LINMA1170 – Devoir 1 : Algorithme QR

Lucien Fiorini Henri Cornette

24. March 2025

1 Questions théoriques

1.1

Partons de la décomposition de Schur de A:

$$A = QTQ^* \tag{1.1}$$

Où Q est une matrice unitaire et T une matrice triangulaire supérieure.

En sachant que $A^*A = AA^*$, if faut que:

$$\begin{aligned} (QTQ^*)^*QTQ^* &= QTQ^*(QTQ^*)^* \\ QT^*Q^*QTQ^* &= QTQ^*QT^*Q^* \\ QT^*TQ^* &= QTT^*Q^* \\ T^*T &= TT^* \end{aligned} \tag{1.2}$$

$(T^*T)_{i,i} = (TT^*)_{i,i}$ peut être réécrit comme $\mathbf{e}_i^T(T^*T)\mathbf{e}_i = \mathbf{e}_i^T(TT^*)\mathbf{e}_i$, où $\mathbf{e}_i$ est un vecteur qui vaut 1 en i et 0 ailleurs.

$$\mathbf{e}_i^T(T^*T)\mathbf{e}_i = \mathbf{e}_i^T(TT^*)\mathbf{e}_i \Leftrightarrow (T\hat{\mathbf{e}}_i)^*(T\mathbf{e}_i) = (T^*\mathbf{e}_i)^*(T^*\mathbf{e}_i) \Leftrightarrow \|T\mathbf{e_i}\|^2 = \|T^*\mathbf{e}_i\|^2 \tag{1.3}$$

Ce qui montre que la $i^{\text{ème}}$ ligne doit avoir la même norme que la $i^{\text{ème}}$ colone. Intéressons nous à T:

$$T = \begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ 0 & \dots & x_{n,n} \end{pmatrix} \tag{1.4}$$

Nous remarquons que pour que la norme de la première colonne et celle de la première ligne soient égales il faut que tous les éléments de la première ligne soient nuls sauf $x_{1,1}$. On peut répéter cet argument jusqu'à montrer que T doit être diagonale. Ce qui prouve que A normale $\Rightarrow$ A diagonalisable par transformations unitaires. Il reste à prouver l'implication inverse:

$$A = QDQ^* \Rightarrow D^*D = DD^* \Leftrightarrow (QDQ^*)^*QDQ^* = QDQ^*(QDQ^*)^* \Leftrightarrow A^*A = AA^* \tag{1.5}$$

Où D est une matrice diagonale. □

Les vecteurs propres de A forment une base orthonormale car Q est unitaire.

1.2

Nous avons montré dans la question précédente que:

$$A^*A = QT^*TQ \tag{1.6}$$

Comme T test une matrice diagonale avec les valeurs propres de A:

$$\begin{aligned} T^*T &= \text{diag}(\bar{\lambda_1}\lambda_1, ..., \bar{\lambda_i}\lambda_i) \\ &= \text{diag}(|\lambda_1|^2, ..., |\lambda_i|^2) \end{aligned} \tag{1.7}$$

Par définition, σ_i est la racine carré de la $i^{\text{ème}}$ valeur propre de A^*A et nous savons que ces valeurs propres se trouvent sur la diagonale de T^*T.

$$\sigma_i = \sqrt{|\lambda_i|^2} = |\lambda_i| \tag{1.8}$$

1.3

Soit une matrice de Hessenberg à l'itération k : $H_k = \begin{pmatrix} a & b \\ \varepsilon & d \end{pmatrix}$ où ε est supposé petit.

1.3.1 Sans shift :

Dans l'algorithme QR sans shift, on décompose H_k en QR puis on calcule $H_{k+1} = \text{RQ}$.

1) Décomposition QR de H_k : Pour trouver Q, nous utilisons une rotation de Givens pour annuler l'élément (2,1).

