Factorización

En matemáticas, la factorización es una técnica que consiste en la descomposición en factores de una expresión algebraica (que puede ser un número, una suma o resta, una matriz, un polinomio, etc.) en forma de producto. Existen distintos métodos de factorización, dependiendo de los objetos matemáticos estudiados. El objetivo es simplificar una expresión o reescribirla en términos de «bloques fundamentales», que reciben el nombre de factores, como por ejemplo un número en números primos, o un polinomio en polinomios irreducibles.

En álgebra lineal la factorización de una matriz es la descomposición de esta como producto de dos o más matrices según una forma canónica. Según sus aplicaciones a la resolución de sistemas de ecuaciones lineales, cálculo de determinantes e inversión de matrices podemos distinguir los siguientes tipos de factorizaciones: Factorización LU, Factorización de Cholesky, y Descomposición en valores singulares entre otras.

Factorización LU

En el álgebra lineal, la factorización o descomposición LU (del inglés Lower-Upper) es una forma de factorización de una matriz como el producto de una matriz triangular inferior y una superior. Debido a la inestabilidad de este método, deben tenerse en cuenta algunos casos especiales, por ejemplo, si uno o varios elementos de la diagonal principal de la matriz a factorizar es cero, es necesario pre multiplicar la matriz por una o varias matrices elementales de permutación. Existe un segundo método llamado factorización $PA = LU$ con pivote. Esta descomposición se usa en el análisis numérico para resolver sistemas de ecuaciones (más eficientemente) o encontrar las matrices inversas.

Cualquier matriz A no singular se puede factorizar en una matriz triangular inferior L y una matriz triangular superior U utilizando procedimientos que ya hemos establecido con eliminación gaussiana. Esto resulta muy útil para el cálculo numérico y, de hecho, es una de las formas más comunes en que la mayoría de los solucionadores de álgebra lineal resuelven sistemas lineales no dispersos (non-sparse).

La factorización o descomposición LU (del inglés Lower-Upper) es una forma de factorización de una matriz como el producto de una matriz triangular inferior y una superior. Cualquier matriz A no singular se puede factorizar en una matriz triangular inferior L y una matriz triangular superior U utilizando procedimientos que ya hemos establecido con eliminación gaussiana.

Sea A una matriz invertible. Tenemos que $A = LU$, donde L y U son matrices inferiores y superiores triangulares respectivamente. Por ejemplo:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

Identificando los elementos de la primera fila de A con los correspondientes de LU, y los de la primera columna de A con los correspondientes de LU, obtenemos:

$$a_{1,j} = u_{1,j}, j = 1, \ldots, n$$

$$a_{i,1} = l_{i,1}u_{1,1} \Rightarrow l_{i,1} = \frac{a_{i,1}}{}, i = 2, \ldots, n$$

Identificando a continuación los elementos de la segunda fila de A con los correspondientes de LU, y los de la segunda columna de A con los correspondientes de LU, obtenemos:

$$a_{2,j} = l_{2,1}u_{1,j} + u_{2,j} \Rightarrow$$
$$u_{2,j} = a_{2,j} - l_{2,1}u_{1,j}, j = 2, \ldots, n$$

$$a_{i,2} = l_{i,1}u_{1,2} + l_{i,2}u_{2,2} \Rightarrow$$
$$l_{i,2} = \frac{1}{u_{2,2}}(a_{i,2} - l_{i,1}u_{1,2}), i = 3, \ldots, n$$

Y en general, si suponemos conocidas las $k-1$ primeras filas de U y las $k-1$ primeras columnas de L, entonces, identificando los elementos de la k-ésima fila de A con los correspondientes de LU, y los de la k-ésima columna de A con los correspondientes de LU, obtenemos:

$$a_{k,j} = l_{k,1}u_{1,j} + \ldots + l_{k,k-1}u_{k-1,j} + u_{k,j} \Rightarrow$$

$$u_{k,j} = a_{k,j} - \sum_{r=1}^{k-1} l_{k,r}u_{r,j}, j = k, \ldots, n$$

$$a_{i,k} = l_{i,1}u_{1,k} + \ldots + l_{i,k-1}u_{k-1,k} + l_{i,k}u_{k,k} \Rightarrow$$

$$l_{i,k} = \frac{1}{u_{k,k}}\left(a_{i,k} - \sum_{r=1}^{k-1} l_{i,r}u_{r,k} \right), i = k+1, \ldots, n$$

El algoritmo general sería:

Para $k = 1, \ldots, n$

$$l_{k,k}u_{k,k} = a_{k,k} - \sum_{r=1}^{k-1} l_{k,r}u_{r,k};$$

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

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

Pseudocódigo para una factorización LU simple

Considere las matrices:

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

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ u_{2,1} & u_{2,2} & \dots & u_{2,n} \end{pmatrix}$$

Inicializamos L con la matriz identidad I de $n \times n$ y $U = A$

For $i = 1, \dots, n$ do

For $j = i + 1, \dots, n$ do

$l_{j,i} = u_{j,i}/u_{i,i}$

$U_j = (U_j - l_{j,i}U_i)$

donde U_i, U_j representan las filas i y j de la matriz U respectivamente.

