

```
/* MODULO DE VALORACION DE CDOs SINTETICOS
```

```
* Modelo de Cópula Normal
```

```
* Lenguaje C
```

```
* Antonio Rivela. Nov 2003
```

```
*/
```

```
#include <stdio.h>
```

```
#include <nag.h>
```

```
#include <nag_stdlib.h>
```

```
#include <nagg05.h>
```

```
#include <nagg01.h>
```

```
double discount (double date)
```

```
{
```

```
double d=0,linear_rate;
```

```
double *swap_rate, *swap_date;
```

```
int tmax=17,a1,m;
```

```
/* Allocate memory */
```

```
if ( !(swap_rate = NAG_ALLOC(tmax, double)) ||
```

```
    !(swap_date = NAG_ALLOC(tmax, double)) )
```

```
{
```

```
    Vprintf("Allocation failure\n");
```

```
    d = -1;
```

```
    goto END_DISCOUNT;
```

```
}
```

```
swap_date[0]=0;
```

```
swap_date[1]=1;
```

```
swap_date[2]=2;
```

```
swap_date[3]=3;
```

```
swap_date[4]=4;
```

```
swap_date[5]=5;
```

```
swap_date[6]=6;
```

```
swap_date[7]=7;
```

```
swap_date[8]=8;
```

```
swap_date[9]=9;
swap_date[10]=10;
swap_date[11]=12;
swap_date[12]=15;
swap_date[13]=20;
swap_date[14]=25;
swap_date[15]=30;
swap_date[16]=35;
swap_date[17]=40;
```

```
swap_rate[0]=.0206;
swap_rate[1]=.025;
swap_rate[2]=.02908;
swap_rate[3]=.03278;
swap_rate[4]=.03568;
swap_rate[5]=.0379;
swap_rate[6]=.03975;
swap_rate[7]=.4;
swap_rate[8]=.4;
swap_rate[9]=.4;
swap_rate[10]=.4;
swap_rate[11]=.4;
swap_rate[12]=.4;
swap_rate[13]=.4;
swap_rate[14]=.4;
swap_rate[15]=.4;
swap_rate[16]=.4;
swap_rate[17]=.4;
```

```
for(a1=0; a1<tmax; a1++)
{
    if (swap_date[a1] > date)
    {
        m = a1;
```

```

        a1 = tmax;
    }
}

linear_rate = swap_rate[m-1] + (swap_rate[m] - swap_rate[m - 1]) * ((date - swap_date[m - 1]) / (swap_date[m] -
swap_date[m - 1]));

d= 1 / pow((1 + linear_rate),date);

return d;

END_DISCOUNT:

if (swap_rate) NAG_FREE(swap_rate);
if (swap_date) NAG_FREE(swap_date);

return d;
}

int main(void)
{
    /* Scalars */
    Integer igen, j, nr;
    Integer exit_status=0;
    Integer pdc;
    NagError fail;
    Nag_OrderType order;

    /* Arrays */
    double *c=0, *r=0, *x=0, *xmu=0, *inv_normal, *time, *intensity, *tmin, *time_ac, *loss;
    double *loss_e, *loss_m, *loss_ss, *input_spread;
    Integer iseed[4];
    Integer k,x1,y1;

    double intensity_e, intensity_m, intensity_ss, intensity_total, ac_e=0, ac_m=0, ac_ss=0, ac_total;

```

```

double spread,loss_ac=0,loss_e_ac=0,loss_m_ac=0,loss_ss_ac=0;
double spread_e,spread_m,spread_ss,duration=0;

/* ***** INPUT DATA***** */
double sub_m=0.25,sub_ss=0.50,thick_m;
double correlation=1      ;
double recovery=0;
Integer n_sim=5000;
Integer m=4;
double time_max=5;

#ifdef NAG_COLUMN_MAJOR
#define C(I,J) c[(J-1)*pdc + I - 1]
    order = Nag_ColMajor;
#else
#define C(I,J) c[(I-1)*pdc + J - 1]
    order = Nag_RowMajor;
#endif

INIT_FAIL(fail);

Vprintf("CREDIT ANALYTICS  MODULO DE VALORACION DE CDOs SINTETICOS. \n\nUn momento por favor \n \n");

thick_m=sub_ss-sub_m;

nr = (m+1)*(m+2)/2;
/* Allocate memory */
if ( !(c = NAG_ALLOC(m * m, double)) ||
    !(r = NAG_ALLOC((m+1)*(m+2)/2, double)) ||
    !(x = NAG_ALLOC(m, double)) ||
    !(inv_normal = NAG_ALLOC(m, double)) ||
    !(time = NAG_ALLOC(m, double)) ||
    !(intensity = NAG_ALLOC(m, double)) ||
    !(tmin = NAG_ALLOC(n_sim, double)) ||
    !(time_ac = NAG_ALLOC(m, double)) ||

```

```

!(loss = NAG_ALLOC(n_sim, double)) ||
!(loss_e = NAG_ALLOC(n_sim, double)) ||
!(loss_m = NAG_ALLOC(n_sim, double)) ||
!(loss_ss = NAG_ALLOC(n_sim, double)) ||
!(input_spread = NAG_ALLOC(m, double)) ||

!(xmu = NAG_ALLOC(m, double)) )
{
Vprintf("Allocation failure\n");
exit_status = -1;
goto END;
}

#ifdef NAG_COLUMN_MAJOR
pdc = m;
#else
pdc = m;
#endif

/* Initialise the seed to a repeatable sequence */
iseed[0] = 1762543;
iseed[1] = 9324783;
iseed[2] = 42344;
iseed[3] = 742355;

/* igen identifies the stream. */
igen = 1;
g05kbc(&igen, iseed);

for(x1=1;x1<=m;x1++)
for(y1=1;y1<=m;y1++)
{
C(x1,y1) = 1;
if(x1 != y1) C(x1,y1)=correlation;
}

for (x1=0;x1<m;x1++)
{

