1 Condizionamento di sistemi lineari

L'oggetto di questa sezione riguarda la domanda: *quanto è accurata la risoluzione numerica di un sistema lineare*?

Nelle lezioni precedenti (cf. Lezioni 9-10 e 11-12) abbiamo avuto modo di familiarizzare con i metodi numerici per la risoluzione di sistemi lineari e con le loro peculiarità. In conclusione, l'uso del **pivoting per righe** nella fattorizzazione LU di una matrice A consente di contenere eventuali errori di arrotondamento (in pratica, questo è già sufficiente a produrre buoni risultati, sebbene in teoria quello totale risulta essere superirore). Ricordiamo che entrambi i comandi `inv` e `\` fanno ricorso a fattorizzazioni LU nei loro codici interni. È ragionevole quindi aspettarsi che questa tecnica produca una soluzione accurata del sistema da risolvere. In realtà, questo **non** è sempre **vero**, ma dipenderà fortemente dal **condizionamento del sistema lineare**. Più precisamente, consideriamo un sistema lineare nella sua forma generale

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

laddove $A = (a_{ij})$ è una matrice quadrata in $\mathbb{R}^{n\times n}$ e $\mathbf{b} \in \mathbb{R}^n$ è un termine noto, e supponiamo che $\mathbf{b}$ sia soggetto ad una *perturbazione* $\delta\mathbf{b}$. Di conseguenza, anche la soluzione di (1) sarà soggetta ad una *perturbazione* $\delta\mathbf{x}$ tale che

$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}. \tag{2}$$

In realtà, quando si risolve numericamente un sistema lineare (1) su un computer non si trova la soluzione esatta $\mathbf{x}$ di (1), bensì la *soluzione esatta*

$$\bar{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x} \tag{3}$$

di un sistema lineare *perturbato* nella forma (2), laddove $\delta\mathbf{b}$ è un vettore di perturbazione che dipende dallo specifico metodo numerico impiegato nella risoluzione del sistema. Uno studio più completo dovrebbe includere anche una matrice di perturbazione δA, cosicché la (2) diventi $(A + \delta A)\bar{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$. Per semplicità dei calcoli, supporremo $\delta A = 0$. È di fondamentale importanza, di conseguenza, che sappiamo rispondere alla domanda

- ▷ quanto *inaccurata* può essere la soluzione $\bar{\mathbf{x}}$ restituita da un computer (che risolve in maniera esatta la (2)) rispetto alla soluzione vera esatta $\mathbf{x}$ della (1)?

In seguito, vediamo che l'inacuratezza che ci possiamo aspettare dipende dal numero di condizionamento della matrice A.

1.1 Numero di condizionamento

Consideriamo una qualsiasi norma matriciale $\|\bullet\|$ indotta da una vettoriale $\|\bullet\|$. Generalmente si utilizza la stessa notazione e la determinazione se si tratti di una norma matriciale o vettoriale viene lasciata con ovvietà al contesto. Il **numero di condizionamento** della matrice A è definito come

$$\kappa(A) = \|A\|\|A^{-1}\|. \tag{4}$$

Il motivo per cui questa quantità è importante è perché si può facilmente dimostrare che esso caratterizza la perturbazione (relativa) in norma della soluzione in funzione della perturbazione (relativa) sul termine noto tramite la disuguaglianza

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}. \tag{5}$$

La disuguaglianza (5) si può riscrivere come

$$\frac{\|\mathbf{x}-\bar{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}, \tag{6}$$

avendo osservato che la perturbazione $\delta\mathbf{x}$ rappresenta l'errore $\delta\mathbf{x} = -(\mathbf{x}-\bar{\mathbf{x}})$ e la perturbazione $\delta\mathbf{b}$ rappresenta il **residuo** della (2) definito come

$$\delta\mathbf{b} = A\bar{\mathbf{x}} - \mathbf{b} = -\mathbf{r} \tag{7}$$

Il residuo è una quantità molto importante in analisi numerica, grazie alla sua relazione stretta con l'errore. Se $\bar{\mathbf{x}}$ fosse la soluzione esatta, il residuo sarebbe il vettore nullo. Di conseguenza, esso può essere ritenuto uno **stimatore dell'errore** commesso $\mathbf{x}-\bar{\mathbf{x}}$. La sua efficacia come stimatore dell'errore dipende dalla grandezza del numero di condizionamento di A.

