```
   Help
#include <stdlib.h>
#include  "bscir2d_stda.h"
#include "error_msg.h"
#include "pnl/pnl_matrix.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(TR_cgmz)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_cgmz)(void*Opt,void *Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static char *infilename;
static double **V,**S;
static double **y,**f;
static int **f_down,**f_up;
static int **y_down,**y_up;
static double **pu_y,**pd_y;
static double **pu_f,**pd_f;
static double *n_min,*n_max,*GM;
static double ***RF,***EQL;
static double *vm,*vpm,*vm1,*vpm1,*vm2,*vpm2,*coeff;
static int Ns;

//Singular points
static int **nb_critical,**nb_critical1;
static int MAX_SING_POINTS=2000;
static double *lm_insurance;

static void SortVect(unsigned long n, double *arr)
{
  PnlVect a;
  PnlVectInt *i;
```

```
  a = pnl_vect_wrap_array (arr, n);
  i = pnl_vect_int_create(0);
  pnl_vect_qsort_index(&a, i, 'i');
  pnl_vect_int_free (&i);
}

static double linear_interpolation(double val,int j,int k)
{
  double res;
  int l;

  if(val<RF[j][k][0])
    return EQL[j][k][0];
  else
    if(val>RF[j][k][nb_critical[j][k]])
      return EQL[j][k][nb_critical[j][k]];
    else
      if(fabs(val-RF[j][k][nb_critical[j][k]])<1.e-8)
        return EQL[j][k][nb_critical[j][k]];
      else
        {
          l=0;
          while ((RF[j][k][l]<val)&&(l<=nb_critical[j][k]))
     l++;
          if(l==0)
            res=EQL[j][k][0];
          else
            res=((val-RF[j][k][l-1])*EQL[j][k][l]+(RF[j][k]
    [l]-val)*EQL[j][k][l-1])/(RF[j][k][l]-RF[j][k][l-1]);

          return res;
        }
}

/*Memory Allocation*/
static int memory_allocation(int Nst)
{
  int i,j;

  n_min=(double *)malloc((Nst+1)*sizeof(double));
  n_max=(double *)malloc((Nst+1)*sizeof(double));
```

```
GM=(double *)malloc((Nst+1)*sizeof(double));
vm=(double *)malloc(MAX_SING_POINTS*sizeof(double));
vpm=(double *)malloc(MAX_SING_POINTS*sizeof(double));
vm1=(double *)malloc(MAX_SING_POINTS*sizeof(double));
vpm1=(double *)malloc(MAX_SING_POINTS*sizeof(double));
vm2=(double *)malloc(MAX_SING_POINTS*sizeof(double));
vpm2=(double *)malloc(MAX_SING_POINTS*sizeof(double));
coeff=(double *)malloc(MAX_SING_POINTS*sizeof(double));

 V=(double**)calloc(Nst+1,sizeof(double*));
if (V==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    V[i]=(double *)calloc(Nst+1,sizeof(double));
    if (V[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

S=(double**)calloc(Nst+1,sizeof(double*));
if (S==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    S[i]=(double *)calloc(Nst+1,sizeof(double));
    if (S[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

pu_y=(double**)calloc(Nst+1,sizeof(double*));
if (pu_y==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    pu_y[i]=(double *)calloc(Nst+1,sizeof(double));
    if (pu_y[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

pd_y=(double**)calloc(Nst+1,sizeof(double*));
if (pd_y==NULL)
```

```
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    pd_y[i]=(double *)calloc(Nst+1,sizeof(double));
    if (pd_y[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

pu_f=(double**)calloc(Nst+1,sizeof(double*));
if (pu_f==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    pu_f[i]=(double *)calloc(Nst+1,sizeof(double));
    if (pu_f[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

pd_f=(double**)calloc(Nst+1,sizeof(double*));
if (pd_f==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    pd_f[i]=(double *)calloc(Nst+1,sizeof(double));
    if (pd_f[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

y=(double**)calloc(Nst+1,sizeof(double*));
