# Factorización QR

La descomposición o factorización $QR$ de una matriz es una descomposición de la misma como producto de una matriz ortogonal por una matriz triangular superior. La descomposición $QR$ es la base del algoritmo utilizado para el cálculo de los vectores y valores propios de una matriz.

La descomposición $QR$ de una matriz cuadrada real $A$ se representa como:

$$A = QR$$

donde $Q$ es una matriz ortogonal ($Q^TQ = I$) y $R$ es una matriz triangular superior.

Existen diferentes métodos para el cálculo de la descomposición $QR$ como el método de ortogonalización de Gram-Schmidt, la transformación de Householder y mediante las rotaciones de Givens.

El proceso de ortogonalización de Gram–Schmidt es un algoritmo para construir, a partir de un conjunto de vectores de un espacio vectorial, otro conjunto ortonormal de vectores que genere el mismo subespacio vectorial. Si bien la aplicación de las proyecciones tiene una analogía geométrica atractiva con la ortogonalización, la ortogonalización en sí es propensa al error numérico. El proceso es inherentemente numéricamente inestable. Una ventaja significativa es la facilidad de implementación.

El método de Householder tiene una estabilidad numérica mayor que la del método de Gram-Schmidt. Una pequeña variación del método de Householder se utiliza para obtener matrices semejantes con la forma de Hessenberg, muy útiles en el cálculo de autovalores por acelerar la convergencia del algoritmo QR reduciendo así enormemente su coste computacional.

El procedimiento de rotación de Givens es útil en situaciones donde sólo pocos elementos fuera de la diagonal necesitan ser anulados y es más fácil de paralelizar que las transformaciones de Householder.

El uso de transformaciones de Householder es inherentemente el más simple de los algoritmos de descomposición $QR$ numéricamente estables debido al uso de reflejos como mecanismo para producir ceros en la matriz $R$. Sin embargo, el algoritmo de reflexión de Householder tiene un gran ancho de banda y no se puede paralelizar, ya que cada reflexión que produce un nuevo elemento cero cambia la totalidad de las matrices $Q$ y $R$.

Ver [Transformación de Householder (http://localhost:8888/notebooks/Mi_unidad/ASoto(GDrive)/My_Projects_GDrive/UnivIntVal/Transformaci%C3](http://localhost:8888/notebooks/Mi_unidad/ASoto(GDrive)/My_Projects_GDrive/UnivIntVal/Transformaci%C3) para más detalles sobre el proceso.

In [2]:

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

A = np.array([[12, -51, 4],
              [6, 167, -68],
              [-4, 24, -41]])
Q, R = la.qr(A)
```

In [3]:

```
Q
```

Out[3]:

```
array([[-0.85714286,  0.39428571,  0.33142857],
       [-0.42857143, -0.90285714, -0.03428571],
       [ 0.28571429, -0.17142857,  0.94285714]])
```

In [4]:

```
R
```

Out[4]:

```
array([[ -14.,  -21.,   14.],
       [   0., -175.,   70.],
       [   0.,    0.,  -35.]])
```

Para verificar si Q es ortogonal podemos verificar si $Q^T Q = I$

In [5]:

```
Q.transpose() @ Q
```

Out[5]:

```
array([[ 1.00000000e+00, -5.04131884e-17, -3.39864191e-17],
       [-5.04131884e-17,  1.00000000e+00,  2.30881074e-17],
       [-3.39864191e-17,  2.30881074e-17,  1.00000000e+00]])
```

Podemos ver que los coeficientes de l diagonal son iguales a 1 y que los coeficientes restantes son menores que $10^{-17}$, lo cual se puede considerar igual a cero teniendo en cuenta la precisión limitada del ordenador.

Por otra parte, si multiplicamos Q por R debemos obtener la matriz A original.

In [6]:

```
Q @ R
```

Out[6]:

```
array([[ 12., -51.,   4.],
       [  6., 167., -68.],
       [ -4.,  24., -41.]])
```

In [7]:

```
A
```

Out[7]:

```
array([[ 12, -51,   4],
       [  6, 167, -68],
       [ -4,  24, -41]])
```

lqqd: lo que queda demostrado

Veamos otro ejemplo

In [8]:

```
A = np.array([[12, -51, 4], [6, 167, -68], [-4, 24, -41]])
A
```

Out[8]:

```
array([[ 12, -51,   4],
       [  6, 167, -68],
       [ -4,  24, -41]])
```

In [9]:

```
Q, R = la.qr(A)
Q
```

Out[9]:

```
array([[-0.85714286,  0.39428571,  0.33142857],
       [-0.42857143, -0.90285714, -0.03428571],
       [ 0.28571429, -0.17142857,  0.94285714]])
```

In [10]:

```
R
```

Out[10]:

```
array([[ -14.,  -21.,   14.],
       [   0., -175.,   70.],
       [   0.,    0.,  -35.]])
```

In [11]:

```
# verificamos si Q es ortogonal
Q.transpose() @ Q
```

Out[11]:

```
array([[ 1.00000000e+00, -5.04131884e-17, -3.39864191e-17],
       [-5.04131884e-17,  1.00000000e+00,  2.30881074e-17],
       [-3.39864191e-17,  2.30881074e-17,  1.00000000e+00]])
```

Al igual que en el caso anterior, los coeficientes de la diagonal son unitarios y los demás coeficientes se pueden considerar iguales a cero.

Probemos a obtener A como producto de Q y R

In [12]:

```
Q @ R
```

Out[12]:

```
array([[ 12., -51.,   4.],
       [  6., 167., -68.],
       [ -4.,  24., -41.]])
```

In [13]:

```
A
```

Out[13]:

```
array([[ 12, -51,   4],
       [  6, 167, -68],
       [ -4,  24, -41]])
```

# Bibliografía


- Factorización QR (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_QR)
- Numpy LinAlg QR (https://numpy.org/doc/stable/reference/generated/numpy.linalg.qr.html)
- La Factorización QR - Uprm (https://academic.uprm.edu/eacuna/lec3comp.pdf)
- Factorización QR MAT–251 (https://www.cimat.mx/~alram/met_num/clases/clase12.pdf)
- Linear algebra (scipy.linalg) (https://docs.scipy.org/doc/scipy/reference/linalg.html)
- LAPACK 3.11.0 Linear Algebra PACKage (https://netlib.org/lapack/explore-html/dd/d9a/group__double_g_ecomputational_ga3766ea903391b5cf9008132f7440ec7b.html)
- QR decomposition (https://en.wikipedia.org/wiki/QR_decomposition)


In [ ]:

```

```