L'espressione (6) mette esplicitamente in relazione la **norma (relativa) dell'errore** $\|\mathbf{x}-\bar{\mathbf{x}}\|/\|\mathbf{x}\|$ con la **norma (relativa) del residuo** $\|\mathbf{r}\|/\|\mathbf{b}\|$ in funzione di $\kappa(A)$ e ci dice che

- ⋆ *un residuo piccolo non garantisce un errore piccolo*:
  - ∗ se $\kappa(A)$ è "piccolo", la (6) dà la certezza che l'errore sarà piccolo se lo è il residuo, mentre ciò non è garantito se *kappa*(A) "grande"
  - ∗ per valori molto grandi di $\kappa(A)$ (solitamente $\kappa(A) > 10^3$) l'errore relativo sulla soluzione può essere molto grande anche se è piccolo l'errore relativo sui dati (cf. Esercizio 1.1).

Si può facilmente vedere che il numero di condizionamento di una matrice è sempre maggiore uguale a 1, in quanto si ha che

$$1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \kappa(A). \tag{8}$$

In generale, se $\kappa(A)$ è "piccolo", ossia dell'ordine dell'unità, la matrice A viene detta **ben condizionata** ed a piccoli errori sui dati corrisponderanno errori dello stesso ordine di grandezza sulla soluzione.

Al contrario, se il numero di condizionamento è molto alto, si parla di **sistemi malcondizionati**. In questi casi, può accadere che a piccole perturbazioni sui dati corrispondano grandi errori sulla soluzione, quindi *meno accurata* potrebbe essere la soluzione del sistema lineare (cf. Esercizo 1.1).

Osserviamo che per matrici simmetriche e definite positive, la (4) diventa $\kappa(A) = \frac{\lambda_{max}}{\lambda_{mix}}$, laddove λ_{max} e λ_{mix} sono rispettivamente il più grande e il più piccolo autovalore della matrice.

Il comando `cond(A)` di MATLAB permette di calcolare il numero di condizionamento nella norma Euclidea $\| \bullet \|_2$ di una matrice A. Specificando anche un secondo argomento `cond(A,p)`, laddove $p = 1, 2$, inf oppure 'fro' si ottiene rispettivamente il numero di condizionamento in norma 1, 2, ∞ e quella di Frobenius di A.

```
A = [3 -1 8; 0 9 0; -1 2 2]
cond(A)          %numero di condizionamento in norma 2
cond(A,1)        %numero di condizionamento in norma 2
cond(A,inf)      %numero di condizionamento in norma infinito
cond(A,'fro')    %numero di condizionamento in norma di Frobenius
```

Per calcolare il numero di condizionamento di matrici sparse bisogna prima trasformare la matrice da formato sparso a formato pieno con il comando `A1=full(A)`. In alternativa, il comando `condest(A)` fornisce una stima (dal basso) del numero di condizionamento in norma 1 di A, laddove ricordiamo che $\|A\|_1 = \max_{i=1,\dots,n} |a_{ij}|$, che sarà ad ogni caso diverso dal risultato fornito dal comando `full(A)`. Ad esempio, riprendiamo la matrice sparsa corrispondente all'operatore differenziale del secondo ordine vista nella lezione precedente

```
n = 10
e = ones(n,1);
A = spdiags([ e -2*e e ], -1:1, n, n)
cond(A) %restituira' un warning indirizzando al comando condest
A1=full(A)
cond(A1)
```

Come lo aspettavamo, il comando `cond` restituisce `Warning: Using CONDEST instead of COND for sparse matrix` e calcola invece il numero di condizionamento restituito dal comando `condest`.

