pnl_vect_set(sub,M,T);
}

static void proba_M_exp(PnlVect *res, double s, double t,
    int Mt, Param *P, PnlMat *work1, PnlMat *work2)
{
  PnlFunc func;
  Param_exp *P_exp;
  int i;
  int N=100;
  double int_res;
  double epsres;
  int m=P->m;
  P_exp=malloc(sizeof(Param_exp));
  memcpy(P_exp,P,sizeof(Param));

  pnl_mat_set_double (work1, 0.);
  for(i=0;i<m-Mt;i++)
    {
      P_exp->l=Mt+i;
      func.function=h_exp;
      func.params=P_exp;
      pnl_integration_GK(&func,t,s,0.000000001,0.0000000001
    ,&int_res,&epsres,&N);
      int_res *= (m-Mt-i);
      pnl_mat_set(work1,i,i,-int_res);
      pnl_mat_set(work1,i,i+1,int_res);
    } //on s'arrête car la dernière ligne de work1 ne conti
```

```
    ent que des zéros
  // on veut la colonne 1 de exp(work1^t) <=> ligne 1 de
    exp(work1)
  pnl_mat_exp(work2,work1);
  pnl_mat_get_row(res,work2,0);
  free(P_exp);
}

//calcul de l'integrale d'une fonction f sur (a,b) par la
//méthode de simpson. sub contient les points de
//discrétisation auxquels est évaluée f (cette fonction mar
    che)
static double integrale_simpson(double a, double b, PnlVec
    t *sub, PnlVect *f)
{
  double res,res1;
  double delta=0.125/6;
  int i;
  //calcul du premier terme, cas particulier pour t
  //quelconque
  res=(pnl_vect_get(sub,4)-pnl_vect_get(sub,0))/12*(pnl_vec
    t_get(f,0)+pnl_vect_get(f,4)+2*pnl_vect_get(f,2)+4*pnl_vec
    t_get(f,3)+4*pnl_vect_get(f,1));
  //ajout des 2 termes extrêmes (en 4 et en T)
  res1=pnl_vect_get(f,4)+pnl_vect_get(f,sub->size-1);
  for(i=2;2*i+1<sub->size;i++)
    {
      res1 += 2*pnl_vect_get(f,2*(i+1))+4*pnl_vect_get(f,2*
    i+1);
    }
  return res1*delta+res;


}

static void build_proba_M(PnlMat *res, PnlVect *subsub,
    double t, int Mt, Param *P)
{
  int i;
  int m=P->m;
  PnlVect *tmp;
```

```
  PnlMat *work1, *work2;
  work1 = pnl_mat_create (m-Mt+1,m-Mt+1);
  work2 = pnl_mat_create (m-Mt+1,m-Mt+1);

  tmp=pnl_vect_create(0);
  pnl_mat_resize(res,subsub->size,m-Mt+1);
  for(i=0;i<subsub->size;i++)
    {
      proba_M_exp(tmp,pnl_vect_get(subsub,i),t,Mt,P,work1,
    work2);
      //proba_M_imp(tmp,pnl_vect_get(subsub,i),t,Mt,P);
      pnl_mat_set_row (res, tmp, i);
    }
  pnl_vect_free(&tmp);
  pnl_mat_free(&work1);
  pnl_mat_free(&work2);
}

//renvoie la matrice des probabilités que l'on ait au moins
     j
// défauts en subsub_t(i) sachant que l'on a Mt
//défauts en t
// res(i,j)=P(T_j<=subsub_t(i)|Mt)
static void loi_T(PnlMat *res, PnlVect *subsub_t, double t,
     int Mt, Param *P, PnlMat *proba_M_t)
{
  int i,j;
  int m=P->m;
  int l_subsub_t=subsub_t->size;
  pnl_mat_resize(res,l_subsub_t,m+1);
  for(j=0;j<=Mt;j++) pnl_mat_set(res,0,j,1);
  for(j=Mt+1;j<=m;j++) pnl_mat_set(res,0,j,0);
  for(i=1;i<l_subsub_t;i++)
    {
      for(j=0;j<=Mt;j++) pnl_mat_set(res,i,j,1);
      for(j=Mt+1;j<=m;j++)
        {
          pnl_mat_set(res,i,j,pnl_mat_get(res,i,j-1)-pnl_
    mat_get(proba_M_t,i,j-Mt-1));
        }
    }
```