if (y==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    y[i]=(double *)calloc(Nst+1,sizeof(double));
    if (y[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

f=(double**)calloc(Nst+1,sizeof(double*));
if (f==NULL)
  return MEMORY_ALLOCATION_FAILURE;
```

```
for (i=0;i<Nst+1;i++)
  {
    f[i]=(double *)calloc(Nst+1,sizeof(double));
    if (f[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

f_down=(int**)calloc(Nst+1,sizeof(int*));
if (f_down==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    f_down[i]=(int *)calloc(Nst+1,sizeof(int));
    if (f_down[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

f_up=(int**)calloc(Nst+1,sizeof(int*));
if (f_up==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    f_up[i]=(int *)calloc(Nst+1,sizeof(int));
    if (f_up[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

y_down=(int**)calloc(Nst+1,sizeof(int*));
if (y_down==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    y_down[i]=(int *)calloc(Nst+1,sizeof(int));
    if (y_down[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

y_up=(int**)calloc(Nst+1,sizeof(int*));
if (y_up==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
```

```
  {
    y_up[i]=(int *)calloc(Nst+1,sizeof(int));
    if (y_up[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

nb_critical=(int**)calloc(Nst+1,sizeof(int*));
if (nb_critical==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    nb_critical[i]=(int *)calloc(2*Nst+1,sizeof(int));
    if (nb_critical[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

nb_critical1=(int**)calloc(Nst+1,sizeof(int*));
if (nb_critical1==NULL)
  return MEMORY_ALLOCATION_FAILURE;
for (i=0;i<Nst+1;i++)
  {
    nb_critical1[i]=(int *)calloc(2*Nst+1,sizeof(int));
    if (nb_critical1[i]==NULL)
      return MEMORY_ALLOCATION_FAILURE;
  }

RF= (double ***)malloc((Nst+1)*sizeof(double**));
for(i=0;i<=Nst;i++)
  RF[i]=(double **)malloc((Nst+1)*sizeof(double*));
for(i=0;i<=Nst;i++)
  for(j=0;j<=Nst;j++)
    RF[i][j]=(double *)malloc(MAX_SING_POINTS*sizeof(
  double));

EQL= (double ***)malloc((Nst+1)*sizeof(double**));
for(i=0;i<=Nst;i++)
  EQL[i]=(double **)malloc((Nst+1)*sizeof(double*));
for(i=0;i<=Nst;i++)
  for(j=0;j<=Nst;j++)
    EQL[i][j]=(double *)malloc(MAX_SING_POINTS*sizeof(
  double));
```

```
  return 1;
}

static void free_memory(int Nst)
{
  int i,j;

  free(n_min);
  free(n_max);
  free(GM);
  free(vm);
  free(vpm);
  free(vm1);
  free(vpm1);
  free(vm2);
  free(vpm2);
  free(coeff);

  for (i=0;i<Nst+1;i++)
    free(S[i]);
  free(S);

  for (i=0;i<Nst+1;i++)
    free(V[i]);
  free(V);

  for (i=0;i<Nst+1;i++)
    free(pu_y[i]);
  free(pu_y);

  for (i=0;i<Nst+1;i++)
    free(pd_y[i]);
  free(pd_y);

  for (i=0;i<Nst+1;i++)
    free(y[i]);
  free(y);

  for (i=0;i<Nst+1;i++)
    free(y_up[i]);
```

```
free(y_up);

for (i=0;i<Nst+1;i++)
  free(y_down[i]);
free(y_down);

for (i=0;i<Nst+1;i++)
  free(pu_f[i]);
free(pu_f);

for (i=0;i<Nst+1;i++)
  free(pd_f[i]);
free(pd_f);

for (i=0;i<Nst+1;i++)
  free(f[i]);
free(f);

for (i=0;i<Nst+1;i++)
  free(f_up[i]);
free(f_up);

for (i=0;i<Nst+1;i++)
  free(f_down[i]);
free(f_down);

for (i=0;i<Nst+1;i++)
  free(nb_critical[i]);
free(nb_critical);

for (i=0;i<Nst+1;i++)
  free(nb_critical1[i]);
free(nb_critical1);

for (i=0;i<Nst+1;i++)
  for (j=0;j<Nst+1;j++)
    free(RF[i][j]);
for (j=0;j<Nst+1;j++)
  free(RF[j]);
free(RF);
```

```
  for (i=0;i<Nst+1;i++)
    for (j=0;j<Nst+1;j++)
      free(EQL[i][j]);
  for (j=0;j<Nst+1;j++)
    free(EQL[j]);
  free(EQL);
}

static double compute_f(double r,double omega)
{
  return 2.*sqrt(r)/omega;
}

static double compute_r(double R,double omega)
{
  double val;

  if(R>0.)
