LINMA1170 - Devoir 3

Giovanni Karra - 45032100

April 7, 2024

Questions théoriques

Question 1

Soit une matrice normale A ($A^*A = AA^*$), montrons que dans sa décomposition de Schur ($A = QTQ^*$, avec Q unitaire et T triangulaire supérieure), T doit être une matrice diagonale. Comme toute matrice carrée a une décomposition de Schur, nous pouvons écrire :

$$A = QTQ^* \Leftrightarrow Q^*AQ = T \tag{1}$$

$$TT^* = Q^*A\underbrace{QQ^*}_{I}A^*Q = Q^*AA^*Q \tag{2}$$

$$T^*T = Q^*A^*\underbrace{QQ^*}_{I}AQ = Q^*A^*AQ \tag{3}$$

Comme A est normale, nous savons que $TT^* = T^*T$, avec T triangulaire supérieure (U) et T^* triangulaire inférieure (L). Nous avons donc:

$$UL = LU$$

En partant de cette égalité, nous pouvons montrer itérativement que T est diagonale.

$$(UL)_{11} = \sum_{k=1}^{n} |t_{1k}|^2 = |t_{11}|^2 = (LU)_{11}$$

$$\Leftrightarrow \sum_{k=2}^{n} |t_{1k}|^2 = 0 \Leftrightarrow t_{1k} = 0 \quad \forall k \in \{2...n\}$$

$$(UL)_{22} = \sum_{k=2}^{n} |t_{2k}|^2 = |t_{22}|^2 + \underbrace{|t_{12}|^2}_{=0} = (LU)_{22}$$

$$\Leftrightarrow \sum_{k=3}^{n} |t_{2k}|^2 = 0 \Leftrightarrow t_{2k} = 0 \quad \forall k \in \{3...n\}$$

.

.

.

$$(UL)_{ii} = \sum_{k=i}^{n} |t_{ik}|^2 = \sum_{k=1}^{i} |t_{ki}|^2 = |t_{ii}|^2 + \underbrace{\sum_{k=1}^{i-1} |t_{ki}|^2}_{=0} = (LU)_{ii}$$

$$\Leftrightarrow \sum_{k=i+1}^{n} |t_{ik}|^2 = 0 \Leftrightarrow t_{ik} = 0 \quad \forall k \in \{i+1...n\}$$

Nous pouvons conclure que T est bel et bien diagonale.
Montrons maintenant que si T est diagonale dans une décomposition de Schur, alors la matrice d'origine A est normale. Nous commençons avec:

$$AA^* = QTT^*Q^* \tag{4}$$
$$A^*A = QT^*TQ^* \tag{5}$$

Comme T est diagonale, nous savons que $TT^* = T^*T$, car dans les deux cas nous avons $elem_{ij} = 0$ si $i \neq j$, et $elem_{ij} = |t_{ij}|^2$ sinon. Nous pouvons donc dire que $QTT^*Q^* = QT^*TQ^*$, et donc que $AA^* = A^*A$.

Question 2

Lorsque nous recevons une matrice hermitienne en entrée ($A^* = A$), nous pouvons grandement optimiser notre algorithme. En effet, lorsque nous opérons la transformation en matrice Hessenberg, nous pouvons nous contenter d'appliquer les transformations unitaires que d'un seul côté de la matrice, car les opérations qui seraient appliquées aux colonnes sont les mêmes que celle appliquées sur les lignes. Nous nous retrouvons finalement avec une matrice tridiagonale.
Ensuite, lors des itérations QR à proprement parler, Nous pouvons utiliser la tridiagonalalité de la matrice pour réduire le temps de calcul en exploitant la grande quantité de zéros.
Pour finir, en supposant que la matrice reste hermitienne, en plus de réduire le temps de calcul (on va de $\mathcal{O}(\frac{10}{3}m^3)$ avec une matrice générique à $\mathcal{O}(\frac{4}{3}m^3)$ avec une matrice hermitienne), nous pouvons également réduire la consommation de mémoire si nous stockons que la moitié de la matrice.

