Laboratorio di Calcolo Numerico
Lezione 5

1 Sistemi con più termini noti

Si può utilizzare l'eliminazione di Gauss anche per risolvere un'equazione della forma

$$AX = B, \quad B \in \mathbb{C}^{n\times s}, \quad X \in \mathbb{C}^{n\times s},$$

che è equivalente al risolvere s sistemi lineari

$$Ax^{(1)} = b^{(1)}, \quad Ax^{(2)} = b^{(2)}, \quad \dots \quad Ax^{(s)} = b^{(s)},$$

in cui $x^{(j)}, b^{(j)}$ rappresentano le colonne di X e B rispettivamente. Quello che si deve fare per applicare l'eliminazione di Gauss in questo contesto è applicare la fase di triangolarizzazione alla matrice aumentata $(A \mid B)$ ed infine risolvere gli s sistemi triangolari trovati (che condividono la stessa matrice dei coefficienti).

Ad esempio per

$$A = \begin{bmatrix} -2 & 1 & 1 \\ -1 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} -2 & 0 \\ -3 & 0 \\ 2 & 1 \end{bmatrix}$$

si ha nella prima fase

$$\left[\begin{array}{ccc|cc} -2 & 1 & 1 & -2 & 0 \\ -1 & -3 & 1 & -3 & 0 \\ 1 & 1 & 2 & 2 & 1 \end{array}\right] \to \left[\begin{array}{ccc|cc} -2 & 1 & 1 & -2 & 0 \\ 0 & \frac{5}{2} & \frac{1}{2} & -2 & 0 \\ 0 & \frac{3}{2} & \frac{5}{2} & 1 & 1 \end{array}\right] \to \left[\begin{array}{ccc|cc} -2 & 1 & 1 & -2 & 0 \\ 0 & \frac{5}{2} & \frac{1}{2} & -2 & 0 \\ 0 & 0 & \frac{11}{5} & \frac{11}{5} & 1 \end{array}\right]$$

ed infine la risoluzione di

$$\begin{bmatrix} -2 & 1 & 1 \\ 0 & \frac{5}{2} & \frac{1}{2} \\ 0 & 0 & \frac{11}{5} \end{bmatrix} x^{(1)} = \begin{bmatrix} -2 \\ -2 \\ \frac{11}{5} \end{bmatrix} \quad \Rightarrow \quad x^{(1)} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} -2 & 1 & 1 \\ 0 & \frac{5}{2} & \frac{1}{2} \\ 0 & 0 & \frac{11}{5} \end{bmatrix} x^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \Rightarrow \quad x^{(2)} = \begin{bmatrix} \frac{2}{11} \\ -\frac{1}{11} \\ \frac{5}{11} \end{bmatrix}$$

che ci dice che la soluzione è

$$X = \begin{bmatrix} 1 & \frac{2}{11} \\ -1 & -\frac{1}{11} \\ 1 & \frac{5}{11} \end{bmatrix}.$$

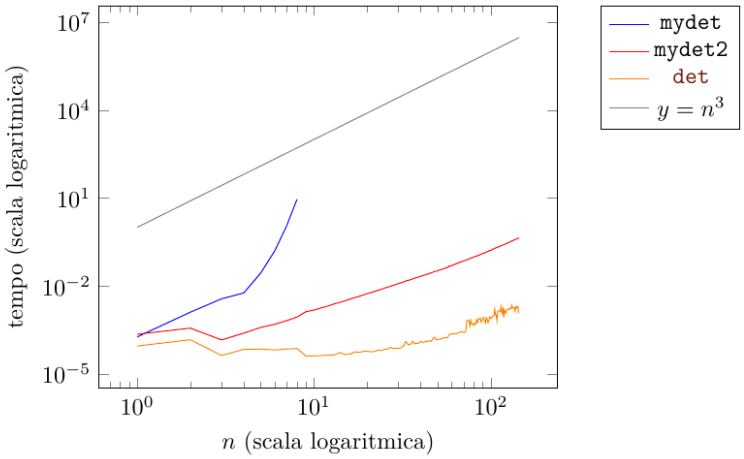
Esercizio 1. Modificare la funzione `sys_solve` implementata nella lezione 4, in modo che gestica il caso di un termine noto B con più di una colonna, risolvendo l'equazione $AX = B$ (quindi ritornando una soluzione con più colonne) tramite l'eliminazione di Gauss generalizzata descritta sopra. **Suggerimento:** per risolvere i sistemi triangolari alla fine si può eseguire un for loop in cui viene chiamata la funzione `sup_solve` implementata nella lezione 4.