    val=SQR(R)*SQR(omega)/4.;
  else
    val=0.;
  return val;
}

static double compute_y(double s,double r,double sigma,
    double omega,double rho)
{
  double Y;

  Y=(log(s)/sigma-rho*2.*sqrt(r)/omega)*1./sqrt(1.-SQR(rho)
    );

  return Y;
}

static double compute_S(double Y,double R,double sigma,
    double rho)
{
  double val;

  val=exp(sigma*(sqrt(1.-SQR(rho))*Y+rho*R));
```

```
  return val;
}

/*Calibration of the tree*/
static int calibration(double r0,double s0,double tt,
    double kappa,double omega,double theta,double rho,double sigma,
    double delta)
{
  int i,j;
  double mu_f;
  int z,j_step;
  double Ru,Rd;
  double val1,val2;
  int j1,j2;
  double sqrt_dt,dt;
  int cont1,cont2;
  int count_jump_r;

  dt=tt/(double)Ns;
  sqrt_dt=sqrt(dt);

  /*Fixed tree for R=f*/
  f[0][0]=compute_f(r0,omega);

  j1=-(int)(f[0][0]/sqrt_dt);
  val1=f[0][0]+j1*sqrt_dt;

  j2=-(int)(f[0][0]/sqrt_dt+1);
  val2=f[0][0]+j2*sqrt_dt;

  if((fabs(val1)<0.00001)||(fabs(val2)<0.00001))
    {
      Ns=Ns+2;
      dt=tt/(double)Ns;
      sqrt_dt=sqrt(dt);
      printf("Change of step number N=%d{n",Ns);
    }

  V[0][0]=compute_r(f[0][0],omega);
  f[1][0]=f[0][0]-sqrt_dt;
```

```
f[1][1]=f[0][0]+sqrt_dt;
V[1][0]=compute_r(f[1][0],omega);
V[1][1]=compute_r(f[1][1],omega);
for(i=1;i<Ns;i++)
  for(j=0;j<=i;j++)
    {
      f[i+1][j]=f[i][j]-sqrt_dt;
      f[i+1][j+1]=f[i][j]+sqrt_dt;
      V[i+1][j]=compute_r(f[i+1][j],omega);
      V[i+1][j+1]=compute_r(f[i+1][j+1],omega);
    }

cont1=0;
cont2=0;
count_jump_r=0;

/*Evolve tree for f*/
for(i=0;i<Ns;i++)
  {
    for(j=0;j<=i;j++)
      {
        /*node F=R*/
        /*Compute mu_f*/
        mu_f=(kappa*(4.*theta-SQR(f[i][j])*SQR(omega))-SQ
  R(omega))/(2.*f[i][j]*SQR(omega));
        z=(int)floor(mu_f*sqrt_dt);/*Compute f+ e f-*/

        if(z%2==0)
          j_step=z;
        else
          j_step=z+1;

        Rd=f[i][j]+(j_step-1)*sqrt_dt;
        Ru=f[i][j]+(j_step+1)*sqrt_dt;

        if(j_step!=0)
          {
            count_jump_r++;
          }

        pu_f[i][j]=(mu_f*dt+(f[i][j]-Rd))/(Ru-Rd);
```

```
          f_down[i][j]=(int)(j_step/2);
          f_up[i][j]=(int)(j_step/2)+1;

          if(Ru-1.e-9>f[i+1][i+1])
            {
              cont1++;
              pu_f[i][j]=1;
              f_up[i][j]=i+1-j;
              f_down[i][j]=i-j;
            }

          if (Rd+1.e-9<f[i+1][0])
            {
              pu_f[i][j]=0.;
              cont2++;
              f_up[i][j]=-j+1;
              f_down[i][j]=-j;
            }
          pd_f[i][j]=1.-pu_f[i][j];
        }
    }


/*Initalise first node Y */
y[0][0]=compute_y(s0,r0,sigma,omega,rho);
S[0][0]=compute_S(y[0][0],f[0][0],sigma,rho);

y[1][0]=y[0][0]-sqrt_dt;
y[1][1]=y[0][0]+sqrt_dt;
S[1][0]=compute_S(y[1][0],f[1][0],sigma,rho);
S[1][1]=compute_S(y[1][1],f[1][1],sigma,rho);

for(i=1;i<Ns;i++)
  for(j=0;j<=i;j++)
    {
      y[i+1][j]=y[i][j]-sqrt_dt;
      y[i+1][j+1]=y[i][j]+sqrt_dt;
      S[i+1][j]=compute_S(y[i+1][j],f[i+1][j],sigma,rho);
      S[i+1][j+1]=compute_S(y[i+1][j+1],f[i+1][j+1],sigma
  ,rho);
```

```
      }
  return 1;
}

////////////////////////////////////////////////////////
    ///////////////////////
/*Compute Equity-Linked Policy Price*/
static double compute_price(double premium,double r0,
    double s0,double tt,double kappa,double omega,double theta,
    double rho,double sigma,double delta, double D,int n_obs,int am,
    int life,int x_age,int n_passi,double h_up)
{
  int i,j,k,l,n_k,kk,jj;
