Método de Factorización de Cholesky

La factorización o descomposición de Cholesky se basa en que una matriz simétrica definida positiva que puede ser descompuesta como el producto de una matriz triangular inferior y la traspuesta de la matriz triangular inferior. La matriz triangular inferior es el triángulo de Cholesky de la matriz original definida positiva.

Una matriz M con coeficientes reales se considera definida positiva si para todo vector x con coeficientes reales se cumple que $x^T.M.x > 0$

El resultado de Cholesky ha sido extendido a matrices con entradas complejas. Es una manera de resolver sistemas de ecuaciones matriciales y se deriva de la factorización LU con una pequeña variación.

Cualquier matriz cuadrada A con pivotes no nulos puede ser escrita como el producto de una matriz triangular inferior L y una matriz triangular superior U; esto recibe el nombre de factorización LU. Sin embargo, si A es simétrica y definida positiva, se pueden escoger los factores tales que, U es la transpuesta de L, y esto se llama la descomposición o factorización de Cholesky.

Tanto la descomposición LU como la descomposición de Cholesky son usadas para resolver sistemas de ecuaciones lineales. Cuando es aplicable, la descomposición de Cholesky es dos veces más eficiente que la descomposición LU.

La descomposición de Cholesky se usa principalmente para hallar la solución numérica de ecuaciones lineales $Ax = b$. Si A es simétrica y definida positiva, entonces se puede solucionar $Ax = b$ calculando primero la descomposición de Cholesky $A = LL^T$, resolviendo $Ly = b$ para y, para finalmente resolver $L^T x = y$ para x.

Algoritmo de Cholesky

Si la matriz A es simétrica y definida positiva, entonces existe una matriz triangular inferior L tal que $A = LL^T$. Este es solo un caso especial de la descomposición LU, donde $U = L^T$.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}, L = \begin{pmatrix} l_{1,1} & l_{1,2} & \dots & l_{1,n} \\ l_{2,1} & l_{2,2} & \dots & l_{2,n} \\ \dots \\ l_{n,1} & l_{n,2} & \dots & l_{n,n} \end{pmatrix}$$

Para $k = 1, \dots, n$,

$$l_{k,k} = \sqrt{a_{k,k} - \sum_{r=1}^{k-1} l_{k,r}^2}$$

$$l_{i,k} = \frac{a_{i,k} - \sum_{r=1}^{k-1} l_{i,r} l_{k,r}}{l_{k,k}} i = k + 1, \dots, n$$

Complejidad algorítmica: $O(n^3)$

In [3]:

```
# Ejemplo factorización de Cholesky
import scipy.linalg as la
import numpy as np

A1 = np.array([[6, 3, 4, 8], [3, 6, 5, 1], [4, 5, 10, 7], [8, 1, 7, 25]])
A1
```

Out[3]:

```
array([[ 6,  3,  4,  8],
       [ 3,  6,  5,  1],
       [ 4,  5, 10,  7],
       [ 8,  1,  7, 25]])
```

In [4]:

```
L = la.cholesky(A1, lower=True)
U = la.cholesky(A1, lower=False)
```

In [5]:

```
L
```

Out[5]:

```
array([[ 2.44948974,  0.        ,  0.        ,  0.        ],
       [ 1.22474487,  2.12132034,  0.        ,  0.        ],
       [ 1.63299316,  1.41421356,  2.30940108,  0.        ],
       [ 3.26598632, -1.41421356,  1.58771324,  3.13249102]])
```

Podemos ver que la matriz es triangular inferior.

In [6]:

```
U
```

Out[6]:

```
array([[ 2.44948974,  1.22474487,  1.63299316,  3.26598632],
       [ 0.        ,  2.12132034,  1.41421356, -1.41421356],
       [ 0.        ,  0.        ,  2.30940108,  1.58771324],
       [ 0.        ,  0.        ,  0.        ,  3.13249102]])
```

Podemos ver que la matriz es triangular superior.

In [7]:

```
L @ U
```

Out[7]:

```
array([[ 6.,  3.,  4.,  8.],
       [ 3.,  6.,  5.,  1.],
       [ 4.,  5., 10.,  7.],
       [ 8.,  1.,  7., 25.]])
```

In [8]:

```
A1
```

Out[8]:

```
array([[ 6,  3,  4,  8],
       [ 3,  6,  5,  1],
       [ 4,  5, 10,  7],
       [ 8,  1,  7, 25]])
```

Podemos ver que el producto de ambas matrices se corresponde con la matriz original.

Veamos otro ejemplo

In [11]:

```
A2 = np.array([[2, -1, 0],[-1, 2, -1.], [0, -1, 2.]])
A2
```

Out[11]:

```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  2.]])
```

In [12]:

```
L = la.cholesky(A2, lower=True)
U = la.cholesky(A2, lower=False)
L
```

Out[12]:

```
array([[ 1.41421356,  0.        ,  0.        ],
       [-0.70710678,  1.22474487,  0.        ],
       [ 0.        , -0.81649658,  1.15470054]])
```

La matriz L es triangular inferior

In [13]:

```
U
```

Out[13]:

```
array([[ 1.41421356, -0.70710678,  0.        ],
       [ 0.        ,  1.22474487, -0.81649658],
       [ 0.        ,  0.        ,  1.15470054]])
```

La matriz U es triangular superior

In [14]:

```
L @ U
```

Out[14]:

```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  2.]])
```

In [15]:

```
A2
```

Out[15]:

```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  2.]])
```

Y vemos que las dos matrices se corresponden

Veamos la implementación siguiente del método de Cholesky

In [16]:

```
def cholesky(A):

    n = A.shape[0]

    L = np.zeros((n, n), dtype=np.double)

    for k in range(n):

        L[k, k] = np.sqrt(A[k, k] - np.sum(L[k, :] ** 2))

        L[(k+1):, k] = (A[(k+1):, k] - L[(k+1):, :] @ L[:, k]) / L[k, k]

    return L
```

In [17]:

```
L = cholesky(A2)
L
```

Out[17]:

```
array([[ 1.41421356,  0.        ,  0.        ],
       [-0.70710678,  1.22474487,  0.        ],
       [ 0.        , -0.81649658,  1.15470054]])
```

Para comprobar el resultado, podemos multiplicar la matriz L por su transpuesta

In [18]:

```
L @ L.transpose()
```

Out[18]:

```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  2.]])
```

In [19]:

```
A2
```

Out[19]:

```
array([[ 2., -1.,  0.],
       [-1.,  2., -1.],
       [ 0., -1.,  2.]])
```

Como vemos, también funciona bien con este ejemplo.

Bibliografía

1. Factorización de Cholesky (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_Cholesky)
2. Tema 3: Resolución de Sistemas de Ecuaciones Lineales (https://www.ugr.es/~mpasadas/ftp/Tema3_apuntes.pdf)
3. LU Factorization (https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_LU.html)
4. Práctica Descomposiciones matriciales LU y Choleski (https://personal.us.es/pmr/images/pdfs/1314gm-cnii-practica1bis.pdf)

In [ ]:

```

```