Analyse Numérique : Devoir 2
Factorisations matricielles

1 LU

1.1 Factorisation

Cette factorisation a pour but de factoriser la matrice initiale A en une matrice *triangulaire inférieure* L et une matrice *triangulaire supérieure* U ainsi, A = LU. Cette factorisation cherche d'abord a faire un *pivotage partiel*, ainsi les lignes sont échangées pour mettre les éléments maximums sous la diagonale sur l'élément de la diagonale, de cette manière on évitera par la suite de diviser par zéro lorsque l'on divisera par le pivot. Ensuite, on applique les *éliminiations de Gauss* de manière itérative sur chaque éléments de L et de U. Comme nous sommes en Python, j'ai cherché à vectoriser un maximum de boucles, ce qui nous donne l'algorithme de 3 lignes suivant une fois la matrice pivotée:

```python
def LU(A):
    """
    Args:
        A (numpy.array) : Matrix to be factorized as LU
    Returns:
        LU (numpy.array) : The lower and the upper matrix squeezed
    """
    LU = A
    n = LU.shape[0]
    for k in range(n):
        LU[k+1:,k] = np.divide(LU[k+1:,k], LU[k,k])
        LU[k+1:,k+1:] -= np.einsum('i,j->ij' ,LU[k+1:,k], LU[k,k+1:] )
    return LU
```

1.2 Résolution

Ayant A comme étant deux *matrices triangulaires*, nous pouvons résoudre de la manière suivante en appliquant pour la résolution avec L, une substitution avant et avec U, une substitiution arrière.