Question 3

Soit $H_k = \begin{bmatrix} a & b \\ \epsilon & d \end{bmatrix}$, avec ϵ petit. Appliquons une étape de l'algorithme QR avec et sans shifts, et analysons leurs convergences.
En appliquant la décomposition QR sur H_k, on obtient

$$Q = \frac{1}{\sqrt{a^2+\epsilon^2}} \begin{bmatrix} |a| & -sign(a)\epsilon \\ sign(a)\epsilon & |a| \end{bmatrix}$$
$$R = \frac{1}{\sqrt{a^2+\epsilon^2}} \begin{bmatrix} a|a| + sign(a)\epsilon^2 & b|a| + sign(a)b\epsilon \\ 0 & d|a| - sign(a)b\epsilon \end{bmatrix}$$

Obtenons le prochain itéré en multipliant Q et R à l'envers.

$$H_{k+1} = RQ = \underbrace{\frac{1}{a^2+\epsilon^2}}_{\approx a^2} \begin{bmatrix} a^3 + a\epsilon^2 + ab\epsilon + d\epsilon^2 & -a^2 - \epsilon^3 + a^2b + ad\epsilon \\ -\epsilon^2 b + ad\epsilon & -a\epsilon b + a^2 d \end{bmatrix}$$

Nous remarquons que l'entrée sous-diagonale converge en $\mathcal{O}(\epsilon)$.
Dans le cas de l'algorithme QR avec shift, nous commençons en soustrayant $I\mu$ à H_k, où le shift μ est le quotient de Rayleigh, c'est-à-dire d.

$$H_k - I\mu = \begin{bmatrix} a-d & b \\ \epsilon & 0 \end{bmatrix}$$

Nous appliquons encore une fois la décomposition QR.

$$Q = \frac{1}{\sqrt{(a-d)^2+\epsilon^2}} \begin{bmatrix} |a-d| & -sign(a-d)\epsilon \\ sign(a-d)\epsilon & |a-d| \end{bmatrix}$$
$$R = \frac{1}{\sqrt{(a-d)^2+\epsilon^2}} \begin{bmatrix} (a-d)|a-d| + sign(a-d)\epsilon^2 & b|a-d| + sign(a-d)b\epsilon \\ 0 & -sign(a-d)b\epsilon \end{bmatrix}$$

Ensuite nous effectuons l'itération.

$$H_{k+1} = RQ + I\mu =$$
$$\underbrace{\frac{1}{(a-d)^2+\epsilon^2}}_{\approx\ (a-d)^2} \begin{bmatrix} (a-d)^3 + (a-d)\epsilon^2 + (a-d)b\epsilon + d((a-d)^2+\epsilon^2) & -(a-d)^2 - \epsilon^3 + (a-d)^2 b \\ -\epsilon^2 b & -(a-d)\epsilon b + d((a-d)^2+\epsilon^2) \end{bmatrix}$$

Nous voyons clairement que la sous-diagonale converge en $\mathcal{O}(\epsilon^2)$.

Question 4

L'algorithme QR suppose que $|\lambda_1| > ... > |\lambda_n|$, et a une convergence de facteur $\left|\frac{\lambda_1}{\lambda_2}\right|$, nous rencontrons donc un problème lorsque deux plus grandes valeurs propres ont le même module[1]. En effet, dans ce cas les valeurs propres commencent à osciller indéfiniment, à moins d'utiliser l'algorithme avec shifts.
Par exemple, en executant l'algorithme sans shifts sur la matrice $\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}$ qui a comme valeurs propres $\sqrt{2}$, $-\sqrt{2}$ et 6, nous voyons qu'au bout de quelques itérations, les entrée correspondant à $\sqrt{2}$ et $-\sqrt{2}$ commencent à osciller entrer ± 1.17 et ± 1.52 indéfiniment, alors qu'avec shifts il suffit de 5 itérations pour converger.

Analyse de performances

Nous avons mesuré les performances des différentes implémentations de l'algorithme QR. Pour ce qui est de nos algorithmes faits maison, on considère qu'une entrée sous-diagonale est nulle lorsque son module est inférieur à 10^{-12}.
Nous savons que l'algorithme QR avec et sans shifts a une complexité temporelle en $\mathcal{O}(\frac{10}{3}m^3)$, avec m le nombre de lignes/colonnes de la matrice.
Cette complexité est la même que ce soit avec ou sans shifts car elle provient de l'algorithme de transformation en matrice Hessenberg, qui est utilisé de la même manière pour les deux versions. La différence vient donc de la vitesse de convergence des valeurs propres (linéaire sans shifts, et

[1] https://en.wikipedia.org/wiki/Power_iteration

quadratique, voire cubique, avec shifts): en effet nous remarquons que cette différence devient négligeable lorsque les deux algorithmes atteignent le plafond d'itérations permises, car dans ce cas la vitesse de convergence n'entre plus en compte.