1 Algorithme du *solver*

Le but de ce devoir était d'implémenter une factorisation QR, et de résoudre un système linéaire donné sous forme matricielle utilisant celle-ci. L'implémentation est divisée en deux parties :

— La fonction `QRfactorise` qui calcule, pour une matrice $A \in \mathbb{R}^{n\times n}$, la matrice triangulaire supérieure R, ainsi qu'une matrice V reprenant les différents réflecteurs de Householder.

— La fonction `QRsolve`, qui résout le système $Ax = b$ en utilisant la fonction `QRfactorise` pour factoriser A, et qui utilise ensuite une factorisation arrière pour trouver x, sans devoir calculer explicitement Q.

L'exactitude de l'implémentation a été vérifiée en comparant ses résultats avec ceux de la fonction de la librairie `SciPy`.

2 Complexité

2.1 `QRfactorise`

La complexité de la fonction `QRfactorise`

2.2 `QRsolve`

Dans la fonction `QRsolve`, on fait appel à `QRfactorise`, cependant dans cette section on détaille uniquement le coût algorithmique propre à la fonction. Celui-ci est dominé par la substitution arrière, dont la complexité est

$$b(n) \sim \sum_{j=1}^{n} \big(2(n-j)+1\big) \sim 2\sum_{k=0}^{n-1} k + n \sim n(n-1) + n \sim n^2\,.$$

2.3 Totale

Comme la complexité $t(m,n)$ qui nous intéresse ici est asymptotique, et que `QRsolve` et `QRfactorise` se font séquentiellement, on peut laisser tomber les termes d'ordre inférieur dus à la substitution arrière. On obtient donc une complexité totale $t(m,n)$ de

$$t(m,n) \sim 2mn^2 - \frac{2}{3}n^3 + n^2 \sim 2mn^2 - \frac{2}{3}n^3 \overset{m=n}{\Longrightarrow} t(n) \sim \frac{4}{3}n^3\,.$$

3 Résultats