2 Determinante con l'eliminazione di Gauss

Esercizio 2. Scrivete una `function d=mydet2(A)` che calcoli il determinante utilizzando l'eliminazione di Gauss. Suggerimenti:

- utilizzate espressioni del tipo `A(i,:)=A(i,:)+A(j,:)` per sommare due righe di una matrice, invece di scrivere un ciclo `for`.
- per ora, ignorate il fatto che i pivot possono essere zero; se incontrate un pivot nullo, terminate con un messaggio di errore (potete farlo con l'istruzione `error('ho incontrato un pivot nullo')`).
- per controllare che tutto vada bene, in una prima fase fatevi scrivere a schermo il valore di `A` dopo ogni passo di eliminazione di Gauss

Esercizio 3. Come nell'esercizio 4 della lezione 2, disegnare un grafico in cui si riportano i tempi di calcolo di `mydet2` per matrici $n \times n$ (generate in maniera casuale) per n che cresce da 1 a 150; si utilizzi la scala logaritmica per entrambi gli assi (usare il comando `loglog` invece di `plot`), in modo da poter confrontare meglio tempi su scale molto diverse. Si aggiungano i tempi di calcolo della funzione `mydet` (implementata nella lezione 2 e per cui ci si può fermare per $n = 10$) e di `det` che è la funzione di Matlab per il calcolo del determinate. Il grafico dovrebbe assomigliare a questo:



Per riferimento abbiamo riportato anche il grafico di $y = n^3$, sempre sulla stessa scala logaritmica. Qual è la pendenza della retta che corrisponde ad esso? Come potete determinare dal grafico che `mydet2` richiede $O(n^3)$ operazioni mentre `mydet` è asintoticamente più costoso? La funzione `det` di MATLAB dovrebbe essere anch'essa cubica, ma il suo comportamento è più complicato da determinare per via delle ottimizzazioni usate; i tempi esatti dipendono fortemente dall'architettura del calcolatore.

3 Metodi di Jacobi e Gauss–Seidel

3.1 Matrici di test

Inserire in MATLAB le seguenti matrici.

$$A1 = \begin{bmatrix} 3 & 0 & 4 \\ 7 & 4 & 5 \\ -1 & -2 & 2 \end{bmatrix}, \qquad A2 = \begin{bmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{bmatrix},$$
$$A3 = \begin{bmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & -6 \end{bmatrix}, \qquad A4 = \begin{bmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{bmatrix}.$$

Vogliamo verificare che

- Su $A1$, il metodo di Jacobi non converge ma quello di Gauss–Seidel sì;
- Su $A2$, Jacobi converge (piano) ma Gauss–Seidel no;
- Su $A3$, convergono entrambi e Gauss–Seidel è più veloce;
- Su $A4$, convergono entrambi (piano) e Jacobi è più veloce.

3.2 Il metodo di Jacobi

La formula che definisce il metodo di Jacobi per la risoluzione del sistema $Ax = b$ è

$$x_i^{(new)} = \frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(old)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(old)} \right), \qquad i = 1, \ldots, n.$$

Esercizio 4. Scrivere una funzione `function x=jacobi(A,b,k)` che esegue `k` passi del metodo di Jacobi. Dopo ogni passo, scrivere sullo schermo la norma-2 del *residuo* con l'istruzione `norm(A*x-b)` (senza punto e virgola finale).

