Equazioni matriciali con struttura Hessenberg-Triangolare

Alessandra Cattafi

Progetto di Calcolo Scientifico, Anno Accademico 2021/22

Sommario

Questo progetto tratta la risoluzione efficiente di equazioni che coinvolgono matrici di Hessenberg. Ciò getta le basi per un metodo per equazioni più generali: il metodo Hessenberg-Schur.

1 Introduzione

Si consideri la seguente equazione:

$$HX + XT = C \tag{1}$$

dove $X \in \mathbb{C}^{m \cdot n}$ è l'incognita, $C \in \mathbb{C}^{m \cdot n}$ il termine noto, $H \in \mathbb{C}^{m \cdot m}$ Hessenberg superiore e $T \in \mathbb{C}^{n \cdot n}$ triangolare superiore. Si vuole risolvere questa equazione determinando una colonna di X alla volta, partendo dalla prima. In particolare, definiamo:

$$X = \begin{bmatrix} & & & \\ x_1 & x_2 & ... & x_n \\ & & & \end{bmatrix}$$

.

2 Il sistema lineare

Si osservi che, moltiplicando l'equazione (1) a destra per e_j:

$$Hx_j + XTe_j = Ce_j \tag{2}$$

Siano t_{ij} gli elementi della matrice T, in particolare, siano t_{jj} i suoi elementi diagonali. Possiamo riscrivere (2) come:

$$Hx_j + \sum_{i \leq j} t_{ij} x_i = Ce_j$$

Ponendo $Ce_j - \sum_{i<j} t_{ij}x_i = b_j$

$$(H - t_{jj}I)x_j = b_j \tag{3}$$

Si noti che b_j dipende solamente da T, C e $x_1, ..., x_{j-1}$.

3 Risoluzione del sistema

È possibile risolvere il sistema lineare presentato in (3) tramite il metodo diretto che sfrutta la fattorizzazione QR (in cui Q è una matrice ortogonale, ed R è triangolare superiore).

$$\begin{cases} Q'y = b_j \\ R'x_j = y \end{cases}$$

Il costo della risoluzione di questo sistema è $O(n^2)$:il calcolo di Q'^T è immediato, richiedendo per $Q'y = b_j$ $O(n^2)$ operazioni; mentre essendo $R'x = y$ con matrice triangolare superiore si risolve per sostituzione all'indietro con $\frac{n^2}{2}$ operazioni aritmetiche.

Si noti che la matrice $H + t_{jj}I = H'$ condivide la struttura upper-Hessenberg di H. La forma particolare di H' rende preferibile ed efficiente fattorizzarla tramite trasformazioni di Givens. Una matrice di Givens G_{ij} di dimensione m rappresenta una rotazione in $\mathbb{R}^m$ ed è strutturata come segue:

$$G_{ij} = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & c & \dots & s & & \\ & & \vdots & \ddots & \vdots & & \\ & & -s & \dots & c & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix}$$

dove $G_{ij} = (g_{ij})_{i \in I, j \in J}$ coincide con la matrice identità tranne che nelle seguenti coordinate:

$$\begin{cases} g_{ii} = c \\ g_{jj} = c \\ g_{ij} = s \\ g_{ji} = -s \end{cases}$$

con c, s, tali che $c^2 + s^2 = 1$.È possibile, dato un vettore in $\mathbb{R}^m$ costruire $m - 1$ matrici di Givens che, moltiplicate a sinistra per il vettore, forniscono il modo per parallelizzare tale vettore rispetto al vettore canonico e_1. È evidente quindi la possibilità di ottenere una fattorizzazione QR sfruttando le matrici elementari di Givens.

Sotto queste ipotesi, il costo del calcolo di R' e Q' è di $O(m^2)$: $O(m)$ rotazioni di Givens, con costo unitario di $O(m)$.

$$H' = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = Q'R' = \begin{bmatrix} \sqsubset & & & & \\ & \sqsubset & & & \\ & & \sqsubset & & \\ & & & \sqsubset & \end{bmatrix} \begin{bmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

Dunque, il costo totale della risoluzione di (3) è $O(m^2) + O(n^2)$,dunque, essendo $m > n$,$O(m^2)$. Ciò rende il costo totale per il calcolo di ogni colonna di X $n \cdot O(m^2 + n^2) = O(nm^2 + n^3)$

4 Implementazione

Per l'implementazione del problema richiesto ho deciso di fornirmi di due funzioni ausiliarie che mi permettono, ripettivamente, di ottenere la fattorizzazione QR di una matrice upper-hessenberg e di risolvere un sistema lineare con matrice triangolare superiore. Seguono i tre algoritmi da me implementati:

