1 Risoluzione di sistemi lineari

Data una matrice quadrata $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ e un termine noto $\mathbf{b} \in \mathbb{R}^n$, siamo interessati a trovare la soluzione $\mathbf{x} \in \mathbb{R}^n$ del **sistema lineare**

$$A\mathbf{x} = \mathbf{b}. \tag{1}$$

Come motivazione all'introduzione dei **metodi diretti** per la risoluzione di (1), vediamo alcuni possibili approcci e i rispettivi comandi MATLAB. In conclusione, vedremo che alcuni di questi sono generalmente *sconsigliati*. In generale, la soluzione tramite MATLAB di (1) può avvenire in due modi principali: calcolando l'**inversa** della matrice A tramite il comando `inv`, oppure utilizzando l'operatore **backshlash** `\` .

1.1 Calcolo dell'inversa tramite il comando `inv`

La soluzione del sistema (1) **esiste** se e solo se la matrice A è **non singolare**. Per definizione, questo significa che esiste una matrice quadrata A^{-1} di ordine n per cui $AA^{-1} = A^{-1}A = I_n$, laddove I_n è la matrice identità di ordine n (`eye(n)`). La soluzione si può allora scrivere come

$$\mathbf{x} = A^{-1}\mathbf{b}. \tag{2}$$

La matrice A^{-1} viene chiamata **matrice inversa** di A e viene calcolata da MATLAB con il comando `inv`. Provare, ad esempio, i seguenti comandi

```
A = pascal(3)          %matrice di pascal di ordine 3
B = inv(A)             %inversa di A
A*B - eye(3)           %prova: AB-I_3
norm(A*B -eye(3))      %prova: norma 2 di AB-I_3
```

Tuttavia, in generale il calcolo dell'inversa per risolvere sistemi lineari richiede più operazioni aritmetiche del necessario e produce una risposta meno accurata, quindi questo approccio è generalmente *sconsigliato*.

Per comprendere meglio e convincersi di questo, è interessante considerare un caso estremo ma illustrativo monodimensionale, in cui il sistema lineare è ridotto all'equazione

$$7x = 21. \tag{3}$$

Il modo migliore per risolvere questo sistema è tramite la divisione con l'operatore `x=21/7` che darà il risultato esatto 3. L'uso della 'matrice' inversa $7^{-1} = 0.1429$, condurrà al risultato inesatto `0.1429*21=3.0009`, avendo fatto operazioni in più (una inversione e un prodotto). Consideriamo infine un secondo esempio di calcolo dell'inversa che produce un risultato meno accurato rispetto al precedente con la matrice di Pascal

```
A = [1 4 -2; 7 9 0; 2 -3 1]
B = inv(A)
A*B - eye(3)
norm(A*B -eye(3))
```

Idealmente $A*B$ dovrebbe produrre la matrice identità. Quello che osserviamo in pratica è che, siccome `inv` calcola l'inversa utilizzando operazioni floating-point, $A*B$ è vicino alla matrice identità, ma non esattamente uguale.

1.2 Operatori Slash e Backslash di MATLAB

Lo strumento fondamentale per risolvere un sistema lineare di equazioni in MATLAB è l'operatore **backslash** \. Innanzitutto, MATLAB dedica due tipi di comandi diversi alla divisione matriciale

- l'operatore **slash** / o **divisione matriciale a destra**: sia dato il sistema lineare

$$XA = B, \tag{4}$$

  con A a destra di X e B avente lo stesso numero di colonne di A, allora $X = B/A$ corrisponde alla divisione matriciale a destra, che è approssimativamente equivalente a `B*inv(A)`, ad eccezione del fatto che è calcolato in maniera diversa;

- l'operatore **backshlash** \ o **divisione matriciale a sinistra**: sia dato il sistema lineare

$$AX = B, \tag{5}$$

  allora $X = A\backslash B$ corrisponde alla divisione matriciale a sinistra, che è approssimativamente equivalente a `inv(A)*B`, ad eccezione del fatto che è calcolato in maniera diversa.

Facciamo osservare che il prodotto matriciale **non** è **commutativo**, quindi le due operazioni di sopra sono due operazioni diverse. Questi operatori non calcolano l'inversa di una matrice ma usano algoritmi differenti per trattare diversi tipi di matrici (cf. argomenti successivi). In generale, questa notazione si applica anche quando A non è una matrice quadrata, quindi quando partiamo da un sistema il cui numero di equazioni non coincide con il numero di incognite.

