1 Lezione del 23-05-25

Continuiamo a vedere i metodi per più autovalori.

1.0.1 Metodo QR per gli autovalori

Introduciamo quindi il **metodo QR per gli autovalori**, che non va confuso col metodo QR *per i minimi quadrati* (visto in 12.2), anche se chiaramente si basa sulla fattorizzazione QR.

Questo è quindi un metodo per trovare tutti gli autovalori di $A \in \mathbb{C}^{n\times n}$ e un insieme di autovettori associati a ciascun autovalore. Abbiamo quindi come input A, de come uscita V e D, con D diagonale di autovalori:

$$D = \begin{pmatrix} \lambda_1 & \dots & 0 \\ & \ddots & \\ 0 & \dots & \lambda_n \end{pmatrix}$$

e V tale che:

$$A = VDV^{-1}$$

L'algoritmo che risolve questo problema viene ideato negli anni '70, ed è considerato fra i più importanti degli ultimi anni. Vedremo una versione *"vanilla"*, quindi semplificata, dell'algoritmo, in quanto le implementazioni moderne (MATLAB o librerie come *LAPACK*)non sarebbero effettivamente utili se non per le ottimizzazioni che sono state fatte nel tempo.

L'idea di fondo è creare una successione di matrici:

$$\{A_k\}_{k\in\mathbb{N}}, \quad A_0 = A$$

tutte simili ad A, cioè tali che:

$$\forall k,\ \exists Q\ :\ Q^{-1}A_kQ = A$$

con Q invertibile ed unitaria, cioè:

$$Q^{-1} = Q^H$$

In questo modo ogni A_k ha gli stessi autovalori di A, mentre gli autovettori di $A_K = Q^HAQ$ saranno Q^hv, dove v è autovettore di A. Inoltre, chiamando:

$$A_\infty = \lim_{k\to+\infty} A_k$$

si ha che A_∞ è unna matrice di cui si possono calcolare gli autovettori e autovalori in modo efficiente. Nello specifico, A_∞ è una matrice triangolare.

1.0.2 Calcolo di autovalori e autovettori per matrici triangolari

Motiviamo quindi quest'ultima affermazione studiando il caso particolare delle matrici triangolari. Poniamo T triangolare superiore:

$$T = \begin{pmatrix} t_{11} & \dots & t_{1n} \\ & \ddots & \vdots \\ 0 & & t_{nn} \end{pmatrix}$$

gli autovalori sono facili, in quanto risulteranno:

$$\{t_{11}, t_{22}, ..., t_{nn}\}$$

gli autovettori sono più difficili, ma vediamo che il primo è banale:

$$Tv_1 = \begin{pmatrix} t_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = t_{11} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \implies v_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Cerchiamo quini il generico autovettore v_j assumedo $t_{ii} \neq t_{jj}$, $\forall i \neq j$ (cioè l'ipotesi degli autovalori distinti) e cerchiamo una soluzione diversa da 0 del sistema:

$$(T - t_{jj}I)v_j = 0$$

che avrà l'aspetto:

$$\begin{pmatrix} t_{11} - t_{jj} & t_{12} & & ... & & & & t_{1n} \\ & \ddots & & ... & & & & \vdots \\ & & t_{j-1,j-1} - t_{jj} & ... & & & & \vdots \\ & & & 0 & & ... & & \vdots \\ & & & & t_{j+1,j+1} - t_{jj} & ... & & \vdots \\ & & & & & & \ddots & \vdots \\ & & & & & & & t_{nn} - t_{jj} \end{pmatrix} v_j = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix}$$

Riscrivendo a blocchi abbiamo:

$$\begin{pmatrix} (T_1) & z & (*) \\ & 0 & (*) \\ & & (T_2) \end{pmatrix} \underbrace{\begin{pmatrix} (v_j^{(1)}) \\ j \\ v_j^{(2)} \end{pmatrix}}_{v_j} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \Leftrightarrow \begin{cases} T_1 v_j^{(1)} + \gamma z + (*) v_j^{(2)} = 0 \\ ... \\ T_2 v_j^{(2)} = 0 \end{cases}$$

Notiamo che:

$$T_2 v_j^{(2)} = 0$$

è un sistema $(n - j - 1) \times (n - j - 1)$ ha matrice invertibile, per cui:

$$v_j^{(2)} = 0$$

e la parte indicata con asterischi del sistema non ci interessa, per cui possiamo dire:

$$T_1 v_j^{(1)} + \gamma z = 0$$

che è un sistema $(j - 1) \times j$, cioè sottodeterminato. Scegliamo ad esempio $\gamma = 1$ per fissare una soluzione, e otteniamo:

$$T_1 v_j^{(1)} = -z \implies v_j^{(1)} = -T_1^{-1} z$$

Per cui abbiamo ricostruito l'intero vettore v_j come:

$$v_j = \begin{pmatrix} -T_1^{-1}z \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Iteriamo quindi per gli $j = 1, ..., n$ indici e troviamo tutti gli autovettori.