$$\begin{aligned} Ax &= b \\ LUx &= b \\ Ly &= b \\ Ux &= y \end{aligned}$$

2 Factorisation QR

3 Analyse

3.1 Complexité temporelle sur maillages différents

3.2 Complexité temporelle sur régimes différents

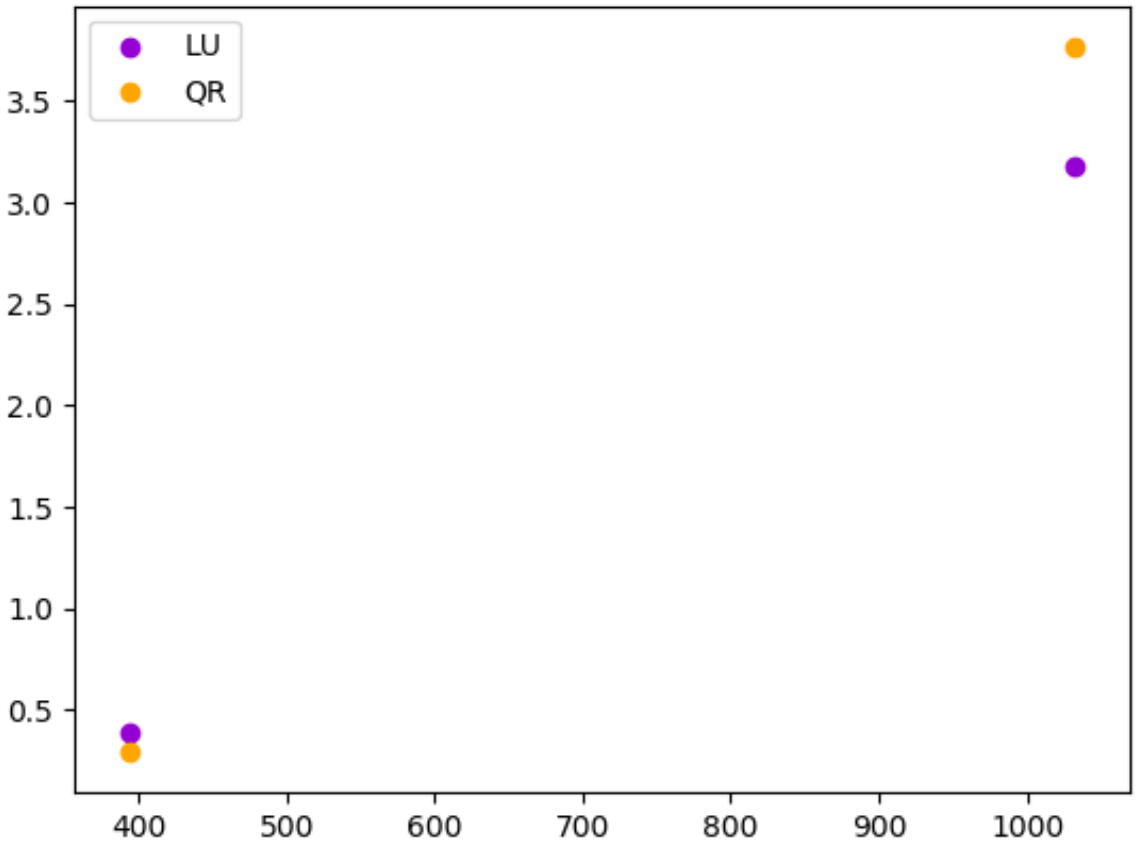

Figure 1: Complexité temporelle de résolution avec LU et QR

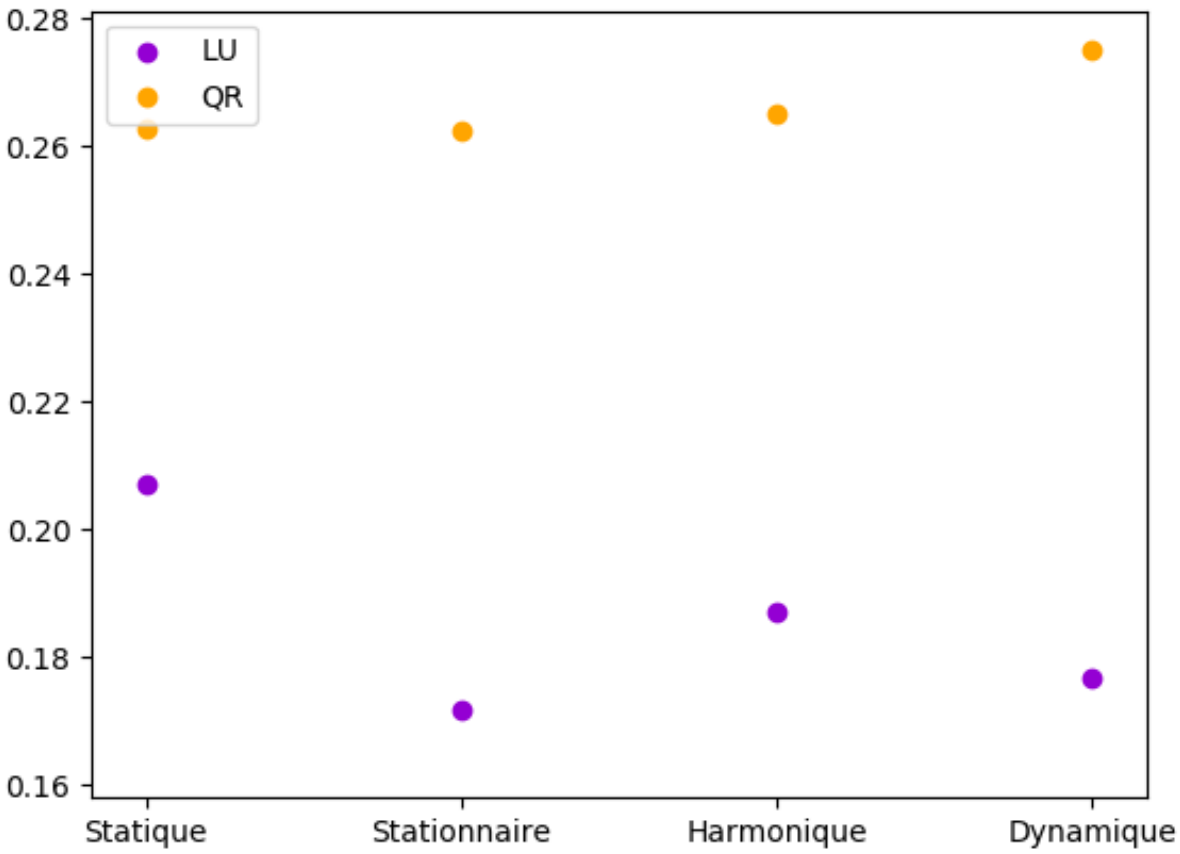

Figure 2: Complexité temporelle de résolution avec LU et QR sur les 4 régimes