1.2 Effetti del malcondizionamento

Esempio 1.1 (Matrice di Hilbert)**.** *Una classe di matrici malcondizionate è fornita dalle* ***matrici di Hilbert*** *di ordine n i cui elementi sono*

$$h_{ij} = \frac{1}{1+i+j}, \quad i, j = 1, \dots, n. \tag{9}$$

In MATLAB tali matrici si creano tramite il comando `hilb(n)`*. Si può facilmente vedere che esse sono simmetriche e definite positive, al variare di n.*

```
n=10;          %fare diverse prove con vari n
H=hilb(n);
isequal(H,H') %controlliamo se H e' simmetrica
chol(H)        %controlliamo se H e' definita positiva
```

Il numero di condizionamento delle matrici di Hilbert è una funzione rapidamente crescente di n: si può dimostrare $\kappa(H_n) \simeq e^{3.5n}$.

```
%Questo script calcola le matrici di Hilbert e il loro numero
%di condizionamento per n da 1 a 20
for n = 1:20
    H = hilb(n); k=cond(H);

    %stampiamo i risultati
    fprintf( '\n \t n %3d  cond  %6.4e',n,k)
end

%risultati prodotti
n   1  cond  1.0000e+00
n   2  cond  1.9281e+01
n   3  cond  5.2406e+02
n   4  cond  1.5514e+04
n   5  cond  4.7661e+05
n   6  cond  1.4951e+07
n   7  cond  4.7537e+08
n   8  cond  1.5258e+10
n   9  cond  4.9315e+11
n  10  cond  1.6025e+13
n  11  cond  5.2202e+14
n  12  cond  1.6212e+16
n  13  cond  4.7864e+17
n  14  cond  2.5515e+17
n  15  cond  2.4960e+17
n  16  cond  4.8875e+17
n  17  cond  4.5144e+17
n  18  cond  1.3500e+18
n  19  cond  1.2579e+18
n  20  cond  2.1065e+18
```

Esercizio 1.1. *Si costruisca uno script che genera la matrice di Hilbert di ordine 13 e risolve il sistema lineare $H\mathbf{x} = \mathbf{b}$ con l'utilizzo del comando backslash, avendo definito a priori il termine noto $\mathbf{b}$ sapendo la soluzione esatta è $\mathbf{x} = [1, \ldots, 1]^T$. Si calcolino l'errore e il residuo relativi e si commentino i risultati.*

Soluzione:

```
n = 13;
H = hilb(n);
xesatta=ones(n,1);
b = H*xesatta;
kappa = cond(H)
x = H\b
err_rel = norm(x-xesatta)/norm(xesatta)
res_rel = norm(b-H*x)/norm(b)

%risultati
kappa = 4.7864e+17
err_rel = 1.4072
res_rel = 7.6177e-17
```

Osserviamo che MATLAB restituisce `Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 3.330077e-18.` Questo warning indica che la matrice è molto **malcondizionata** (tanto da essere ritenuta '*vicina ad una matrice singolare*') e quindi i risultati potrebbero *non* essere *attendibili*. Di fatto, diverse componenti della soluzione $\mathbf{x}$ calcolata sono abbastanza distinte da quelle della soluzione esatta $\mathbf{x} = [1, \ldots, 1]^T$

```
%vettore x'
Columns 1 through 7
   1.0000    1.0000    0.9997    1.0048    0.9574    1.2303    0.1991

Columns 8 through 13
   2.8538   -1.8841    3.9799   -0.9605    1.7432    0.8764
```

Inoltre, sebbene il residuo relativo sia molto piccolo, inferiore alla precisione di macchina, l'errore relativo è dell'ordine di 10^1. Questo risultato non è sorprendente, visto il numero di condizionamento molto grande della matrice.