Suggerimento: attenzione che non potete "riutilizzare" la variabile `x` in un ciclo del tipo

```
for i=1:n
...
x(i) = (formule che coinvolgono elementi di x);
...
end
```

perché in questo modo dopo la prima iterazione il vecchio valore di `x(1)` va perso, ma a voi serve ancora nelle formule che calcolano i nuovi `x(2)`,`x(3)`,... Dovrete invece fare qualcosa del tipo

```
x_new=zeros(n,1);
for i=1:n
...
x_new(i) = (formule che coinvolgono elementi di x);
...
end
x=x_new;
```

Testare la funzione sulle quattro matrici di test $A1, \ldots, A4$, con termine noto b a piacere (per esempio il vettore con tutti gli elementi uguali a 1) e vettore iniziale $x^{(0)} = b$: dovrebbe venire qualcosa simile all'output qui sotto. I numeri precisi calcolati da voi possono cambiare (dipende da un paio di scelte che potete fare nell'algoritmo), però il comportamento della successione (va a zero? Quanto velocemente?) dovrebbe corrispondere.

```
>> jacobi(A1,ones(3,1),10)
ans = 13.6839
ans = 38.2823
ans = 57.3885
ans = 68.4757
ans = 189.6514
ans = 136.0946
ans = 488.7049
ans = 571.4353
ans = 977.2891
ans = 2.1035e+03

ans =

149.5063
218.0709
9.7088

>> jacobi(A2,ones(3,1),10)
ans = 17.088015
ans = 26.818686
ans = 53.445524
ans = 93.296349
ans = 177.191043
ans = 320.662796
ans = 597.366966
ans = 1094.297461
```

```
ans = 2024.546226
ans = 3724.075589

ans =

574.9793
387.4866
337.3805

>> jacobi(A3,ones(3,1),10)
ans = 7.1111
ans = 2.2222
ans = 1.5062
ans = 0.40329
ans = 0.24829
ans = 0.13397
ans = 0.029064
ans = 0.025834
ans = 0.010817
ans = 0.0028290

ans =

0.287301
-0.046952
-0.104023

>> jacobi(A4,ones(3,1),10)
ans = 23.400
ans = 12.514
ans = 4.3015
ans = 3.6876
ans = 2.8242
ans = 1.0743
ans = 0.75329
ans = 0.68841
ans = 0.29360
ans = 0.25730

ans =

-0.261799
0.430087
0.056597
```

I risultati non sono molto significativi perché 10 iterazioni sono molto poche, provare per esempio con $k = 50$. Quando c'è convergenza? Quante iterazioni servono perché il residuo scenda sotto la soglia $\varepsilon = 10^{-8}$?

Notate che è possibile implementare il metodo di Jacobi con un solo ciclo `for` sulle iterazioni del metodo. Come?

3.3 Il metodo di Gauss–Seidel

La formula che definisce il metodo di Gauss–Seidel per la risoluzione del sistema $Ax = b$ è invece

$$x_i^{(new)} = \frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(new)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(old)} \right), \qquad i = 1, \ldots, n.$$

Notate che l'unica cosa che cambia rispetto a Jacobi è il (new) in rosso.

Esercizio 5. Scrivere una funzione `function x=gaussseidel(A,b,k)` che esegua `k` passi del metodo di Gauss–Seidel. Dopo ogni passo, scrivere sullo schermo la norma-2 del residuo con l'istruzione `norm(A*x-b)` (senza punto e virgola finale).

Suggerimento: come prima, aggiornate gli elementi $1, 2, 3, \ldots$ in quest'ordine: ad ogni passo i, nell'equazione che corrisponde alla riga i-esima della matrice c'è solo un elemento incognito $x_i^{(new)}$, quindi potete calcolarlo risolvendo l'equazione. In questo modo evitate di dover risolvere il sistema triangolare superiore con `sup_solve`...

Testare il metodo sulle quattro matrici di test. Quante iterazioni servono per ognuna delle matrici perché il residuo scenda sotto $\varepsilon = 10^{-8}$?