  double puu,pud,pdu,pdd,stock;
  int fv_up,fv_down,yv_up,yv_down;
  double pu_ys,pd_ys;
  double mu_f,mu_y,mu_x;
  int z,j_step;
  double Ru,Rd;
  double val_rf;
  int flag_monit;
  double stockuu,stockud,stockdu,stockdd;
  double val_min,val_max;
  double prezzo1,prezzo2,prezzo3,prezzo4;
  double rf1,rf2,rf3,rf4;
  double tol1,tol2;
  int max_critical=-100.;
  int max_critical1=-100.;
  int old_nb_critical,index;
  double x1,x2,y1,y2,b,a;
  double val;
  double pv,qm;
  int i1;
  double error,errp,m1;
  double sqrt_dt,dt;
  int cont1y,cont2y;
  int count_jump_y;
  double D_m,price;

  dt=tt/(double)Ns;
  sqrt_dt=sqrt(dt);
```

```
tol1=0.0000001;
tol2=0.00000001;

// warnings
val_min=0.;
val_max=0.;
n_k = 0;
//Maximum and minimum number of equities
n_min[0]=D/s0;
n_max[0]=D/s0;

for(i=1;i<=Ns;i++)
  {
    if((i%n_passi)==0)
      {
        if(rho>=0)
          {
            n_min[i]=n_min[i-1]+D/compute_S(y[i][i],f[i][
  i],sigma,rho);
            n_max[i]=n_max[i-1]+D/compute_S(y[i][0],f[i][
  0],sigma,rho);
          }
        else
          {
            n_min[i]=n_min[i-1]+D/compute_S(y[i][i],f[i][
  0],sigma,rho);
            n_max[i]=n_max[i-1]+D/compute_S(y[i][0],f[i][
  i],sigma,rho);
          }
      }
    else
      {
        n_min[i]=n_min[i-1];
        n_max[i]=n_max[i-1];
      }
  }

GM[0]=0;
val=D;
for(i=1;i<=Ns;i++)
  {
```

```
    if((i%n_passi)==0)
      {
        if(delta>0)
          GM[i]=D*exp(delta)*(exp(delta*(double)(i)*dt)-1
  )/(exp(delta)-1.);
        else
          GM[i]=(double)i*dt*D;
        val=GM[i]+D;
      }
    else
      {
        if(((i-1)%n_passi)==0)
            GM[i]=val*exp(delta*dt);
        else
            GM[i]=GM[i-1]*exp(delta*dt);
      }
  }

/*Singular points at Maturity for Equity-linked policy*/
for(j=0;j<=Ns;j++)
  for(k=0;k<=Ns;k++)
    {
      stock=compute_S(y[Ns][j],f[Ns][k],sigma,rho);

      if((stock*n_min[Ns-1]<GM[Ns])&&(stock*n_max[Ns-1]>
  GM[Ns]))
        {

          RF[j][k][0]=stock*n_min[Ns-1];
          RF[j][k][1]=GM[Ns];
          RF[j][k][2]=stock*n_max[Ns-1];

          EQL[j][k][0]=GM[Ns];
          EQL[j][k][1]=GM[Ns];
          EQL[j][k][2]=RF[j][k][2];

          nb_critical[j][k]=2;

        }
      else
        if((stock*n_min[Ns-1]>GM[Ns]))
```

```
            {

              RF[j][k][0]=stock*n_min[Ns-1];
              RF[j][k][1]=stock*n_max[Ns-1];
              EQL[j][k][0]=RF[j][k][0];
              EQL[j][k][1]=RF[j][k][1];
              nb_critical[j][k]=1;

            }
          else
            if((stock*n_max[Ns-1]<GM[Ns]))
              {
                RF[j][k][0]=stock*n_min[Ns-1];
                RF[j][k][1]=stock*n_max[Ns-1];
                EQL[j][k][0]=GM[Ns];
                EQL[j][k][1]=GM[Ns];
                nb_critical[j][k]=1;
              }
      }

  /*Dynamic Programming*/
  for(i=Ns-1;i>=0;i--)
    {
      if(life==1)
        qm=(lm_insurance[x_age+(int)(i*dt)]-lm_insurance[x_
    age+(int)(i*dt)+1])/lm_insurance[x_age+(int)(i*dt)]*(1./((
    double)n_passi));
      else qm=0;

      pv=1.-qm;



      if((((i)%n_passi)==0)&&(i!=Ns)&&(i>0))
        flag_monit=1;
      else flag_monit=0;

      for(j=0;j<=i;j++)
        for(k=0;k<=i;k++)
          {
            /*node Y*/
```

```
        mu_x=(V[i][k]-0.5*SQR(sigma))/sigma;
        mu_f=(kappa*(4.*theta-SQR(f[i][k])*SQR(omega))-
SQR(omega))/(2.*f[i][k]*SQR(omega));
        mu_y=(mu_x-rho*mu_f)/sqrt(1.-SQR(rho));

        z=(int)floor(mu_y*sqrt_dt);/*Compute y+ e y-*/