Esempio 1.2 (Esempio di matrice malcodizionata: matrice di Vandermonde)**.** *Un altro esempio di matrice malcondizionata è dato dalle* ***matrici di Vandermonde****. Queste matrici si incontrano, ad esempio, nella teoria dell'interpolazione polinomiale. Dato un vettore* $\mathbf{x} = (x_1, \ldots, x_n)$*, le componenti della matrice di Vandermonde di ordine* n *sono*

$$v_{ij} = x_i^{n-1}, \quad i, j = 1, \ldots, n \tag{10}$$

In MATLAB tali matrici si creano tramite il comando `vander(x)`*.*

```
%Questo script calcola i numeri di condizionamento delle matrici
%di Vandermonde costruite su n punti equispaziati nell'intervallo
%punti equispaziati dell'intervallo [1,2], per n = 1,...,10

%inizializziamo a zero una matrice che conterra' i numeri di
%condizionamento
vcond=zeros(10);

for n=3:10
      x=1:1/(n-1):2; %n punti equispaziati tra 1 e 2
                     %significa che il passo e' h=1/(n-1)
      A=vander(x);
      c=cond(A);
      vcond(n)=c;
end
semilogy(3:10, vcond(3:10), 'b-*');
xlabel('dimensione della matrice n')
ylabel('log(con(n)')
title('Condizionamento della matrice di Vandermonde')
```

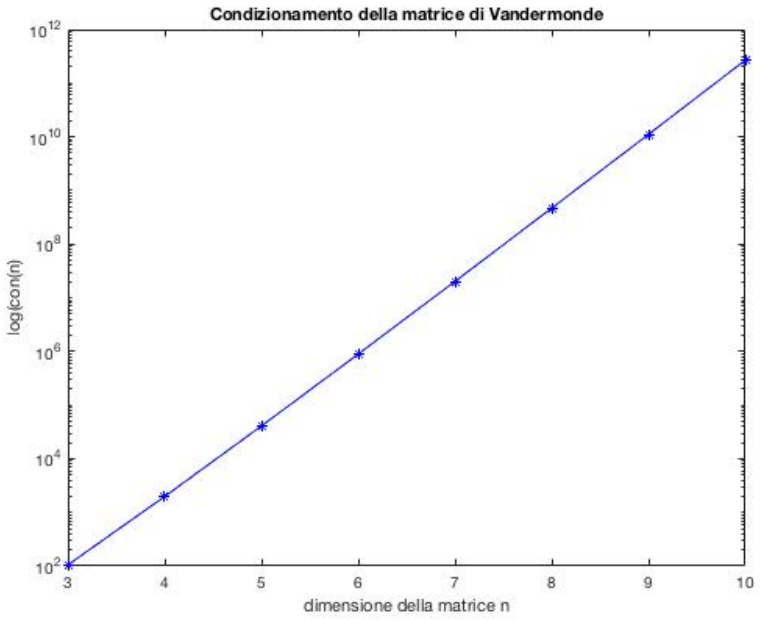

Figure 1: Condizionamento della matrice di Vandermonde in scala semilogaritmica al crescere della dimensione n

.

Gli effetti del malcondizionamento di questa matrice si vedranno nelle prossime lezioni sull'interpolazione polinonimale.

2 Calcolo di autovalori e autovettori

Consideriamo una matrice quadrata $A \in \mathbb{K}^{n\times n}$, laddove il campo $\mathbb{K} = \mathbb{R}, \mathbb{C}$. Un vettore *non-nullo* $\mathbf{v} \in \mathbb{K}^n$ viene detto **autovettore** di A se esiste uno scalare $\lambda \in \mathbb{K}$ tale che

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{11}$$

Lo scalare λ è chiamato **autovalore** di A corrispondente all'autovettore $\mathbf{v}$.

Si noti che la definizione stessa richiede che l'autovettore sia non-nullo. Di fatto, se così non fosse e $\mathbf{v} = \mathbf{0}$ fosse permesso come autovettore, allora ogni scalare λ sarebbe un autovalore in quanto $A\mathbf{0} = \lambda\mathbf{0}$ è vera per ogni $\lambda \in \mathbb{K}$. D'altra parte, possiamo avere $\lambda = 0$, e $\mathbf{v} \neq \mathbf{0}$.