4.1 Fattorizzazione QR di una matrice upper-hessenberg tramite rotazioni di Givens

In quest'implementazione mi sono servita del comando givens di matlab per la costruzione delle matrici di rotazione. Il comando givens restituisce la matrice di rotazione di Givens complessa $2 \cdot 2$ seguente:

$$G = \begin{bmatrix} c & s \\ -conj(s) & c \end{bmatrix}$$

tale che

$$G * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

dove $c \in \mathbb{R}$, $s \in \mathbb{C}$ e $c^2 + |s|^2 = 1$.

```
function [R,y]= solve_givens(H,b)
%Applica i primi due passaggi del metodo diretto QR
   sulla matrice
%upper-hessenberg H (operando tramite rotazioni di
   Givens) e restituisce la
%matrice triangolare superiore R, e il vettore dei
   termini noti y=Q'b
k=size(H);
for i=1:k(2)-1
        G=givens(H(i,i), H(i+1,i));
        H(i:i+1, :)=G*H(i:i+1,:);
        b(i:i+1)= G*b(i:i+1);
end
```

```
R=H;
y=b;
end
```

4.2 Risoluzione di un sistema con matrice triangolare superiore con sostituzione all'indietro

```
function x= upper_eq(T,b)
%risolve un sistema lineare con matrice triangolare
   superiore tramite il
%metodo di sostituzione all'indietro.
[mt,nt]=size(T);
if(mt~=nt)
    error('la matrice T non quadrata')
end
nb=length(b);
if(nb~=nt)
    error('le dimensioni di T e b noti sono
       incompatibili')
end
x=b(:);
if T(nb,nb)==0
    error('impossibile risolvere il sistema T
       singolare')
end
x(nb)=x(nb)/T(nb,nb);
for i=nb-1:-1:1
    if T(i,i)==0
        error('impossibile risolvere il sistema, T
           singolare')
    end
    x(i)=(x(i)-T(i, i+1:nb)*x(i+1:nb))/T(i,i);
end
end
```

4.3 Risoluzione dell'equazione matriciale HX + XT = C

Questo algoritmo implementa il metodo discusso nella sezione precedente, servendosi delle due funzioni ausiliarie discusse in precedenza.

```
function X= hess_eq(H,T,C)
% risolve l'equazione HX + XT = C (H upper-hessenberg,
    T triangolare superiore),
% determinando le colonne x_j di X in successione e
   con un costo totale di
%O(m^2n + n^3).
m=size(C,1);
n=size(C,2);
X=zeros(m,n); %inizializzo la matrice a nulla
for j=1:n
    if j~=1
        sum=zeros(m,1);
        for i=1:j-1
            sum=sum + T(i,j)*X(:,i);
        end
        b=C(:,j)-sum;
    else
        b=C(:,j);
    end
    [R,y]=solve_givens(H+T(j,j)*eye(m),b);
    X(:,j)=upper_eq(R,y);
end
end
```

5 Sperimentazione

In questa sezione presenterò i risultati da me ottenuti nel testare l'algoritmo implementato: i test sono stati svolti nella versione di Matlab disponibile in Matlab online, che ha una potenza di calcolo maggiore rispetto al mio portatile. Le principiali caratteristiche da me testate sono la complessità e la correttezza dell'algoritmo.

5.1 Complessità: elapsed time

La tabella seguente compara l'elapsed time in secondi dell'algoritmo precedente al variare della taglia di H, calcolato utilizzando il comando tic toc di Matlab. Le matrici di test sono state generate in maniera casuale usando i comandi seguenti: $H = hess(randn(m)); T = schur(randn(n), 'complex'); X = randn(m, n); C = HX + XT$. A causa della complessità $O(nm^2 + n^3)$ ho deciso di mantenere la taglia delle matrici ragionevolmente bassa, a causa dei limiti della macchina su cui sto lavorando.

Taglia H	Taglia T	Elapsed time
50	50	0.028094
100	50	0.071168
200	50	0.180312
400	50	0.446838
800	50	1.692726
1600	50	6.727691
3200	50	25.986186
6400	50	116.298403
12800	50	764.831243

Tabella 1: Questa tabella racchiude e confronta l'elapsed time in secondi dell'algoritmo da me implementato, al variare dell'ordine di grandezza delle taglia della matrice H.

I seguenti grafici plottano l'elapsed time rispettivamente in funzione della taglia di H e di T, riconfermando anche graficamente la dipendenza temporale riscontrata teoricamente.