        if(z%2==0)
          j_step=z;
        else
          j_step=z+1;

        if(j_step!=0)
          {
            count_jump_y++;
          }

        Rd=y[i][j]+(j_step-1)*sqrt_dt;
        Ru=y[i][j]+(j_step+1)*sqrt_dt;

        pu_ys=(mu_y*dt+(y[i][j]-Rd))/(Ru-Rd);

        yv_down=(int)(j_step/2);
        yv_up=(int)(j_step/2)+1;


        if(Ru-1.e-9>y[i+1][i+1])
          {
            pu_ys=1.;
            cont1y++;
            yv_up=i+1-j;
            yv_down=i-j;
          }

        if(Rd+1.e-9<y[i+1][0])
          {
            pu_ys=0.;
            cont2y++;
            yv_up=-j+1;
            yv_down=-j;
          }
```

```
          pd_ys=1.-pu_ys;

          puu=pu_ys*pu_f[i][k];
          pud=pu_ys*pd_f[i][k];
          pdu=pd_ys*pu_f[i][k];
          pdd=pd_ys*pd_f[i][k];

          fv_up=f_up[i][k];
          fv_down=f_down[i][k];

          //Current stock
          stock=compute_S(y[i][j],f[i][k],sigma,rho);

          //Next stocks
          stockuu=compute_S(y[i+1][j+yv_up],f[i+1][k+fv_
up],sigma,rho);
          stockud=compute_S(y[i+1][j+yv_up],f[i+1][k+fv_
down],sigma,rho);
          stockdu=compute_S(y[i+1][j+yv_down],f[i+1][k+fv
_up],sigma,rho);
          stockdd=compute_S(y[i+1][j+yv_down],f[i+1][k+fv
_down],sigma,rho);

          //Val Min et Val max
          if(i==0)
            {
              val_min=D;
              val_max=D;
            }
          else
            if((flag_monit==1)&&(i>0))
              {
                val_min=stock*n_min[i-1];
                val_max=stock*n_max[i-1];
              }
            else
              if((flag_monit==0)&&(i>=0))
                {
                  val_min=stock*n_min[i-1];
                  val_max=stock*n_max[i-1];
                }
```

```
          //RF min
          vm[0]=val_min;

          //interior RF
          if(i>0)
            {
              n_k=1;

              //uu
              for(l=0;l<=nb_critical[j+yv_up][k+fv_up];l+
+)
                {
                  if(flag_monit==0)
                    val_rf=RF[j+yv_up][k+fv_up][l]*stock/
stockuu;
                  else
                    val_rf=(RF[j+yv_up][k+fv_up][l])*sto
ck/stockuu-D;

                  if((val_rf<val_max)&&(val_rf>val_min))
                    {
                      vm[n_k]=val_rf;
                      n_k++;
                    }
                }
              //ud
              for(l=0;l<=nb_critical[j+yv_up][k+fv_down];
l++)
                {

                  if(flag_monit==0)
                    val_rf=RF[j+yv_up][k+fv_down][l]*sto
ck/stockud;
                  else
                    val_rf=(RF[j+yv_up][k+fv_down][l])*
stock/stockud-D;

                  if((val_rf<val_max)&&(val_rf>val_min))
                    {
                      vm[n_k]=val_rf;
```

```
                    n_k++;
                  }
              }

            //du
            for(l=0;l<=nb_critical[j+yv_down][k+fv_up];
l++)
              {

                if(flag_monit==0)
                  val_rf=RF[j+yv_down][k+fv_up][l]*sto
ck/stockdu;
                else
                  val_rf=(RF[j+yv_down][k+fv_up][l])*
stock/stockdu-D;

                if((val_rf<val_max)&&(val_rf>val_min))
                  {
                    vm[n_k]=val_rf;
                    n_k++;
                  }
              }
            //dd
            for(l=0;l<=nb_critical[j+yv_down][k+fv_dow
n];l++)
              {
                if(flag_monit==0)