```
}

//renvoie P(tau appartienne à ds|Mt) ou tau est le temps de
//défaut d'une entreprise et s varie sur subsub_t
static void densite_tau(PnlVect *res, PnlVect *subsub_t,
    double t, int Mt, Param *P, PnlMat *proba_M_t)
{
  int i,j;
  double sum;
  int m=P->m;
  int l_subsub_t=subsub_t->size;
  pnl_vect_resize(res,l_subsub_t);
  for(i=0;i<l_subsub_t;i++)
    {
      sum=0;
      for(j=Mt;j<=m-1;j++)
        sum += h(pnl_vect_get(subsub_t,i),j,P)*pnl_mat_get(
    proba_M_t,i,j-Mt)*(m-j);
      pnl_vect_set(res,i,sum/m);
    }
}



//renvoie un vecteur contenant P(tau<=z(k)|Mt) où k varie
    de nt+1 (nb de dates
//de paiement avant T) à N et z contient le tableau des
//dates de paiement (cette fonction marche)
static void proba_tau(PnlVect *res, double t, double T,int
    Mt,  Param *P, PnlVect *z, PnlMat *proba_M_t, PnlVect *su
    bsub_t)
{
  int i;
  PnlVect *densite_tau_res;
  int nt=n_t(t,z);//nombre de dates de paiement jusqu'au
  //temps t
  int lz=z->size;
  double z_nt_1=pnl_vect_get(z,nt+1);
  double z_nt_i;
  densite_tau_res=pnl_vect_create(0);
  pnl_vect_resize(res,lz-nt-1);
```

```
  densite_tau(densite_tau_res,subsub_t,t,Mt,P,proba_M_t);
  pnl_vect_set(res,0,(z_nt_1-t)*(pnl_vect_get(densite_tau_
    res,3)+pnl_vect_get(densite_tau_res,0))/2);

  for(i=2;i<lz-nt;i++)
    {
      z_nt_i=pnl_vect_get(z,nt+i);
      pnl_vect_set(res,i-1,pnl_vect_get(res,i-2)+(z_nt_i-pn
    l_vect_get(z,nt+i-1))*pnl_vect_get(densite_tau_res,subsub(
    i-1)+2));
    }
  pnl_vect_free(&densite_tau_res);
}


//renvoit l_x^[l,u]:=(x-ml)_+ - (x-mu)_+
static double v(double x, double l, double u, int m)
{
  double s1,s2,res;
  s1=(x<m*l)?0:x-m*l;
  s2=(x<m*u)?0:x-m*u;
  res=s1-s2;
  return res;
}

//calcule E[L_s^[a,b]|F_t], voir rapport de stage (3.25),s
//varie sur subsub_t
static void cond_loss(PnlVect *res, PnlVect *subsub_t,
    double t,int Mt,double l, double u, Param *P, PnlMat *proba_M_t)
{
  int i,k;
  double delta=1-P->R;
  double sum;
  int m=P->m;
  int l_subsub_t=subsub_t->size;
  pnl_vect_resize(res,l_subsub_t);
  for(i=0;i<l_subsub_t;i++)
    {
      sum=0;
      for(k=Mt;k<=m;k++)
        sum+=v(k*delta,l,u,m)*pnl_mat_get(proba_M_t,i,k-Mt)
```

```
    ;
      pnl_vect_set(res,i,sum);
    }
}

//calcule E[L_s^[a,b]|F_t], voir rapport de stage (3.25), i
//représente le ième élément de subsub_t
static double cond_loss_double(int i, double t,int Mt,
    double l, double u, Param *P, PnlMat *proba_M_t)
{
  int k;
  double delta=1-P->R;
  double sum;
  int m=P->m;
  sum=0;
  for(k=Mt;k<=m;k++)
    sum+=v(k*delta,l,u,m)*pnl_mat_get(proba_M_t,i,k-Mt);
  return sum;
}