Soit $c = \frac{a}{\sqrt{a^2+\varepsilon^2}}$ et $s = \frac{\varepsilon}{\sqrt{a^2+\varepsilon^2}}$

Alors $Q = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ et $Q^T H_k = \begin{pmatrix} \sqrt{a^2+\varepsilon^2} & cb+sd \\ 0 & -sb+cd \end{pmatrix} = R$

2) Calcul de $H_{k+1} = RQ$:

$$\begin{aligned} H_{k+1} &= \begin{pmatrix} \sqrt{a^2+\varepsilon^2} & cb+sd \\ 0 & -sb+cd \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \\ H_{k+1} &= \begin{pmatrix} c\sqrt{a^2+\varepsilon^2} - s(cb+sd) & s\sqrt{a^2+\varepsilon^2} + c(cb+sd) \\ -s(-sb+cd) & c(-sb+cd) \end{pmatrix} \end{aligned} \tag{1.9}$$

3) L'élément sous-diagonal de H_{k+1} est : $-s(-sb+cd) = s^2b - scd = \frac{\varepsilon^2 b - \varepsilon ad}{a^2+\varepsilon^2}$

Comme $\varepsilon^2 \approx 0$, on peut retirer sa contribution et l'élément sous-diagonal est approximativement : $\frac{\varepsilon^2 b}{a^2} - \frac{\varepsilon d}{a}$

Le terme dominant est $\frac{\varepsilon d}{a}$, qui est d'ordre $O(\varepsilon)$.

1.3.2 Avec shift de Rayleigh-quotient :

Avec le shift $\mu = d$ (dernier élément diagonal), nous travaillons avec la matrice $H_k - \mu I$:

$$H_k - \mu I = \begin{pmatrix} a-d & b \\ \varepsilon & 0 \end{pmatrix} \tag{1.10}$$

on applique la même procédure que pour la matrice non shifté et nous tombons sur la sous diagonale de H_{k+1} qui vaut x = $\frac{-b\varepsilon^2}{(a-b)+\varepsilon^2}$. De nouveau, $\varepsilon^2 \approx 0$ et donc sa contribution au dénumérateur est nulle et donc x = $\frac{-b\varepsilon^2}{(a-b)}$ ce qui indique une convergence quadratique $O(\varepsilon^2)$

2 Analyses numériques

2.1 Laplacien 2D

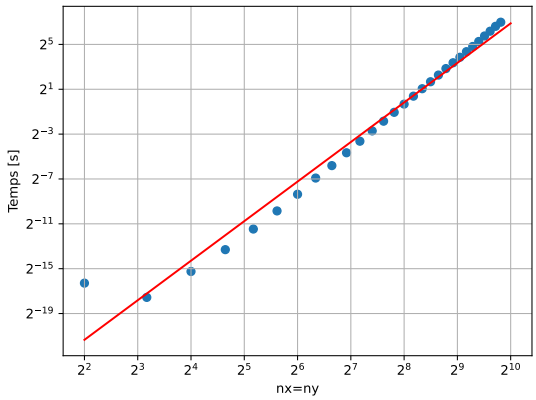

Abb. 1: Graphe log-log de la complexité temporelle sur le Laplacien en 2D

En effectuant une régression linéaire sur les données nous arrivons à une pente de 3.528 ce qui voudrait dire que notre algorithme est $O(n^{3.528})$ ce qui se rapproche de la complexité théorique qui est $O(n^3)$.

2.2 Laplacien 1D

Commençons par discrétiser le Laplacien en 1D par différences finies: $\frac{u(x_{i+1})-2u(x_i)+u(x_{i-1})}{h^2}$, ce qui se traduit en cette matrice:

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & 1 & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix} \tag{2.1}$$

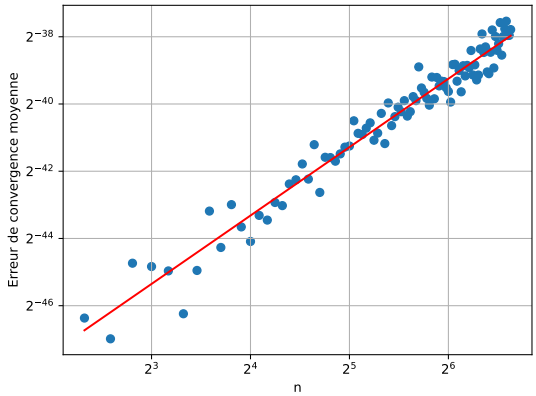

Abb. 2: Erreur de convergence moyenne des valeurs propres du spectre du Laplacien 1D en log-log

Nous avons calculé la moyenne de la différence des valeurs propres calculées avec notre programme C par rapport à celles calculées par `sympy` pour n allant de 5 à 100.

Nous avons calculé la moyenne de la différence entre les valeurs propres obtenues par notre programme en C et celles calculées avec `sympy`, pour des valeurs de n allant de 5 à 100.

En utilisant une régression linéaire et en regardant la pente, nous observons que l'erreur croit en $O(n^{2.0343}) \approx O(n^2)$.