Gli autovettori non sono mai unici. Se $\mathbf{v}$ è un autovettore per A con autovalore corrispondente λ, allora così lo è anche $c\mathbf{v}$, per ogni $c \in \mathbb{K}$ tale che $c \neq 0$. Questo si vede banalmente da

$$A(c\mathbf{v}) = cA\mathbf{v} = c(\lambda\mathbf{v}) = \lambda(c\mathbf{v}). \tag{12}$$

Più in generale, si definisce l'**autospazio** associato a λ come

$$\begin{aligned} E_\lambda &:= \{\mathbf{v} \in \mathbb{K}^n | A\mathbf{v} = \lambda\mathbf{v}\} \\ &:= \{\text{autovettori associati a } \lambda\} \cup \{\mathbf{0}\}. \end{aligned} \tag{13}$$

Osservando che la (11) si può riscrivere come $(A - \lambda I_n)\mathbf{v} = \mathbf{0}$, si deduce che $E_\lambda = \mathrm{Ker}(A - \lambda I_n)$. Definendo il **polinomio caratteristico** associato ad A come

$$p_A(\lambda) = \det(A - \lambda I_n), \tag{14}$$

si ha che λ è un autovalore di una data matrice $A \in \mathbb{K}^{n\times n}$ se e solo se $p_A(\lambda) = 0$. L'equazione $p_A(\lambda) = 0$ viene detta **equazione caratteristica** della matrice A. Siccome $p_A(\lambda)$ è un polinomio di grado n in λ, il **Teorema fondamentale dell'Algebra** garantisce l'esistenza di n radici (non necessariamente reali e distinti) dell'equazione caratteristica, e di conseguenza di n autovalori complessi di A contati con la loro molteplicità.

2.1 L'istruzione `eig`

MATLAB calcola autovalori e autovettori con il comando `eig` secondo la sintassi

- ⋆ `E=eig(A)` produce un vettore colonna E contenente gli autovalori della matrice quadrata A;
- ⋆ `[V,D] = eig(A)` produce una matrice diagonale D contenente gli autovalori e una matrice *piena* V le cui colonne sono gli autovettori corrispondenti tali che `A*V = V*D`;
- ⋆ `[V,D,W] = eig(A)` produce inoltre una matrice *piena* W le cui colonne sono gli autovettori sinistri tali che `W'*A = D*W'`; si ricordi che gli *autovettori sinistri* sono tali che $\mathbf{w}^T A = \lambda\mathbf{w}^T$;

⋆ `E = eig(A,B)` produce un vettore colonna E contenente gli autovalori generalizzati delle matrici quadrate A e B che soddisfano $A\mathbf{w} = \lambda B\mathbf{v}$.

Per una più ampia panoramica delle funzionalità di `eig` si faccia riferimento all'istruzione `help` di MATLAB. Per vedere alcuni esempi, facciamo riferimento al comando `gallery` di MATLAB che è utile a generare matrici test specificate da un `matrixname`. Illustriamo alcuni esempi.

```
%generiamo una matrice simmetrica definita positiva di ordine 4
A = gallery('lehmer',4)

%calcoliamo gli autovalori di A, memorizzati nel vettore colonna
e=eig(A)
```

Si ricordi che una matrice **circolante** è una matrice quadrata in cui ogni vettore riga è ruotato di un elemento a destra rispetto al vettore riga precedente. Questa è un particolare caso di matrice di **Toeplitz**.

```
%generiamo una matrice circolante di ordine 3
A = gallery('circul',3)

%calcoliamo gli autovettori di B, memorizzandoli come colonne
%in una matrice V, e gli autovalori, memorizzandoli come gli
%elementi diagonali di una matrice diagonale D
[V,D] = eig(A)

%verifichiamo che i risultati soddisfano A*V = A*D
A*V - A*D
```

Idealmente, la relazione dovrebbe essere esatta. Come visto abbastanza frequentemente, siccome `eig` fa i calcoli utilizzando operazioni floating-point, allora `A*V` può, at massimo, avvicinarsi al meglio possibile a `V*D`. In altre parole , `A*V - V*D` è vicino a, ma non esattamente la matrice nulla.