In generale, risolvere il sistema (1) tramite `A\b` è 2-3 volte più vecole rispetto a `inv(A)*b` e spesso produce un residuo minore. Questo lo possiamo vedere meglio tramite il seguente esempio

```
n = 500;
Q = orth(randn(n,n));
d = logspace(0,-10,n);
A = Q*diag(d)*Q';
x = randn(n,1);
b = A*x;
```

Questo esempio crea una matrice random A di ordine 500 tale che il suo numero di condizionalmente è $1e10$ e la sua norma è 1. Imponiamo esplicitamente la soluzione esatta **x** come un vettore random di lunghezza 500, e calcoliamo esplicitamente il termine noto. Lo scopo di avere costruito la matrice così è di considerare un sistema mal condizionato, ma consistente.

Risolviamo il sistema lineare con il comando `inv` e misuriamo il tempo computazionale utilizzando i comandi `tic` e `toc`

```
tic
y = inv(A)*b;
t = toc
```

Troviamo l'errore assoluto e residuo del calcolo

```
err_inv = norm(y-x)
res_inv = norm(A*y-b)
```

Ora, risolviamo lo stesso sistema lineare utilizzando l'operatore backslash \, misurando il tempo computazionale. Infine, ripetiamo i calcoli per trovare l'errore assoluto e residuo

```
tic
z = A\b;
t1 = toc
err_bs = norm(z-x)
res_bs = norm(A*z-b)
```

In conclusione, osserviamo una migliore accuratezza con l'utilizzo del comando backslash con un errore assoluto `2.6879e-06` e residuo `2.9833e-15`, in confronto all'errore assoluto `3.8113e-06` e residuo `4.3393e-07` prodotti con il comando `inv`.

1.3 Regola di Cramer

È meritevole richiamare brevemente un noto approccio teorico e discuterne l'applicabilità. È noto che la matrice A è **non singolare** se e solo se il **determinante** di A è **non nullo**. In questo caso, la **regola di Cramer** permette di calcolare la componente i-esima x_i della soluzione in funzione dei seguenti $n+1$ determinanti

$$x_i = \frac{\det(A_i)}{\det(A)}, \quad i = 1, \ldots, n, \tag{6}$$

dove A_i è la matrice ottenuta da A sostituendo la i-esima colonna con $\mathbf{b}$.

Come abbiamo visto alcune lezioni fa, il calcolo del determinante con la regola di Laplace ha complessità fattoriale. In termini di **costo computazionale**, questo significa che su un calcolatore in grado di eseguire 10^9 *flops* (i.e. 1 *Giga flops*) servirebbero circa 12 ore per risolvere un sistema di dimensione $n = 15$, 3240 anni se $n = 20$ e 10^{143} anni se $n = 100$. Il costo computazionale può essere drasticamente ridotto e portato all'ordine di circa $n^{3.8}$ operazioni se gli $n+1$ determinanti vengono calcolati con un altro algoritmo anzichè la regola di Laplace. Tuttavia, tale costo è ancora troppo elevato per problemi pratici. Nuovamente, questo risulta un approccio *sconsigliato*.

2 Metodi diretti - parte 1

Queste e altre osservazioni fanno capire la necessità di sviluppare **metodi numerici** per la risoluzione di sistemi lineari che garantiscano una *buona accuratezza* della soluzione approssimata, allo stesso tempo mantengano un *basso costo computazionale*. Ci sono due possibili approcci alternativi:

- ⋆ **metodi diretti** che calcolano la soluzione del sistema in un *numero finito di passi*
- ⋆ **metodi iterativi** che necessitano (teoricamente) un *numero infinito di passi*

In generale, la scelta fra un metodo diretto ed uno iterativo è determinata da vari fattori fra cui

- ⋆ l'efficienza teorica dello schema
- ⋆ il particolare tipo di matrice
- ⋆ le richieste di occupazione di memoria
- ⋆ il tipo di computer disponibile

Parlando di *costo computazionale*, richiamando la forma componenti per componenti di (1)

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned} \tag{7}$$

si vede esplicitamente che ci sono n^2 entrate della matrice A, e se le equazioni sono tra loro veramente accoppiate come in (7), è lecito aspettarsi che ognuna delle entrate venga almeno una volta interessata da una operazione. In conclusione, questo significa che (1) non potrà essere risolto con meno di n^2 operazioni.

2.1 Sistemi triangolari

Un generico sistema **triangolare inferiore** si può scrivere come

$$L\mathbf{y} = \mathbf{b} \tag{8}$$

dove la matrice triangolare inferiore $L = (l_{ij}) \in \mathbb{R}^{n\times n}$ è tale che $l_{ij} = 0$, se $i < j$

$$L = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix}. \tag{9}$$

Un generico sistema **triangolare superiore** si può scrivere come

$$U\mathbf{x} = \mathbf{y} \tag{10}$$

dove la matrice triangolare superiore $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ è tale che $u_{ij} = 0$, se $i > j$

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}. \tag{11}$$

Esempio 2.1 (Sostituzioni in avanti)**.** *Sia $L \in \mathbb{R}^{n\times n}$ triangolare inferiore non singolare, allora il sistema $L\boldsymbol{y}=\boldsymbol{b}$ si può scrivere in forma estesa come*

$$\begin{aligned} &l_{11}y_1 = b_1, \\ &l_{21}y_1 + l_{22}y_2 = b_2, \\ &\quad\vdots \\ &l_{n1}y_1 + l_{n2}x_2 + \cdots + l_{nn}x_n = b_n. \end{aligned} \tag{12}$$

In particolare, essendo L non singolare, $l_{11} \neq 0$, quindi possiamo risolvere la prima riga $y_1 = b_1/l_{11}$, e sostituire il valore di y_1 nelle successive equazioni. Procedendo nella stessa maniera ricaviamo la formua ricorsiva che risolve (12)

$$\begin{aligned} y_1 &= \frac{1}{l_{11}} b_1 \\ y_i &= \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right), i = 2, \dots, n. \end{aligned} \tag{13}$$

```
function [y]=forward(L,b)
 % y=forward(L,b)
 % Risolve il sistema triangolare inferiore Ly=b
 % usando il metodo delle sostituzioni in avanti.
 [n,m]=size(L);
 if n~=m, error('Solo sistemi quadrati'); end
 if min(abs(diag(L)))==0, error('Il sistema e'' singolare'); end
 y(1,1)=b(1)/L(1,1);
 for i=2:n
   y(i,1)=(b(i)-L(i,1:i-1)*y(1:i-1,1))/L(i,i);
 end
end
```

Si osservi in particolare l'implemetazione vettorizzata (in inglese *vectorized implementation*) del prodotto scalare corrispondente alla somma

$$\sum_{j=1}^{i-1} l_{ij}y_j = L(i,1:i-1) * y(1:i-1,1). \tag{14}$$

Esempio 2.2 (Sostituzioni all'indietro)**.** *Sia $U \in \mathbb{R}^{n\times n}$ triangolare superiore non singolare, allora il sistema $U\boldsymbol{x}=\boldsymbol{y}$ si può scrivere in forma estesa come*

$$\begin{aligned} u_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= y_1, \\ u_{21}x_2 + \cdots + a_{2n}x_n &= y_2, \\ &\vdots \\ u_{nn}x_n &= y_n. \end{aligned} \tag{15}$$

In questo caso, essendo U non singolare, $u_{nn} \neq 0$, quindi possiamo risolvere l'ultima riga $x_n = y_n/u_{nn}$, e sostituire il valore di y_n nelle equazioni precedenti. Procedendo nella stessa maniera ricaviamo la formua ricorsiva che risolve (15)

$$\begin{aligned} x_n &= \frac{1}{u_{nn}} y_n \\ x_i &= \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j \right), i = n-1, \ldots, 1. \end{aligned} \tag{16}$$

```
function [x]=backward(U,y)
 % x=backward(U,y)
 % Risolve il sistema triangolare superiore Ux=y.
```

```
 % Usa il metodo delle sostituzioni all'indietro.
 [n,m]=size(U);
 if n~=m, error('Solo sistemi quadrati'); end
 if min(abs(diag(U)))==0, error('Il sistema e'' singolare'); end
 x(n,1)=y(n)/U(n,n);
 for i=n-1:-1:1
   x(i,1)=(y(i)-U(i,i+1:n)*x(i+1:n,1))/U(i,i);
 end
end
```

Si può facilmente dimostrare che la complessità di questi algoritmi è quadratica.

Ultima versione aggiornata: 13 Maggio 2020

Link alle lezioni precedenti:
Lezioni 7-8
Lezioni 5-6
Lezioni 3-4
Lezioni 1-2

Referenze

1. MATLAB® The language of technical computing: computation, visualization, programming, The MathWorks Inc
2. MATLAB: An introduction with applications, A. Gilat, Wiley
3. Scientific Computing with MATLAB and Octave, A. Quarteroni, , F. Saleri, P. Gervasio, Springer
4. MATLAB Guide1 D. J. Higham and N. J. Higham, SIAM
5. Numerical Computing with MATLAB, C. Moler, SIAM