//renvoie le reste de la division euclidienne de i par j
static int reste_div(int i, int j)
{
  div_t d;
  d = div (i, j);
  return d.rem;
}



//calcul de la jambe de défaut du CDS (formule (6) de l'ar
    ticle de 2007)
static double default_leg_CDS(double t,double T,int Mt,
    Param *P, PnlVect *z, PnlMat *proba_M_t, PnlVect *subsub_t)
{
  double sum;
  int l_subsub_t=subsub_t->size;
  int i;
  PnlVect *densite_tau_res;
```

```
  PnlVect *f_a_integrer;
  int r=P->r;
  densite_tau_res=pnl_vect_create(0);
  densite_tau(densite_tau_res,subsub_t,t,Mt,P,proba_M_t);
  f_a_integrer=pnl_vect_create(l_subsub_t);
  for(i=0;i<l_subsub_t;i++)
    pnl_vect_set(f_a_integrer,i,exp(-r*(pnl_vect_get(subsu
    b_t,i)-t))*pnl_vect_get(densite_tau_res,i));
  sum=integrale_simpson(t,T,subsub_t,f_a_integrer);
  pnl_vect_free(&f_a_integrer);
  pnl_vect_free(&densite_tau_res);
  return (1-P->R)*sum;
}


//calcul de la jambe de défaut (formule (3.24) du rapport
    de stage)
static double default_leg(double t,double T,int Mt, double
    l, double u, Param *P, PnlVect *z,  PnlMat *proba_M_t, PnlV
    ect *subsub_t)
{
  double sum=0;
  double res1;
  int l_subsub_t=subsub_t->size;
  int i;
  double sub_i, sub_i_1;
  int m=P->m;
  int r=P->r;
  int R=P->R;
  PnlVect *cond_loss_res;
  cond_loss_res=pnl_vect_create(0);
  cond_loss(cond_loss_res,subsub_t,t,Mt,l,u,P,proba_M_t);
  if(r>0.001)
    {sum=r*(pnl_vect_get(subsub_t,1)-t)*exp(-r*(pnl_vect_
    get(subsub_t,1)-t))*pnl_vect_get(cond_loss_res,1);
      for(i=0;2*i+3<l_subsub_t;i++)
        {
          sub_i=pnl_vect_get(subsub_t,2*i+1);
          sub_i_1=pnl_vect_get(subsub_t,2*i+3);
          sum+=r*exp(-r*(pnl_vect_get(subsub_t,2*i+2)-t))*
    pnl_vect_get(cond_loss_res,2*i+2)*(sub_i_1-sub_i);
```

```
        }
    }
  //res1 contient les 2 premiers termes de la jambe de
  //défaut (voir (3.24) du rapport de stage)
  res1=exp(-r*(T-t))*pnl_vect_get(cond_loss_res,l_subsub_t-
    1)-v(Mt*(1-R),l,u,m);
  pnl_vect_free(&cond_loss_res);
  return sum+res1;
}



//calcul de la jambe de paiement du CDS (formule (6) de
//l'article de 2007) (approximation grossière de l'intégra
    le)
static double premium_leg_CDS(double t,double T,int Mt,
    Param *P, PnlVect *z, PnlMat *proba_M_t, PnlVect *subsub_t)
{
  int nt,n,i,j;
  double sum;
  double delta_z=0.25;
  PnlVect *probatau;
  PnlVect *densite_tau_res;
  PnlVect *f_a_integrer;
  int l_subsub_t=subsub_t->size;
  int lz=z->size;
  int r=P->r;
  densite_tau_res=pnl_vect_create(0);
  f_a_integrer=pnl_vect_create(l_subsub_t);
  densite_tau(densite_tau_res,subsub_t,t,Mt,P,proba_M_t);
  nt=n_t(t,z);//nb de dates de paiement avant t
  probatau=pnl_vect_create(0);
  proba_tau(probatau,t,T,Mt,P,z,proba_M_t,subsub_t);
  //sum contient l'intégrale
  for(i=0;i<l_subsub_t;i++)
    {
      j=reste_div(i,4);
      pnl_vect_set(f_a_integrer,i,exp(-r*(pnl_vect_get(su
    bsub_t,i)-t))*pnl_vect_get(densite_tau_res,i)*(j*0.0625));
    }
  sum=integrale_simpson(t,T,subsub_t,f_a_integrer);
```