Di default, `eig` **non** restituisce sempre gli autovalori e autovettori **ordinati**. Per questo si fa riferimento alla funzione `sort` per ordinare gli autovalori in ordine crescente e poi riordinare gli autovettori corrispondenti. Vediamo un esempio concreto.

```
%Calcoliamo gli autovalori e autovettori di una matrice magica
%di ordine 5
 A = magic(5)
[V,D] = eig(A)
```

```
%selezioniamo il primo autovettore
v1=V(:,1) %estriamo la prima colonna di D

%selezioniamo il primo autovalore
d=diag(D)
lambda1=d(1)

%verifichiamo che A*v1= lambda1*v1
A*v1
lambda1*v1
```

Come si vede dal risultato, gli autovalori non sono ordinati. In questo caso, possiamo utilizzare le istruzioni `[d,ind] = sort(diag(D))` per prima costruire un vettore colonna che estrae gli elementi diagonali di D con il comando `diag(D)`, successivamente ordinare il vettore in maniera crescente.

```
%ordiniamo gli autovalori in ordine crescente
[d,ind] = sort(diag(D))
```

Il secondo output di `sort` restituisce il vettore di permutazione degli indici. Questo è utile per riordinare gli elementi diagonali di D, tramite le istruzioni

```
%riordiniamo gli elementi diagonali di D
Ds = D(ind,ind)
```

Infine, è necessario riordinare anche le colonne di V, per avere la corrispondenza corretta tra autovalori e autovettori. In questo caso si utilizza ancora il vettore `ind`, seguendo le istruzioni

```
%riordiniamo gli autovettori corrispondenti di V
Vs = V(:,ind)
```

Facciamo notare in conclusione, che sia (V, D) che (Vs, Ds) producono la decomposizione agli autovalori di A, con la differenza che gli ultimi sono ordinati in ordine crescente. I risultati `A*V-V*D` e `A*Vs-Vs*Ds` coincidono, a meno di errori round-off.

```
e1 = norm(A*V - V*D);
e2 = norm(A*Vs - Vs*Ds);
e = abs(e1 - e2)
```

Ricordiamo infine che il **raggio spettrale** di una matrice quadrata A è definito come il **modulo dell'autovalore di massimo modulo**

$$\rho(A) = \max_{i=1,\dots,n} \{|\lambda_i| \,|\, \lambda_i \text{ è autovalore di } A\}. \tag{15}$$

Esso si calcola in MATLAB tramite i comandi

```
rho = max(abs(eig(A)))
```

2.2 Condizionamento del calcolo degli autovalori

Dedichiamo una breve sezione al problema del **condizionamento** nell'ambito del **calcolo degli autovalori**.

Supponiamo che le entrate della matrice di partenza sono soggette ad una *perturbazione*, ad esempio, dovuta ad errori di arrotondamento durante la sua memorizzazione. Siamo interessati a sapere

- ▷ quanto *inaccurato* può essere il calcolo degli autovalori, in seguito a perturbazioni degli elementi di A?

Il seguente esempio mostra che possiamo aspettarci molta inaccuratezza.

```
A = [101, -90; 110 -98]

%perturbiamo gli elementi
epsilon=0.001

%matrice perturbata
Aperturb = [101-epsilon, -90-epsilon; 110 -98]
%fare attenzione agli spazi!

e=eig(A);
eperturb = eig(Aperturb)

%errore di perturbazione della matrice
norm(A - Aperturb)

%errore relativo sugli autovalori
err_rel_autovalori=abs(e- eperturb)./abs(e)
```

Questo risultato viene precisato in maniera rigorosa dal **Teorema di Bauer Filke**. In sostanza, il fattore determinante sarà il *condizionamento della matrice degli autovettori*: se essa è mal condizionata, allora il calcolo degli autovettori sarà mal condizionato.