                  val_rf=RF[j+yv_down][k+fv_down][l]*
stock/stockdd;
                else
                  val_rf=(RF[j+yv_down][k+fv_down][l])*
stock/stockdd-D;

                if((val_rf<val_max)&&(val_rf>val_min))
                  {
                    vm[n_k]=val_rf;
                    n_k++;
                  }
              }
          }
        //RF max
```

```
          vm[n_k]=val_max;
          nb_critical1[j][k]=n_k;

          /*Sorting*/

          SortVect(nb_critical1[j][k],vm);

          /*Eliminate close singular points*/
          vm1[0]=vm[0];

          kk=0;
          l=0;
          do {
            do {
              l++;
            }while((vm[l]<=vm1[kk]+tol1)&&(l<nb_critical1
[j][k]));
            kk++;
            vm1[kk]=vm[l];

          }while((l<nb_critical1[j][k]));

          nb_critical1[j][k]=kk;

          if(fabs(vm1[nb_critical1[j][k]]-vm1[nb_criti
cal1[j][k]-1])<tol2)
            nb_critical1[j][k]--;

          //Compute Singular ordinate
          for(n_k=0;n_k<=nb_critical1[j][k];n_k++)
            {
              val_rf=vm1[n_k];
              if(flag_monit==0)
                {
                  rf1=val_rf*stockuu/stock;
                  prezzo1=linear_interpolation(rf1,j+yv_
up,k+fv_up);

                  rf2=val_rf*stockud/stock;
                  prezzo2=linear_interpolation(rf2,j+yv_
up,k+fv_down);
```

```
                rf3=val_rf*stockdu/stock;
                prezzo3=linear_interpolation(rf3,j+yv_
down,k+fv_up);
                rf4=val_rf*stockdd/stock;
                prezzo4=linear_interpolation(rf4,j+yv_
down,k+fv_down);
              }
            else
              {
                rf1=(val_rf+D)*stockuu/stock;
                prezzo1=linear_interpolation(rf1,j+yv_
up,k+fv_up);
                rf2=(val_rf+D)*stockud/stock;
                prezzo2=linear_interpolation(rf2,j+yv_
up,k+fv_down);
                rf3=(val_rf+D)*stockdu/stock;
                prezzo3=linear_interpolation(rf3,j+yv_
down,k+fv_up);
                rf4=(val_rf+D)*stockdd/stock;
                prezzo4=linear_interpolation(rf4,j+yv_
down,k+fv_down);
              }

            vpm1[n_k]=exp(-V[i][k]*dt)*
            (puu*prezzo1+pud*prezzo2+pdu*prezzo3+pdd*
prezzo4);

            vpm1[n_k]*=pv;

            prezzo1=MAX(rf1,GM[i+1]);
            prezzo2=MAX(rf2,GM[i+1]);
            prezzo3=MAX(rf3,GM[i+1]);
            prezzo4=MAX(rf4,GM[i+1]);
            vpm1[n_k]+=exp(-V[i][k]*dt)*qm*(puu*prezzo1
+pud*prezzo2+pdu*prezzo3+pdd*prezzo4);
          }

        //Subtract the premium
        if((flag_monit==1)||(i==0))
          {
            for(l=0;l<=nb_critical1[j][k];l++)
```

```
                {
                  vpm1[l]=vpm1[l]-premium;
                }
            }

          max_critical1=MAX(nb_critical1[j][k],max_criti
cal1);

          //Upper Bound
          i1=0;
          index=0;
          vm2[0]=vm1[0];
          vpm2[0]=vpm1[0];
          while(i1<nb_critical1[j][k]-1)
            {
              l=1;
              do
                {
                  l++;
                  m1=(vpm1[i1+l]-vpm1[i1])/(vm1[i1+l]-vm1
[i1]);
                  error=0.;
                  for(jj=1;jj<=l-1;jj++)
                    {
                      errp=m1*(vm1[i1+jj]-vm1[i1])+vpm1[
i1]-vpm1[i1+jj];
                      if (errp<0) errp=-errp;
                      if (errp>error) error=errp;
                    }
                }
              while(!((error>h_up)||((i1+l)==nb_critical1