Coste general de la factorización: $O(n^3)$

Ejemplo:

In [1]:

```
import numpy as np

A1 = np.array([ [1.,1.,0.], [2.,1.,-1.], [3.,-1.,-1.]])
A1
```

Out[1]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

In [2]:

```
# Las instrucciones de asignación en Python no copian objetos,
# crean enlaces entre un objetivo y un objeto.
# Para las colecciones que son mutables o contienen elementos mutables,
# a veces se necesita una copia para poder cambiar uno de
# los objetos copiados sin afectar al otro.
U = np.copy(A1)
U
```

Out[2]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

In [3]:

```
L = np.identity(3)
L
```

Out[3]:

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

In [4]:

```
# NOTA: observe que los índices en Python comienzan en cero a diferencia de
# cómo se acostumbra en Matemáticas
i = 0 # primera columna
j = 1 # segunda fila
L[j,i] = U[j,i]/U[i,i]
L
```

Out[4]:

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

In [5]:

```
U[j] = U[j]-L[j,i]*U[i]
U
```

Out[5]:

```
array([[ 1.,  1.,  0.],
       [ 0., -1., -1.],
       [ 3., -1., -1.]])
```

In [6]:

```
j=2 # tercera fila, la i se mantiene igual
L[j,i] = U[j,i]/U[i,i]
L
```

Out[6]:

```
array([[1., 0., 0.],
       [2., 1., 0.],
       [3., 0., 1.]])
```

In [7]:

```
U[j] = U[j]-L[j,i]*U[i]
U
```

Out[7]:

```
array([[ 1.,  1.,  0.],
       [ 0., -1., -1.],
       [ 0., -4., -1.]])
```

In [8]:

```
# incrementamos la i y la j comienza en i+1
i = 1
j = 2
L[j,i] = U[j,i]/U[i,i]
L
```

Out[8]:

```
array([[1., 0., 0.],
       [2., 1., 0.],
       [3., 4., 1.]])
```

Con lo cual hemos obtenido la matriz inferior

In [9]:

```
U[j] = U[j]-L[j,i]*U[i]
U
```

Out[9]:

```
array([[ 1.,  1.,  0.],
       [ 0., -1., -1.],
       [ 0.,  0.,  3.]])
```

Con lo cual hemos obtenido la matriz triangular superior

Podemos verificar nuestros resultados utilizando Python. Al multiplicar la matriz L por la matriz U, obtenemos la matriz A original.

El operador @ es preferible a otros métodos cuando se calcula el producto matricial entre arreglos de 2 dimensiones. La función numpy.matmul implementa el operador @.

In [10]:

```
L @ U
```

Out[10]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

In [11]:

```
A1
```

Out[11]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

La librería Scipy de Python tiene un módulo dedicado a Álgebra Lineal, el cual incluye una función o método que implementa la factorización LU. El método aquí implementado es más general que el que hemos estudiado anteriormente.

In [12]:

```
import scipy.linalg as la

L, U = la.lu(A1, permute_l=True, overwrite_a=False)
L
```

Out[12]:

```
array([[0.33333333, 0.8       , 1.        ],
       [0.66666667, 1.        , 0.        ],
       [1.        , 0.        , 0.        ]])
```

In [13]:

```
U
```

Out[13]:

```
array([[ 3.        , -1.        , -1.        ],
       [ 0.        ,  1.66666667, -0.33333333],
       [ 0.        ,  0.        ,  0.6       ]])
```

Como podemos ver, ninguna de las dos matrices se corresponde con las matrices que habíamos calculado.

In [14]:

```
L @ U
```

Out[14]:

```
array([[ 1.00000000e+00,  1.00000000e+00, -1.11022302e-16],
       [ 2.00000000e+00,  1.00000000e+00, -1.00000000e+00],
       [ 3.00000000e+00, -1.00000000e+00, -1.00000000e+00]])
```

In [15]:

```
A1
```

Out[15]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

Pero el producto sí se corresponde a la matriz original.

