9. *Décomposition QR*

Pour $j \in [\![1,n]\!]$, notons e_j le j-ième vecteur colonne de la matrice A. Comme A est inversible, la famille $(e_1,\dots,e_n)$ est libre (c'est une base de $E=\mathbb{R}^n$!). Je lui applique l'algorithme d'orthonormalisation de Gram-Schmidt en posant :

$$v_1 = e_1\ , \quad \varepsilon_1 = \frac{1}{\|v_1\|}.v_1 \quad \text{et, pour } j \in [\![2,n]\!] \quad v_j = e_j - \sum_{k=1}^{j-1} (\varepsilon_k|e_j)\,.\varepsilon_k\ , \quad \varepsilon_j = \frac{1}{\|v_j\|}.v_j$$

J'ai alors en "retournant" les formules, pour $j \in [\![1,n]\!]$, puisque $v_j = \|v_j\|\,.\varepsilon_j$:

$$e_j = \sum_{k=1}^{j-1} (\varepsilon_k|e_j)\,.\varepsilon_k + \|v_j\|\,.\varepsilon_j = \sum_{k=1}^{n} r_{k,j}.\varepsilon_k \tag{1}$$

où j'ai posé (habilement)

$$\forall\,(k,j) \in [\![1,n]\!]^2 \quad r_{k,j} = \left|\begin{array}{l} (\varepsilon_k|e_j) \text{ si } k<j \\ \|v_j\| \text{ si } k=j \\ 0 \text{ si } k>j \end{array}\right. .$$

En notant $R=(r_{i,j})_{1\leq i,j\leq n}$ et Q la matrice dans la base canonique de la famille $(\varepsilon_1,\dots,\varepsilon_n)$, je reconnais les combinaisons de colonnes correspondant au produit matriciel : $A = Q \times R$. Pour s'en convaincre, dessiner les matrices... Pour le démontrer, écrire la i-ième coordonnée (dans la base canonique) du vecteur e_j de la relation (1) ci-dessus :

$$a_{i,j} = \sum_{k=1}^{n} q_{i,k} r_{k,j}$$

où l'on reconnaît la définition du produit de $Q=(q_{i,j})$ par R !

Or par construction $Q \in O_n(\mathbb{R})$ car $(\varepsilon_1,\dots,\varepsilon_n)$ est orthonormale et R est triangulaire supérieure par définition !

Compte tenu du coût en $O(n)$ d'un produit scalaire, le coût de la décomposition QR est en $O\left(n^3\right)$, comme celui de la résolution du système $AX=B$, pour B donné dans $\mathbb{R}^n$. Mais si l'on doit résoudre de nombreux systèmes de cette forme avec divers second membres et **la même** matrice A, la méthode devient rentable car chaque nouvelle résolution se fait en $O\left(n^2\right)$: comme Q est orthogonale,

$$AX = B \Leftrightarrow RX = {}^tQB$$

et l'on est ramené à un système triangulaire.

N.B. : comme pour Gram-Schmidt, on peut montrer l'unicité de la décomposition QR de A si l'on ajoute la condition que les coefficients diagonaux de R sont positifs.

Programmation en Python

Je suppose le module `numpy` chargé avec l'alias habituel `np`. Après avoir programmé le calcul du produit scalaire et de la norme euclidienne canoniques, je recopie l'algorithme ci-dessus, sans oublier les transpositions nécessaires car `A[j]` donne la `j`-ième ligne du tableau `numpy A` ! Noter aussi que j'utilise le même vecteur `v` à chaque passage dans la boucle : c'est la matrice `epsilon` qui sert à mémoriser tous les vecteurs calculés de proche en proche.
Je remplis la matrice `R` au fur et à mesure et `Q` est la transposée de `epsilon`, à cause du remplissage ligne par ligne...

```
def pscal(u, v):
    n = len(u)
    s = 0
    for k in range(n):
        s += u[k]*v[k]
    return s

def norme(u):
    return sqrt(pscal(u, u))
```

```
def QR(A):
    n = len(A)
    e = np.transpose(A);
    R = np.zeros((n, n))
    epsilon = np.zeros((n, n))
    for j in range(n):
        v = e[j]
        for k in range(j):
            R[k, j] = pscal(epsilon[k], e[j])
            v = v - R[k, j]*epsilon[k]
        R[j, j] = norme(v)
        epsilon[j] = v/R[j, j]
    return np.transpose(epsilon), R
```