[j][k])));
              index++;
              vm2[index]=vm1[i1+l-1];
              vpm2[index]=vpm1[i1+l-1];
              i1=i1+l-1;
            }
          while(i1<nb_critical1[j][k])
            {
              index++;
              vm2[index]=vm1[i1+1];
```

```
            vpm2[index]=vpm1[i1+1];
            i1 =i1+1;
          }
        nb_critical1[j][k]=index;

        //Copy
        for(l=0;l<=nb_critical1[j][k];l++)
          {
            vm1[l]=vm2[l];
            vpm1[l]=vpm2[l];
          }

        //Early exercise
        D_m=D;
        D=0.;
        if(am==1)
          if((flag_monit==1)&&(i>0))
            {
              old_nb_critical=nb_critical1[j][k];

              //American with reference fund
              if((vm1[old_nb_critical]-D)>=vpm1[old_nb_
critical])
                {
                  if((vm1[0]-D)>=vpm1[0])
                    {
                      vm2[0]=vm1[0];
                      vm2[1]=vm1[old_nb_critical];
                      vpm2[0]=vm2[0]-D;
                      vpm2[1]=vm2[1]-D;
                      nb_critical1[j][k]=1;
                    }
                  else
                    if(vpm1[0]>(vm1[0]-D))
                      {
                        l=1;
                        while((vpm1[l]>(vm1[l]-D))&&(l<
=nb_critical1[j][k]))l++;

                        if(l<nb_critical1[j][k])
                          {
```

```
                                x1=vm1[l-1];
                                x2=vm1[l];
                                y1=vpm1[l-1];
                                y2=vpm1[l];
                                a=(y2-y1)/(x2-x1);
                                b=(y1*x2-x1*y2)/(x2-x1);

                                vm2[l]=(D+b)/(1.-a);
                                vpm2[l]=vm2[l]-D;
                                vm2[l+1]=vm1[old_nb_criti
    cal];
                                vpm2[l+1]=vm2[l+1]-D;
                                nb_critical1[j][k]=l+1;
                              }
                          }
                        //Copy
                        for(l=0;l<=nb_critical1[j][k];l++)
                          {
                            vm1[l]=vm2[l];
                            vpm1[l]=vpm2[l];
                          }

                      }

                    old_nb_critical=nb_critical1[j][k];

                    //American with guaranted minimum
                    if(GM[i]>=vpm1[old_nb_critical])
                      {
                        vm2[0]=vm1[0];
                        vm2[1]=vm1[old_nb_critical];
                        vpm2[0]=GM[i];
                        vpm2[1]=GM[i];
                        nb_critical1[j][k]=1;
                      }
                    else
                      if(GM[i]>vpm1[0])
                        {
                          l=1;
                          while(((vpm1[l])<=GM[i])&&
                                (l<=nb_critical1[j][k]))
```

```
                        l++;

                      vm2[0]=vm1[0];
                      vpm2[0]=GM[i];

                      x1=vm1[l-1];
                      x2=vm1[l];
                      y1=vpm1[l-1];
                      y2=vpm1[l];
                      a=(y2-y1)/(x2-x1);
                      b=(y1*x2-x1*y2)/(x2-x1);

                      vm2[1]=(GM[i]-b)/a;
                      vpm2[1]=GM[i];

                      index=2;

                      while(l<=nb_critical1[j][k])
                        {