```

```

        xmu[x1] = 0;
        input_spread[x1]=0.01;
        intensity[x1] = input_spread[x1]/(1-recovery);
        loss[x1]=0;
    }

/* Set up reference vector and generate numbers */
g05lzc(order, 0, m, xmu, c, m, x, igen, iseed, r, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from g05lzc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Generate numbers */

for (k=0;k<n_sim;k++)
{

g05lzc(order, 2, m, xmu, c, m, x, igen, iseed, r, &fail);

if (fail.code != NE_NOERROR)
{
    Vprintf("Error from g05lzc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

for (j = 0; j < m; ++j)
{
    inv_normal[j]=g01eac(Nag_LowerTail,x[j],NAGERR_DEFAULT);

    time[j]=-log(1-inv_normal[j])/intensity[j];

```

```

        time_ac[j]=time_ac[j]+time[j];

        if(time[j]<=time_max) loss[k]=loss[k]+(1-recovery)*discount(time[j]); /* hay que multiplicar al 1 por el
discount factor .91 ?? */

/*      printf(time[j],"/",discount(time[j]),"/"); */
    }

    loss[k]=loss[k]/m;
/*      printf("...Loss(k)",loss[k]); */

}

for(j=0;j<m;++j)
    {
        time_ac[j]=time_ac[j]/n_sim;
/*      printf(j,"...Tmedio",time_ac[j],"/"); */
    }

for(k=0;k<n_sim;++k)
    {
        loss_ac=loss_ac+loss[k];

        if (loss[k]>=sub_ss)
        {
            loss_e[k]=sub_m;
            loss_m[k]=thick_m;
            loss_ss[k]=loss[k]-sub_ss;
        }

        if ((loss[k]>sub_m) & (loss[k]<sub_ss))
        {
            loss_e[k]=sub_m;
            loss_m[k]=loss[k]-sub_m;

```

```

        loss_ss[k]=0;
    }

    if (loss[k]<=sub_m)
    {
        loss_e[k]=loss[k];
        loss_m[k]=0;
        loss_ss[k]=0;
    }

    loss_e_ac=loss_e_ac+loss_e[k];
    loss_m_ac=loss_m_ac+loss_m[k];
    loss_ss_ac=loss_ss_ac+loss_ss[k];

/*      printf(k,"...Loss:",loss[k],"/");      */
    }

loss_ac=loss_ac/n_sim;
loss_e_ac=loss_e_ac/n_sim/sub_m;
loss_m_ac=loss_m_ac/n_sim/thick_m;
loss_ss_ac=loss_ss_ac/n_sim/(1-sub_ss);

intensity_total=-log(1-loss_ac)/time_max;
intensity_e=-log(1-loss_e_ac)/time_max;
intensity_m=-log(1-loss_m_ac)/time_max;
intensity_ss=-log(1-loss_ss_ac)/time_max;

for(x1=1;x1<=time_max;x1++)
{
    ac_total=ac_total+exp(-intensity_total*x1)*discount(x1)*365/360;
    ac_e=ac_e+exp(-intensity_e*x1)*discount(x1)*365/360;
    ac_m=ac_m+exp(-intensity_m*x1)*discount(x1)*365/360;
    ac_ss=ac_ss+exp(-intensity_ss*x1)*discount(x1)*365/360;
    duration=duration+discount(x1)*365/360;
}

```

```

spread=loss_ac/ac_total*100;
spread_e=loss_e_ac/ac_e*100;
spread_m=loss_m_ac/ac_m*100;
spread_ss=loss_ss_ac/ac_ss*100;

printf("N.Sim:",n_sim,"      N.Variables:",m,"\n \n");
printf("Intensity:",intensity[1,"\n");
printf("Recovery:",recovery,"      Correlation(r):",correlation,"\n");
printf("\n","Loss Ac:",loss_ac,"\n","Loss Equity:",loss_e_ac,"      Loss Mezz:",loss_m_ac,"      Loss SS:",loss_ss_ac,"\n");
printf("Riskless Duration:",duration,"\n");
printf("\nDuration Equity:",ac_e,"      Duration Mezz:",ac_m,"      Duration SS:",ac_ss,"\n \n");

printf("\nSpread:",spread);
printf("\nSpread Equity:",spread_e,"      Spread Mezz:",spread_m,"      Spread SS:",spread_ss,"\n \n");
printf("\n","      Sub.Mezz:",sub_m,"      Sub.SS:",sub_ss,"\n \n");

END: /*
if (c) NAG_FREE(c);
if (r) NAG_FREE(r);
if (x) NAG_FREE(x);
if (xmu) NAG_FREE(xmu);
if (inv_normal) NAG_FREE(inv_normal);
if (time) NAG_FREE(time);
if (intensity) NAG_FREE(intensity);
if (tmin) NAG_FREE(tmin);
if (time_ac) NAG_FREE(time_ac);
if (loss) NAG_FREE(loss);
if (loss_e) NAG_FREE(loss);
if (loss_m) NAG_FREE(loss);
if (loss_ss) NAG_FREE(loss);
if (input_spread) NAG_FREE(input_spread);*/
return exit_status;
}

```