In [16]:

```
# Numpy ofrece otro medio de verificar el resultado
# np.allclose devuelve True si dos arreglos son iguales elemento a elemento
# con una cierta tolerancia.

np.allclose(A1 - L @ U, np.zeros((3,3)))
```

Out[16]:

```
True
```

La función lu descompone la matriz de la siguiente forma:

$$A = PLU$$

donde L es una matriz triangular inferior con unos en la diagonal, U es una matriz triangular superior y P es una matriz de permutación. Dado que no todas las matrices tienen descomposición LU, este método trata de encontrar una matriz de permutación P tal que PA tenga descomposición LU: $PA = LU$, donde L y U son nuevamente matrices triangulares inferior y superior, y P es una matriz de permutación. En este caso, decimos que A tiene una factorización PLU.

In [17]:

```
P, L, U = la.lu(A1)
```

In [18]:

```
P
```

Out[18]:

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

In [19]:

```
L
```

Out[19]:

```
array([[1.        , 0.        , 0.        ],
       [0.66666667, 1.        , 0.        ],
       [0.33333333, 0.8       , 1.        ]])
```

In [20]:

```
U
```

Out[20]:

```
array([[ 3.        , -1.        , -1.        ],
       [ 0.        ,  1.66666667, -0.33333333],
       [ 0.        ,  0.        ,  0.6       ]])
```

In [21]:

```
P @ L @ U
```

Out[21]:

```
array([[ 1.00000000e+00,  1.00000000e+00, -1.11022302e-16],
       [ 2.00000000e+00,  1.00000000e+00, -1.00000000e+00],
       [ 3.00000000e+00, -1.00000000e+00, -1.00000000e+00]])
```

In [22]:

```
A1
```

Out[22]:

```
array([[ 1.,  1.,  0.],
       [ 2.,  1., -1.],
       [ 3., -1., -1.]])
```

Las matrices L y U obtenidas nuevamente no se corresponden con las obtenidas manualmente, pero al multiplicarlas sí se obtiene la matriz original.

In [23]:

```
# Numpy ofrece otro medio de verificar el resultado
#

np.allclose(A1 - P @ L @ U, np.zeros((3,3)))
```

Out[23]:

```
True
```

In [24]:

```
# Veamos otro ejemplo

A2 = np.array([[2, 5, 8, 7], [5, 2, 2, 8], [7, 5, 6, 6], [5, 4, 4, 8]])
A2
```

Out[24]:

```
array([[2, 5, 8, 7],
       [5, 2, 2, 8],
       [7, 5, 6, 6],
       [5, 4, 4, 8]])
```

In [25]:

```
P, L, U = la.lu(A2)
P
```

Out[25]:

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

In [26]:

```
L
```

Out[26]:

```
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.28571429,  1.        ,  0.        ,  0.        ],
       [ 0.71428571,  0.12      ,  1.        ,  0.        ],
       [ 0.71428571, -0.44      , -0.46153846,  1.        ]])
```

In [27]:

```
U
```

Out[27]:

```
array([[ 7.        ,  5.        ,  6.        ,  6.        ],
       [ 0.        ,  3.57142857,  6.28571429,  5.28571429],
       [ 0.        ,  0.        , -1.04      ,  3.08      ],
       [ 0.        ,  0.        ,  0.        ,  7.46153846]])
```

In [28]:

```
L @ U
```

Out[28]:

```
array([[7., 5., 6., 6.],
       [2., 5., 8., 7.],
       [5., 4., 4., 8.],
       [5., 2., 2., 8.]])
```

In [29]:

```
# podemos comparar el resultado con A2

A2
```

Out[29]:

```
array([[2, 5, 8, 7],
       [5, 2, 2, 8],
       [7, 5, 6, 6],
       [5, 4, 4, 8]])
```

Observe que algunas filas de la matriz han cambiado de posición pero son matrices equivalentes.

In [30]:

```
np.allclose(A2 - P @ L @ U, np.zeros((4, 4)))
```

Out[30]:

```
True
```

Bibliografía

- Factorización (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n)
- Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)
- Factorización LU (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU)
- LU Factorization (https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_LU.html)
- Tema 3 Resolución de Sistemas de Ecuaciones Lineales (https://www.ugr.es/~mpasadas/ftp/Tema3_apuntes.pdf)
- PLU Factorization (https://www.cfm.brown.edu/people/dobrush/cs52/Mathematica/Part2/PLU.html)

In [ ]: