# Matrici e sistemi lineari

**Norme di vettori e matrici**
È spesso utile valutare la norma di vettori o matrici. Matlab fornisce per questo la funzione *norm*. Vediamone l'uso, richiamando le definizioni.

>> norm(X,k)

Se X è un vettore:

norm(X,p) % indica la p-norma di X [=SUM(ABS(X).^p)^(1/p)], per ogni p>=1.
norm(X)=norm(X,2)
norm(X,Inf) % indica la norma infinito di X, cioè il più grande elemento di abs(X)
norm(X,-Inf) % fornisce il più piccolo elemento di abs(X)

Se X è una matrice mxn :

norm(X,1) % norma matriciale indotta dalla norma 1 (=max somma per colonne dei valori
% assoluti)
norm(X,Inf) % norma matriciale indotta dalla norma infinito (=max somma per righe dei
% valori assoluti)
norm(X)=norm(X,2) % norma matriciale indotta dalla norma due (=sqrt(rho(X' X))), con rho
% raggio spettrale di X, cioè il più grande valore singolare di X.
% Se X simmetrica, coincide con l'autovalore di X di massimo modulo
norm(X,'fro') % norma di Frobenius (=sqrt(trace(X' X))

Nella risoluzione dei sistemi lineari risulta fondamentale il buon **condizionamento** della matrice del sistema (v. richiami di teoria). Matlab fornisce il comando *cond*:

>> cond(X,p) % = norm(X,p) * norm(inv(X),p)

dove p=1,2,Inf oppure 'fro' [ e cond(X)=cond(X,2), e coincide col rapporto tra il più grande e il più piccolo valore singolare di X].

**Esercizio 1**. Presa una matrice quadrata A (nxn), calcolare le norme matriciali 1, 2, infinito e Frobenius di A usando opportunamente i comandi di Matlab *max, sum, sqrt, trace, eig* e le usuali operazioni tra matrici. Poi confrontare i valori ottenuti con quanto fornito dai comandi *norm* corrispondenti.

**Esercizio 2**. Si consideri la matrice di Hilbert, definita da A(i,j)=1/(i+j-1), notoriamente mal condizionata. Verifichiamolo risolvendo un sistema lineare associato e guardando l'errore commesso in funzione del numero di condizionamento. Si esegua lo script seguente:

```
A=hilb(n);
solex=ones(n,1);
b=A*solex;
x=A \ b
err=norm(x-solex)/norm(solex)
cond(A)
```

per n=10, 12, 15, osservando gli effetti del peggioramento del condizionamento di A.

Ricordando altre funzioni Matlab per le matrici, provate ad eseguire anche i seguenti comandi (facendo le considerazioni opportune):

```
>> A=hilb(10);
>> det(A)
>> format long
>> L=eig(A)      % calcola il vettore degli autovalori di A
>> max(L)/min(L)
```

Calcolare anche il condizionamento rispetto alle altre norme matriciali.

**Costruzione di matrici sparse**

In molte applicazioni compaiono matrici di dimensioni notevoli, ma essenzialmente sparse, cioè con pochi valori diversi da zero, e spesso concentrati lungo delle diagonali. Vediamo come Matlab aiuti a costruire queste matrici risparmiando operazioni inutili (moltiplicazioni per zero) e occupazione di memoria.

Esempio 1.

Supponiamo ad esempio di voler costruire una matrice A di dimensioni 7x7 i cui unici elementi non nulli sono a(2,3)=1, a(3,6)=-2, a(5,5)=3, a(7,1)=1. Potremo scrivere

```
>> i=[2,3,5,7]; j=[3,6,5,1]; v=[1,-2,3,1];
>> A=sparse(i,j,v)
>> T=full(A)
>> nnz(A)
>> find(A)
```

I tre vettori sono rispettivamente il vettore degli indici i, quello degli indici j e quello dei valori. Nella sua forma *sparse* vengono tenuti in memoria solo gli elementi non nulli insieme ai suoi indici. Per vedere l'intera matrice dovremo 'riempirla con gli zeri' attraverso il comando *full*. La funzione *nnz*(A) restituisce il numero di elementi non zero di A. Cosa trova il comando *find* ?

Esempio 2.

Supponiamo ora di voler costruire una matrice tridiagonale A 9x9, con tutti 2 sulla diagonale principale e -1 nelle due diagonali adiacenti. Possiamo usare il comando *spdiags* (sparse diagonals):

```
>> e=ones(9,1);
>> A=spdiags([-e 2*e -e], -1:1, 9,9);
>> full(A)
```

In generale spdiags(B,d,m,n) crea una matrice sparsa mxn ponendo le colonne della matrice B lungo le diagonali specificate da d (0=diagonale principale, interi positivi = sopradiagonali, negativi = sottodiagonali).

Ricordiamo poi i comandi *tril(A)* e *triu(A)* che estraggono la parte triangolare inferiore o superiore di una matrice A. Nell'esempio seguente estraiamo una matrice tridiagonale da una piena:

```
>> F=floor(10*rand(8)); T=triu(tril(F,1),-1)   % capite come funziona?
```

**Risoluzione dei sistemi lineari**
Il comando \ (backslash):
>> x=A\b
calcola la soluzione x (vettore colonna nx1) del sistema Ax=b, se A è una matrice quadrata nxn e b un vettore colonna nx1. Risolve il sistema con l'algoritmo più efficiente, dopo aver effettuato test preliminari su A. Ad esempio se A è triangolare inferiore, usa il metodo della sostituzione in avanti; se è triangolare superiore quello della sostituzione all'indietro.

Il comando / (slash):
>> y=b/A
calcola la soluzione del sistema yA=b, se ora y e b sono vettori riga 1xn (corrisponde alla divisione tra array).

**Esercizio 3**.
Se A e il vettore e sono quelli dell'Esempio 2 , sia B=10*A, b=e/10 e si risolva il sistema lineare Bx=b nei tre modi diversi:

>> x=inv(B)*b
>> x=B\b
>> x=(b'/B)'

**Esercizio 4**.
Scrivere una funzione di Matlab che implementi il metodo della sostituzione in avanti. Applicarla nella risoluzione del sistema Lx=b=ones(n,1), dove L è la parte triangolare inferiore di una matrice A nxn. Calcolare alla fine il residuo r.

**Esercizio 5**.
Confrontiamo l'efficienza della risoluzione di un sistema tridiagonale per n=20,50,500,1000 memorizzando nel modo *sparse* oppure no:

>> e=ones(n,1); d=-2*e;
>> T=spdiags([e d e],[-1 0 1],n,n); A=full(T);
>> b=ones(n,1); s=sparse(b);
>>tic, T\s; sparsetime=toc
>>tic, A\b; fulltime=toc

**Stima del numero di condizionamento di una matrice**
Poiché k(A)=||A|| ||inv(A)||, il calcolo esatto di k richiede la costosa operazione di invertire la matrice A. Vediamo come si possa arrivare ad una stima di k con molto meno lavoro.
Siano y_1,y_2,...,y_m m vettori arbitrari e calcoliamo z_i=A y_i ; allora, ricordando la definizione di norma di matrice: || y_i ||/|| z_i ||<= || inv(A) ||, e possiamo considerare
C= max || y_i ||/|| z_i || una stima dal basso della norma della matrice inversa, così che
C || A || sarà una stima per difetto del numero di condizionamento k.

**Esercizio 6**.
Scrivere un programma che data una matrice quadrata A nxn calcola una stima per difetto di k(A) rispetto alla norma infinito usando per esempio i vettori y_i=e_i (base di R^n), e poi confronta il valore ottenuto con quello fornito dalla funzione Matlab *cond*.
Applicarlo alla matrice A=[1,1,0.7;2,-3,-18;1,2,-1]. Ripetere poi l'esperimento con la matrice di Hilbert di dimensioni 10x10 (da cui si vede che non si ottiene necessariamente una buona stima).

**Decomposizione di matrici**
Per risolvere un sistema lineare spesso è utile decomporre la matrice nel prodotto di una matrice triangolare inferiore per una triangolare superiore. Matlab fornisce per questo i comandi *lu* e *chol*.

>> [L,U]=lu(A)
produce una matrice U triangolare superiore e una matrice L che una permutazione di righe può rendere triangolare inferiore con tutti gli elementi diagonali uguali a 1, con LU=A

>> [L,U,P]=lu(A)
produce una matrice U triangolare superiore, una matrice L triangolare inferiore con tutti gli elementi diagonali uguali a 1 e P una matrice di permutazione tali che LU=PA.

>> R=chol(A)
se A è una matrice simmetrica definita positiva (altrimenti errore), produce una matrice R triangolare superiore con tutti gli elementi diagonali positivi, tale che R' R=A.

**Esercizio 7**.
Scrivere un programma che data una matrice A simmetrica definita positiva la decompone mediante la funzione *chol* e poi risolve il sistema Ax=b (per ogni vettore b assegnato) mediante la risoluzione successiva di due sistemi triangolari (si faccia uso di due funzioni scritte per questo scopo, v. Esercizio 4). Confrontare alla fine il risultato ottenuto con quanto fornisce Matlab direttamente (x=A\b).
[Come esempi si può considerare A=hilb(n), oppure T=spdiags([-e 2*e -e],[-1 0 1],n,n), con e=ones(n,1)]

**Metodi iterativi per la risoluzione di sistemi lineari (Jacobi, Gauss-Seidel)**
Vediamo come potrebbe essere una routine compatta per la risoluzione di un sistema lineare mediante il metodo di Jacobi con criterio d'arresto basato sull'incremento.
Commentare la verifica della dominanza diagonale. Cercare altre soluzioni altrettanto (o più) compatte per tale verifica (*).

```
function [x,nit]=Jacobi(A,b,toll,nmax,x0)
x=x0;
P=diag(diag(A)); N=P-A; % A=P-N
lambda=max(sum(abs(N'))./diag(abs(A))') %verifica dominanza diagonale
if lambda>1
    disp(' non c''e'' dominanza diagonale!')
end
for nit=1:nmax
xnew=P\(b+N*x); % P xnew = b + N x
inc=norm(x-xnew);
x=xnew;
if inc < toll, return, end
end
```

(*) Ecco un'altra possibilità (spiegare):

```
B=diag(1./diag(A))*A;
lambda=norm(B,Inf)-1
```

**Esercizio 8**.
Scrivere una analoga funzione per la risoluzione del sistema mediante il metodo di Gauss-Seidel con criterio d'arresto basato sulla norma del residuo.
Confrontare i due metodi per la matrice B=[9,14,-3;14,24,-6;-3,-6,5], con b=[1;1;1].

**Sistemi lineari sopra o sottodeterminati**
Se la matrice A è rettangolare (mxn) significa che il sistema è sopra o sottodeterminato, cioè il numero delle equazioni è maggiore o minore del numero delle incognite del problema.
Nel primo caso (m>n) l'istruzione x=A\b determinerà la soluzione ai minimi quadrati del sistema Ax=b, cioè il vettore x che minimizza la norma euclidea di (r'r), dove r=Ax-b indica il residuo del sistema .
Nel secondo caso (m<n) l'istruzione x=A\b determinerà la soluzione col maggior numero di componenti nulle.

Esempio.
Calcolare x=A\b e la norma 2 del residuo nei seguenti casi:
1) A=rand(5,2), b=ones(5,1).
2) A=rand(2,5), b=ones(2,1).

**Calcolo di autovalori e autovettori**
Matlab fornisce alcune funzioni specifiche allo scopo.

>> E=eig(A)

restituisce un vettore colonna E contenente gli autovalori di una matrice quadrata A

>> [V,D]=eig(A)

restituisce una matrice diagonale D con gli autovalori di A e una matrice piena V le cui colonne sono i corrispondenti autovettori. Ne segue che A*V=V*D (controllare)

>> E=eigs(A)

restituisce il vettore E dei 6 maggiori autovalori (in modulo) di una matrice quadrata A

>> E=eigs(A,k)

restituisce il vettore E dei k maggiori autovalori (in modulo) di una matrice quadrata A

>>[V,D]=eigs(A,k)

restituisce una matrice diagonale D con i k maggiori autovalori di A (in modulo) e una matrice piena V le cui colonne sono i corrispondenti autovettori.

**Esercizio 9.**
Data una matrice quadrata A, verificare usando le funzioni Matlab che la traccia di A coincide con la somma dei suoi autovalori, il determinante di A col loro prodotto.

**Esercizio 10**.
Data una matrice quadrata A non simmetrica, calcolare i suoi autovettori sinistri, cioè le righe della matrice W tale che W*A=D*W.

**Esercizio 11**.
Sia C una matrice quadrata definita positiva. Calcolare solo il suo autovalore minimo mediante i comandi *inv* e *eigs*.

**Sui grafici in animazione**
Esempio di grafico 'animato':

```
%grafico in animazione
clear
x=0:0.01:1;
ff=@(x)x.*(1-x);
for j=1:30
   plot(x,ff(x)/j)
    axis([0 1 0 0.25])
    xlabel(['iterazione = ',num2str(j)])
    pause(0.2);
end
```

Osservare l'uso di *pause*. Che succede se si usa senza argomento? Che succede se non si usa il comando *axis*? Che succede se si inserisce *hold on* nel ciclo for?

**Esercizio 12**.
Consideriamo una sbarra metallica posta sull'intervallo L=[0,1] dell'asse reale che viene raffreddata alla temperatura di zero gradi agli estremi. Se la distribuzione iniziale di temperatura della sbarra è data in ogni punto dalla funzione u0(x)=x(1-x), la sua evoluzione nel tempo è allora regolata dall'equazione del calore omogenea: u_t = u_{xx}.
Se suddividiamo L mediante m punti x_i=(i-1)h, per h=1/(m-1), la temperatura u(t_n,x_i) ai tempi t_n=n*dt nei punti interni x_i per un dato passo temporale dt può essere approssimata mediante la soluzione del seguente schema iterativo (*Eulero implicito*):

(*)  Bu^(n+1)=u^n  per n=0,1,2,...  con u^0 calcolato a partire da u0(x)

dove B è la matrice tridiagonale (m-2)x(m-2) con valori (1+2C) sulla diagonale principale e (-C) sulle due diagonali adiacenti, per C=dt/(h^2).
Costruire la matrice B per assegnati valori di m e dt, dimostrare che è definita positiva e calcolarne l'indice di condizionamento.
Scrivere un programma Matlab che, assegnati m e dt implementi lo schema precedente risolvendo ad ogni iterazione il sistema (*). Usare la decomposizione di Choleski per risparmiare costi di calcolo.
Rappresentare graficamente l'andamento della temperatura per N=15 passi temporali nella stessa finestra grafica. In alternativa disegnare il grafico in animazione facendo uso del comando *pause*.
Ripetere i test per differenti valori di m, dt e N.