Résolution d'un système tri-diagonal à l'aide de la fonction `choleski`

On considère l'équation matricielle :

$$Mx = y$$

où la matrice M (de dimensions $n \times n$) et le vecteur y (de dimension n) sont des données, et où le vecteur x (de dimension n) est l'inconnue.
La matrice est tri-diagonale c'est à dire que seuls les éléments $m_{i,i}$, $m_{i,i\pm1}$, $m_{i\pm1,i}$ de la diagonale principale et des deux sous-diagonales adjacentes à la diagonale principale sont non nuls :

$$\begin{pmatrix} b_0 & c_0 & 0 & ... & & & & & & & \\ a_1 & b_1 & c_1 & 0 & ... & & & & & & \\ 0 & a_2 & b_2 & c_2 & 0 & ... & & & & & \\ ... & 0 & a_3 & b_3 & c_3 & 0 & ... & & & & \\ & & & & & ... & & & & & \\ & & & & ... & 0 & a_{n-4} & b_{n-4} & c_{n-4} & 0 & ... \\ & & & & & ... & 0 & a_{n-3} & b_{n-3} & c_{n-3} & 0 \\ & & & & & & ... & 0 & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & & & & ... & 0 & a_{n-1} & b_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ ... \\ x_{n-4} \\ x_{n-3} \\ x_{n-2} \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ ... \\ y_{n-4} \\ y_{n-3} \\ y_{n-2} \\ y_{n-1} \end{pmatrix}$$

La fonction `choleski` de la bibliothèque du magistère résoud cette équation pour des valeurs réelles.
`a`, `b` et `c` étant trois tableaux à un indice et à `n` éléments on place les éléments de la matrice dans ces tableaux de la façon suivante :

`a[0]`$=0$, `a[1]`$=a_1$, ..., `a[n-1]`$=a_{n-1}$
`b[0]`$=b_0$, `b[1]`$=b_1$, ..., `b[n-1]`$=b_{n-1}$
`c[0]`$=c_0$, `c[1]`$=c_1$, ..., `c[n-1]`$=0$

`x` et `y` étant deux tableaux à un indice et à `n` éléments on place les n éléments du vecteur y dans le tableau `y` :

`y[0]`$=y_0$, `y[1]`$=y_1$, ..., `y[n-1]`$=y_{n-1}$

et après l'appel `choleski(a,b,c,x,y,n)`, si la valeur retournée par cette fonction est 0, le tableau `x` contient les éléments du vecteur x qui est la solution du système.

Exemple d'appel de `choleski` :

```
#define K 4
int main(void)
{
  int i;
  double p[K]={0.,1.,6.,2.};
  double q[K]={2.,1.,-2.,-3.};
  double r[K]={1.,-3.,1.,0.};
  double v[K]={7.,-10.,7.,13.};
  double u[K];
  choleski(p,q,r,u,v,K);
  for(i=0; i<K; i++) cout << u[i] << " " << endl;
}
```

Dans le cas de valeurs complexes, on utilise exactement la même méthode, en remplaçant la fonction `choleski` par la fonction `choleski_complex` si on utilise des tableaux ou des tableaux-pointeurs, ou la fonction `choleski_complex_vector` si on utilise des `vector`.

Exemple d'appel de `choleski_complex` :

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <bibli_fonctions.h>
#define K 4
int main(void)
{
```

```
  int i;
  complex<double> I(0.,1.);
  complex<double> p[K]={0.+I*0.,6.+I*2.,4.+I*8.,3.+I*9.};
  complex<double> q[K]={4.+I*7.,3.+I*7.,6.+I*5.,4.+I*3.};
  complex<double> r[K]={5.+I*8.,4.+I*3.,3.+I*9.,0.+I*0.};
  complex<double> v[K]={1.+I*8.,4.+I*3.,2.+I*6.,3.+I*7.};
  complex<double> u[K];
  choleski_complex(p,q,r,u,v,K);
  //for(i=0; i<K; i++)   cout << "real=" << real(u[i]) << " imag=" << imag(u[i]) << endl;
  for(i=0; i<K; i++) cout << i << " " << u[i] << endl;
  // Verification :
  cout << q[0]*u[0]+r[0]*u[1] << v[0] << endl;
  for(i=1; i<K-1; i++) cout << p[i]*u[i-1]+q[i]*u[i]+r[i]*u[i+1] << v[i] << endl;
  cout << p[K-1]*u[K-2]+q[K-1]*u[K-1] << v[K-1] << endl;
  return 0;
}
```

Référence : J.P. Nougier, Méthodes de calcul numérique, page 29.