2.3 Metodo delle potenze

Molto spesso, non è necessario conoscere tutto lo spettro di A (ossia l'insieme di tutti i suoi autovalori), ma soltato gli autovalori di modulo massimo e minimo. Il **metodo delle potenze** è un metodo iterativo che consente di approssimare l'autovalore λ_1 di massimo modulo di una data matrice e l'autovettore ad esso corrispondente. Data una matrice $A \in \mathbb{R}^{n\times n}$, assumiamo che i suoi autovalori siano ordinati in ordine non-decrescente

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|, \tag{16}$$

laddove abbiamo assunto che l'autovalore di modulo massimo ha moltepcilità uno ed è separato dagli altri. Sia $\mathbf{x}_1$ l'autovettore di lunghezza unitaria associato a λ_1. Se gli autovettori di A sono linearmente indipendenti, possiamo calcolare λ_1 e $\mathbf{x}_1$ tramite il seguente procedimento iterativo.

Esempio 2.1 (Metodo delle potenze)**.** *Dato un vettore iniziale arbitrario* $\mathbf{x}^{(0)} \in \mathbb{C}^n$ *e posto* $\mathbf{y}^{(0)} = \mathbf{x}^{(0)}/\|\mathbf{x}^{(0)}\|$*, si calcoli per k=1,2,...*

$$\begin{aligned} \mathbf{x}^{(k)} &= A\mathbf{y}^{(k-1)}, \\ \mathbf{y}^{(k)} &= \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|} \\ \lambda^{(k)} &= (\mathbf{y}^{(k)})^H A\mathbf{y}^{(k)} \end{aligned} \tag{17}$$

Il seguente script implementa il metodo delle potenze. Gli argomenti sono la matrice A, il vettore iniziale `x0`, la tolleranza `tol` impiegata nel criterio d'arresto ed il numero massimo di iterazioni consentite `kmax`. La procedura iterativa si arresta alla prima iterazione k in corrispondenza della quale si ha

$$|\lambda^{(k)} - \lambda^{(k-1)}| < \epsilon|\lambda^{(k)}|, \tag{18}$$

dove ϵ =`tol` è la tolleranza assegnata, oppure se il numero di iterazioni è più grande di `kmax`. Gli output in uscita sono l'autovalore di modulo massimo `lambda`, l'autovettore associato `x` ed il numero di iterazioni `iter` che sono state effettuate.

```
function [lambda,x,iter] = potenze(A,tol,kmax,x0)
%Calcola l'autovalore di modulo massimo tramite il Metodo
%delle potenze.
%Gli argomenti sono la matrice A, una tollerenza richiesta dal
%criterio d'arresto, il massimo numero di iterazioni, ed un
%vettore iniziale x0
%restituisce l'autovalore di modulo massimo lambda, l'autovettore
%corrispondente unitario, ed il numero di iterazioni
%necessarie per calcolare lambda

[n,m]=size(A);
if n~=m, error('La matrice non e'' quadrata'); end
```

```
if nargin==1
      tol=1.e-06; x0=ones(n,1); kmax=100;
 end
x0 = x0/norm(x0);
y = A*x0;
lambda = x0'*y;
err = tol*abs(lambda) + 1;
iter = 0;
   while(err> tol*abs(lambda) & abs(lambda)~=0 & iter <=kmax)
      x = y;
      x = x/norm(x);
      y = A*x;
      lambdanuovo = x'*y;
      err = abs(lambdanuovo - lambda);
      lambda = lambdanuovo;
      iter = iter + 1;
   end
end
```

Link alle lezioni precedenti:
Lezioni 11-12
Lezioni 9-10
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