```
  //et la somme discrète
  for(n=nt+1;n<lz;n++)
    {
      sum+=delta_z*(exp(-r*(pnl_vect_get(z,n)-t))*(1-pnl_
    vect_get(probatau,n-nt-1)));
    }
  pnl_vect_free(&probatau);
  pnl_vect_free(&densite_tau_res);
  return sum;
}

//calcul de la jambe de paiement (on a approché la dernière
//intégrale par la valeur au point milieu)
static double premium_leg(double t, double T, int Mt,
    double l, double u, Param *P, PnlVect *z, PnlMat *proba_M_t, Pn
    lVect *subsub_t)
{
  double sum1,sum2,sum3,expo;
  int n,k,nt,i,j;
  double delta=1-P->R;
  PnlMat *loiT;
  int lz=z->size;
  double delta_z=0.25;
  PnlVect *f_a_integrer;
  int m=P->m;
  int r=P->r;
  loiT=pnl_mat_create(0,0);
  f_a_integrer=pnl_vect_create(subsub_t->size);
  loi_T(loiT,subsub_t,t,Mt,P,proba_M_t);
  nt=n_t(t,z);//nb de dates de paiement avant t
  sum1=0;
  sum2=0;
  sum3=0;
  if(l<=0.001) {sum1=0.05*m*(u-l);}
  for(n=nt+1;n<lz;n++)
    {
      expo=exp(-r*(pnl_vect_get(z,n)-t));
      sum1+=expo*delta_z*(m*(u-l)-cond_loss_double(subsub(
    n-nt),t,Mt,l,u,P,proba_M_t));
    }
```

```
  for(k=1;k<=m;k++)
    {
      sum2=0;
      for(n=nt+1;n<lz;n++)
        {
          sum2+=exp(-r*(pnl_vect_get(z,n)-t))*delta_z*pnl_
    mat_get(loiT,subsub(n-nt),k);
        }
      for(i=0;i<subsub_t->size;i++)
        {j=reste_div(i,4);
          pnl_vect_set(f_a_integrer,i,exp(-r*pnl_vect_get(
    subsub_t,i))*(1-r*j*0.0625)*pnl_mat_get(loiT,pnl_vect_get(
    subsub_t,i),k));
        }
      sum3+=(sum2-integrale_simpson(t,T,subsub_t,f_a_
    integrer))*(v(k*delta,l,u,m)-v((k-1)*delta,l,u,m));
    }
  pnl_mat_free(&loiT);
  pnl_vect_free(&f_a_integrer);
  return sum1+sum3;
}

static double V_CDS(double t,double T,int Mt, Param *P, Pn
    lVect *z, PnlMat *proba_M_t, PnlVect *subsub_t, double spr
    ead)
{
  double dl,pl;
  /* if (p==0) */
  /*      proba=pnl_mat_create_from_file("proba10"); */
  /*   else */
  /*      proba=pnl_mat_create_from_file("proba11"); */
  dl=default_leg_CDS(t,T,Mt,P,z,proba_M_t,subsub_t);
  pl=premium_leg_CDS(t,T,Mt,P,z,proba_M_t,subsub_t);
  return dl-spread*pl;
}


static double V_CDO(double t,double T,int Mt, double l,
    double u,double spread, Param *P, PnlVect *z, PnlMat *proba_M_t,
     PnlVect *subsub_t)
{
```