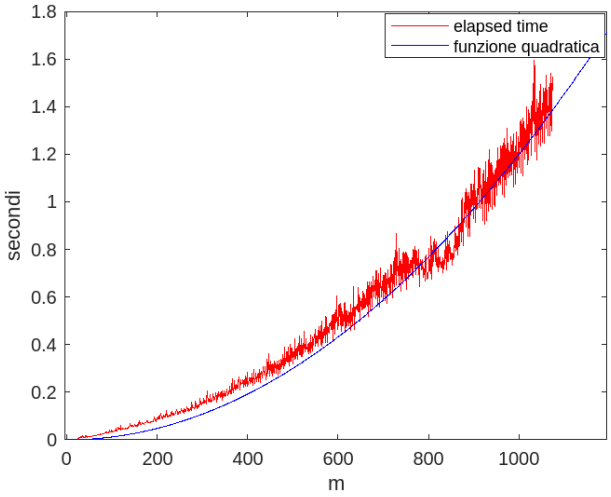

Figura 1: Elapsed time in funzione della taglia di H, con taglia di T fissa pari a 25

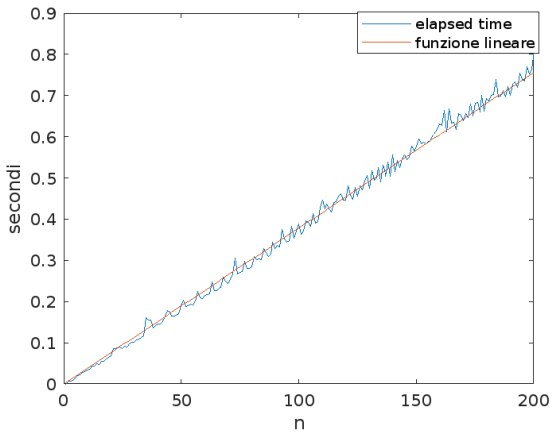

Figura 2: Elapsed time in funzione della taglia di T, con taglia di H fissa pari a 200

Si osservi che, anche in pratica, l'algoritmo ha una complessità in tempo conforme a quanto osservato e calcolato nelle sezioni precedenti.

5.2 Correttezza: residuo ed errore relativo

Ho ritenuto interessante misurare la correttezza del mio algoritmo al variare della taglia delle matrici H e T. I risultati sulle seguenti coppie di taglie sono stati ottenuti nel modo seguente:

```
media=0;
mediainf=0;
med=0;
medinf=0;
for t=1:10
H=hess(randn(m));
T=schur(randn(n), 'complex');
X=randn(m,n); %soluzione vera
C=H*X+X*T;
Y=hess_eq(H,T,C); %soluzione garantita dall'algoritmo
media=media + norm(Y-X)/norm(X); %errore relativo
    norma 2
mediainf= mediainf + norm(Y-X, Inf)/norm(X); %errore
    relativo norma infinito
med= med + norm(H*Y + Y*T -C)/norm(Y); %residuo norma
    2
medinf= medinf + norm(H*Y + Y*T -C, Inf)/norm(Y); %
    residuo norma infinito
end
media=media/10;
mediainf=mediainf/10;
med=med/10;
medinf=medinf/10;
```

Taglia H	Taglia T	Residuo norma 2	Residuo norma ∞
25	25	1.7053e-15	5.1335e-15
50	25	2.8941e-15	7.9818e-15
75	25	3.6194e-15	9.3204e-15
100	50	4.8541e-15	1.7525e-14
150	50	6.7116e-15	2.2894e-14
200	50	8.6922e-15	2.7755e-14
400	100	1.2449e-14	6.8319e-14
800	100	1.8412e-14	1.0547e-13
1600	100	3.2462e-14	1.6057e-13

Tabella 2: Questa tabella racchiude e confronta il residuo sull'equazione da risolvere al variare delle taglie di H e T.

Si noti che il residuo risulta soddisfacentemente piccolo.

Taglia H	Taglia T	Errore relativo norma 2	Errore relativo norma ∞
25	25	5.3290e-14	3.7827e-14
50	25	1.2204e-13	8.4237e-14
75	25	3.2930e-13	1.8338e-13
100	50	2.2880e-13	2.3057e-13
150	50	3.2916e-13	3.0982e-13
200	50	1.4967e-13	1.4229e-13
400	100	8.1100e-13	5.6888e-13
800	100	5.5078e-13	3.7174e-13
1600	100	7.7525e-13	3.5025e-13

Tabella 3: Questa tabella racchiude e confronta l'errore relativo accumulato dal mio algoritmo al variare delle taglie di H e T.

Al netto dei dati appena esposti, l'algoritmo risulta sufficientemente stabile all'indietro.