Dal punto di vista del costo, per trovare v_j si deve risolvere il sistema $T_1 x = -z$ di dimensione $(j-1) \times (j-1)$ e triangolare, che costa $O(j^2)$. Col fatto che iteriamo per j volte abbiamo complessivamente:

$$O\left(\sum_{j=1}^{n} j^2\right) = O(n^3)$$

1.0.3 Metodo QR per la successione

Vediamo quindi come formare la successione vera e propria. L'idea è che prendiamo $A_0 = A$, e per $k = 1, 2, ...$ prendiamo:

$$A_{k-1} = Q_{k-1}R_{k-1}$$

$$A_k = R_{k-1}Q_{k-1}$$

In effetti le matrici A_k sono simili:

$$A_{k+1} = R_k Q_k = Q_k^H Q_k R_k Q_k = Q_k^H A_k Q_k$$

espandendo R_k, e quindi A_{k+1} è simile ad A_k. Iterando il ragionamento si ottiene che A_{k+1} è simile ad A_0 e quindi ad A.

In particolare:

$$A_{k+1} = Q_k^H A_k Q_k = Q_k^H Q_{k-1}^H A_{k-1} Q_{k-1} Q_k = ... = \underbrace{Q_k^H Q_{k-1}^H ... Q_1^H}_{\tilde{Q}_k^H} A \underbrace{Q_1 ... Q_{k-1} Q_k}_{\tilde{Q}_k}$$

e quindi:

$$A_{k+1} = \tilde{Q}_k^H A \tilde{Q}_k$$

da cui immediatamente la similarità con $\tilde{Q}_k$ matrice di traduzinoe.

Potremo quindi scrivere l'implementazione:

```
A1 = A;
for k = 1:n
  Ak = Qk Rk;
  Ak+1 = R_k Q_k;
end

A_n = triu(A_k) % mantengo solo la parte tiriangolare superiore

[V, D] = eig(A_k)
V = Q_tildek V % insomma correggi V
```

A giustificare questo abbiamo il teorema:

Teorema 1.1: Validità del metodo QR agli autovalori

Presa $A \in \mathbb{R}^{n \times n}$ e supposto:

$$|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$$

allora:

$$\lim_{n \to +\infty} A_k = T = \begin{pmatrix} t_{11} & ... & t_{1n} \\ & \ddots & \vdots \\ 0 & & t_{nn} \end{pmatrix}$$

dove i blocchi T_{jj} sono 1×1 per autovalori reali e 2×2 per autovalori complessi coniugati.

Vediamo che i termini sotto a cui abbiamo definito l'algoritmo sono estremamente restrittivi (autovalori distinti, ecc...). Chiaramente nelle implementazioni reali ci sono, oltre che alle ottimizzazioni, vari accorgimenti che rendono l'algoritmo robusto alle varie casistiche (autovalori ripetuti, ecc...), mantenendo il costo cubico $O(n^3)$.

Alcuni accorgimenti sono:

1. Si deve tenere il numero di iterazioni N sotto controllo, e in particolare viene garantito che:
$$N = O(n)$$

2. Calcolare la fattorizzazione QR costa $O(n^3)$, se la calcoliamo ad ogni passo otteniamo un costo $O(n^4)$ che non è ottimale. Nella versione finale si fa un preprocessing sulla matrice A che la riduce alla cosiddetta *forma di Hessenberg* che rende il costo del calcolo della fattorizzazione quadratico $O(n^2)$ anziché cubico, così che l'algoritmo complessivo costi $O(n^3)$;

3. Nella soluzione che abbiamo visto si fanno assunzioni sugli autovalori (autovettori distinti, moduli distinti, ecc...). Queste possono essere aggirate, e il metodo QR completo non ha restrizioni sulla matrice A.