```
  double dl,pl;
  // prime de 500bp pour la tranche equity
  if((l==0)&&(u=0.03)) spread+=0.05;
  dl=default_leg(t,T,Mt,l,u,P,z,proba_M_t,subsub_t);
  pl=premium_leg(t,T,Mt,l,u,P,z,proba_M_t,subsub_t);
  return -dl+spread*pl;
}



//V_t^{CDS}-V_{t-}^{CDS} en un instant de défaut
static double DV_CDS(double t, double T,int Mt, Param *P,
    PnlVect *z, PnlMat *proba_M_t, PnlMat *proba_M_t_1, PnlVec
    t *subsub_t, double spread)
{
  double v1,v2;
  v1=V_CDS(t,T,Mt-1,P,z,proba_M_t_1,subsub_t,spread);
  v2=V_CDS(t,T,Mt,P,z,proba_M_t,subsub_t,spread);
  return v2-v1;
}

static double DV_CDO(double t, double T,int Mt,double l,
    double u, Param *P, PnlVect *z, PnlMat *proba_M_t, PnlMat *prob
    a_M_t_1, PnlMat *proba_M_0, PnlVect *subsub_t, PnlVect *su
    bsub_0)
{
  double v1,v2,spread;
  spread=default_leg(0,T,0,l,u,P,z,proba_M_0,subsub_0)/prem
    ium_leg(0,T,0,l,u,P,z,proba_M_0,subsub_0);
  v1=V_CDO(t,T,Mt-1,l,u,spread,P,z,proba_M_t_1,subsub_t);
  v2=V_CDO(t,T,Mt,l,u,spread,P,z,proba_M_t,subsub_t);
  return v2-v1;
}

//formule (11) page 7, où t est un instant de défaut
static double DG_CDO(double t, double T,int Mt,double l,
    double u, Param *P , PnlVect *z, PnlMat *proba_M_t, PnlMat *prob
    a_M_t_1, PnlMat *proba_M_0, PnlVect *subsub_t, PnlVect *su
    bsub_0)
{
  double R=P->R;
```

```
  int m=P->m;
  return DV_CDO(t,T,Mt,l,u,P,z,proba_M_t,proba_M_t_1,proba_
    M_0,subsub_t,subsub_0)-(v(Mt*(1-R),l,u,m)-v((Mt-1)*(1-R),l,
    u,m));
}

//formule (13) page 8)
static double delta(double t, double T,int Mt,double l,
    double u, Param *P, PnlVect *z, PnlMat *proba_M_t, PnlMat *prob
    a_M_t_1, PnlMat *proba_M_0, PnlVect *subsub_t, PnlVect *su
    bsub_0)
{
  int nt;
  double spread_CDS;
  double DG_IND;
  int m=P->m;
  double R=P->R;
  nt=n_t(t,z);//nb de dates de paiement avant t
  //spread_CDS=0.0026;
  spread_CDS=default_leg_CDS(0,T,0,P,z,proba_M_0,subsub_0)/
    premium_leg_CDS(0,T,0,P,z,proba_M_0,subsub_0);
  DG_IND=(m-Mt-1)*DV_CDS(t,T,Mt+1,P,z,proba_M_t,proba_M_t_1
    ,subsub_t,spread_CDS)-V_CDS(t,T,Mt,P,z,proba_M_t_1,subsub_
    t,spread_CDS)+(1-R)-spread_CDS*(t-pnl_vect_get(z,nt));
  return -DG_CDO(t,T,Mt+1,l,u,P,z,proba_M_t,proba_M_t_1,
    proba_M_0,subsub_t,subsub_0)/DG_IND;

}


int CALC(Hedging_FreyBackhaus)(void *Opt, void *Mod, Prici
    ngMethod *Met)
{
  TYPEOPT *ptOpt;
  TYPEMOD *ptMod;
  int      n_tranch, i, k;
  int      n;
  double   R, T, r, t, n_defaults;
  PnlVect *tranch;
  Param   *P;
  PnlVect *z;
```