                          vm2[index]=vm1[l];
                          vpm2[index]=vpm1[l];
                          index++;
                          l++;
                        }
                      nb_critical1[j][k]=index-1;
                    }
                  else
                    for(l=0;l<=nb_critical1[j][k];l++)
                      {
                        vm2[l]=vm1[l];
                        vpm2[l]=vpm1[l];
                      }
              }
          D=D_m;

          max_critical=MAX(nb_critical1[j][k],max_criti
cal);

          //Copy
          for(l=0;l<=nb_critical1[j][k];l++)
            {
```

```
                RF[j][k][l]=vm2[l];
                EQL[j][k][l]=vpm2[l];
              }
          }

      for(j=0;j<=i;j++)
        for(k=0;k<=i;k++)
          nb_critical[j][k]=nb_critical1[j][k];

    }

  price=EQL[0][0][0];

  /*Price*/
  return price;

}


static int cgmz(int am,double s0,NumFunc_1  *p,double tt,
    double sigma,double r0,double kappa,double theta,double omega,
    double rho,double D,double premium,double delta,double x_age,
    int n_step_for_period,double h_up,double *ptprice)
{
  double val;
  int n_obs,i;
  FILE* f_in;
  int life=1;

  lm_insurance=(double *)malloc(120*sizeof(double));
  f_in=fopen(infilename,"rb");

  //Read Statistics Mortality
  for(i=0;i<120;i++)
    {
      fscanf(f_in,"%lf{n",&val);
      lm_insurance[i]=val;
    }

  n_obs=(int)tt;
  Ns=n_obs*n_step_for_period;
```

```
  /*Memory Allocation*/
  memory_allocation(Ns+2);

  //Calibrate interest rate tree
  calibration(r0,s0,tt,kappa,omega,theta,rho,sigma,delta);

  //Compute Price of equity-linked policy
  *ptprice=compute_price(premium,r0,s0,tt,kappa,omega,thet
    a,rho,sigma,delta,D,n_obs,am,life,x_age,n_step_for_period,
    h_up);

  fclose(f_in);

   //Memory desallocation
  free_memory(Ns+2);
  free(lm_insurance);

  return OK;
}

int CALC(TR_cgmz)(void *Opt,void *Mod,PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  double kappa,theta,omega;

  omega=ptMod->Sigma.Val.V_PDOUBLE;
  kappa=ptMod->k.Val.V_PDOUBLE;
  theta=ptMod->theta.Val.V_PDOUBLE;
  infilename= ptMod->Mortality.Val.V_FILENAME;
  if(2*kappa*theta<SQR(omega))
    {
      Fprintf(TOSCREEN,"UNTREATED CASE{n{n{n");
      return WRONG;
    }
    else
      return cgmz(ptOpt->EuOrAm.Val.V_BOOL,ptMod->S0.Val.V_
    PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Maturity.Val.V_
    DATE-ptMod->T.Val.V_DATE,ptMod->Sigma.Val.V_PDOUBLE,ptMod->
```

```
    r0.Val.V_PDOUBLE,ptMod->k.Val.V_PDOUBLE,ptMod->theta.Val.V_
    PDOUBLE,ptMod->SigmaR.Val.V_PDOUBLE,ptMod->Rho.Val.V_PDOUB
    LE,ptOpt->DeemedContribution.Val.V_PDOUBLE,ptOpt->Premium.
    Val.V_PDOUBLE,ptOpt->MinimumGuaranteedInterestRate.Val.V_PDO
    UBLE,ptOpt->InitialAge.Val.V_PDOUBLE,Met->Par[0].Val.V_INT,
    Met->Par[1].Val.V_PDOUBLE,&(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(TR_cgmz)(void *Opt, void *Mod)
{
  if ((strcmp( ((Option*)Opt)->Name,"EquityLinkedSurrenderE
    ndowment")==0))
    return OK;
      return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_INT2=10;
      Met->Par[1].Val.V_PDOUBLE=0.001;
    }

  return OK;
}

PricingMethod MET(TR_cgmz)=
{
  "TR_CGMZ",
  {{"Step Numbers between Monitoring Dates",INT2,{100},ALL
    OW},{"Tollerance Error",PDOUBLE,{100},ALLOW},{" ",PREMIA_
    NULLTYPE,{0},FORBID}},
  CALC(TR_cgmz),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(TR_cgmz),
  CHK_tree,
```

```
  MET(Init)
};
```

References