```
PnlMat  *proba_M_t;
PnlMat  *proba_M_t_1;
PnlMat  *proba_M_0;
PnlVect *sub_t;
PnlVect *sub_0;
PnlVect *res;
PnlVect *subsub_t;
PnlVect *subsub_0;
int      lz, Mt;

ptOpt = (TYPEOPT*)Opt;
ptMod = (TYPEMOD*)Mod;

tranch =  ptOpt->tranch.Val.V_PNLVECT;
n_tranch  = tranch->size-1;
n = ptMod->Ncomp.Val.V_PINT;
r = ptMod->r.Val.V_DOUBLE;
T = ptOpt->maturity.Val.V_DATE;
t = ptOpt->date.Val.V_DATE;
n_defaults = ptOpt->n_defaults.Val.V_INT;
R = ptOpt->p_recovery.Val.V_DOUBLE;

lz=4*T+1;
Mt=n_defaults;
P=malloc(sizeof(Param));
P->m=n;
P->R=R;
P->r=r;
P->psi=0.005;
P->lambda0=0.85910;
P->lambda1=0.18803;
P->lambda2=22.125;
P->sp=0.0026;

z=pnl_vect_create(lz);
for(k=0;k<lz;k++)
  {pnl_vect_set(z,k,k*1./4);}
sub_t=pnl_vect_create(0);
sub_0=pnl_vect_create(0);
res=pnl_vect_create(0);
subsub_t=pnl_vect_create(0);
```

```
  subsub_0=pnl_vect_create(0);
  //on subdivise z en 2
  create_sub(sub_t,t,T,z);
  create_sub(sub_0,0,T,z);
  //on subdivise sub en 2
  create_sub(subsub_t,t,T,sub_t);
  create_sub(subsub_0,0,T,sub_0);
  proba_M_t=pnl_mat_create(0,0);
  proba_M_t_1=pnl_mat_create(0,0);
  proba_M_0=pnl_mat_create(0,0);

  build_proba_M(proba_M_t,subsub_t,t,Mt+1,P);


  build_proba_M(proba_M_t_1,subsub_t,t,Mt,P);
  build_proba_M(proba_M_0,subsub_0,0,0,P);

  for ( i=0 ; i<n_tranch ; i++ )
    {
      double l = GET (tranch, i);
      double u = GET (tranch, i+1);
      LET(Met->Res[0].Val.V_PNLVECT, i) = 10000.*delta(t,T,
    Mt,l,u,P,z,proba_M_t,proba_M_t_1,proba_M_0,subsub_t,subsub_
    0);
    }

  pnl_vect_free(&sub_t);
  pnl_vect_free(&sub_0);
  pnl_vect_free(&res);
  pnl_vect_free(&subsub_t);
  pnl_vect_free(&subsub_0);
  pnl_vect_free(&z);
  pnl_mat_free(&proba_M_t);
  pnl_mat_free(&proba_M_t_1);
  pnl_mat_free(&proba_M_0);
  free(P);

  return OK;
}
```

```
static int CHK_OPT(Hedging_FreyBackhaus)(void *Opt, void *
    Mod)
{
  Option*  ptOpt   = (Option*)Opt;
  if (strcmp (ptOpt->Name, "CDO_HEDGING") != 0) return WRON
    G;
  return OK;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  TYPEOPT *ptOpt;
  int      n_tranch;
  ptOpt = (TYPEOPT*)Opt->TypeOpt;
  if ( Met->init == 0)
    {
      Met->init=1;
       Met->HelpFilenameHint = "FB_cdo_hedging";
      Met->Par[0].Val.V_DOUBLE = 0.3;
      n_tranch = ptOpt->tranch.Val.V_PNLVECT->size-1;

      Met->Res[0].Val.V_PNLVECT = pnl_vect_create_from_
    double (n_tranch, 0.);
    }
  return OK;
}

PricingMethod MET(Hedging_FreyBackhaus) =
{
  "Hedging_FreyBackhaus",
  {
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(Hedging_FreyBackhaus),
  {{"Delta(bp)",PNLVECT,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(Hedging_FreyBackhaus),
  CHK_ok,
  MET(Init)
};
```

References