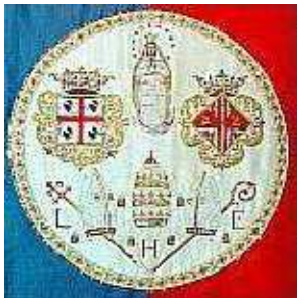
Università degli studi di Cagliari

Facoltà di ingegneria



Corso di laurea in ingegneria elettronica

Tesina per il corso di

Calcolo Numerico 2

Prof. Giuseppe Rodriguez

Autovalori e autovettori

A cura di

Pamela Sulis

Matricola 37850

Indice

Brevi richiami sulle matrici

Una matrice m^xn è una tabella di m^xn numeri reali, o complessi, disposti in m righe e n colonne

$$A=\begin{bmatrix} a11 & \cdots & a1n \\ \vdots & \ddots & \vdots \\ an1 & \cdots & ann \end{bmatrix}$$

Una matrice è quadrata se m=n, in caso contrario viene detta rettangolare.

Sia A appartenente a $C^{(mxn)}$. La matrice aggiunta si ottiene scambiando le righe di A con le sue colonne e coniugandone gli elementi

$$(A^*)_{ij}= a_{ji}$$

Se la matrice è reale, si parla di matrice trasposta.
L'elemento neutro è la matrice identità

$$I=diag(1,\ldots,1)$$

Una matrice quadrata A si dice invertibile o non singolare se esiste una matrice $A^{(-1)}$, detta matrice inversa, tale che $AA^{-1}=I$
In Matlab esistono differenti modi per costruire una matrice:

```
A=[1 2 3; 4 6 7]
B=rand(4,3)
C=[A;B]
```

Il determinante è una funzione che associa a ciascuna matrice quadrata un numero reale. Tralasciando la sua definizione formale,ricordiamo che, fissata una riga i, esso può essere calcolato mediante formula di Laplace :

$$\det(A)=\sum_{j=1}^{n}(-1)^{(i+j)}a_{ij}\det(A_{ij})$$

essendo A_{ij} la sottomatrice che si ottiene da A eliminando la i-esima riga e la j-esima colonna
In Matlab il determinante di una matrice si calcola usando la seguente istruzione

$$det(A)=\det(A)$$

Esempio

```
» A= [ 2 3 1; 4 6 8; 9 34 2]

A =

    2     3     1

    4     6     8

    9    34     2

» d=det(A)

d =

  -246
```

Il rango di una matrice può essere definito indifferentemente come il massimo numero di righe (colonne) linearmente indipendenti o come l'ordine della più grande sottomatrice con determinante non nullo.

L'istruzione Matlab che ci consente di determinare il rango di una matrice è la seguente:

rank(A)=rank(A)

```
n=rank(A)

n =
     3
```

Data una matrice quadrata A appartenente a $R^{n\times n}$, si chiama autovalore di A, quel numero reale o complesso β tale che per ogni vettore $x \neq 0$ soddisfa l'equazione $Ax = \beta x$.
Il vettore x viene detto autovettore associato all'autovalore β. Osserviamo che l'autovettore x non é unico. Infatti, se α, appartenente a R, è un qualsiasi numero reale non nullo, allora il vettore $y=\alpha x$ è ancora un autovettore associato all'autovalore β.
Se l'autovettore x è noto, il corrispondente autovalore si può determinare usando il quoziente di Rayleigh

$$\beta = (x^T Ax)/(x^T x) .$$

Dalla definizione precedente, segue che β è autovalore di A, se è una radice del polinomio caratteristico

$$p_a(\beta) = \det(A - \beta I) .$$

Infatti, l'equazione precedente è equivalente a

$$(A - \beta I)x = 0$$

ma essendo $x \neq 0$ essa sarà soddisfatta se e solo se la matrice $A-\beta I$ risulta essere singolare.
Inoltre, il polinomio caratteristico associato ad una matrice A di ordine n, ha n radici reali e/o complesse. Se β, appartenente a C, è un autovalore di A, anche β è un autovalore complesso di A.
Premettiamo due utili risultati circa gli autovalori di matrici: il primo ci ricorda che le trasformazioni per similitudine conservano gli autovalori, mentre il secondo ci ricorda che le matrici simmetriche hanno autovalori reali.
Definiamo come spettro di una matrice l'insieme di tutti i suoi autovalori

$$\sigma(A)= \{\beta_1,.....\beta_n\}$$

e raggio spettrale (rho) il massimo dei moduli degli autovalori.
In Matlab, per calcolare gli autovalori di A e il suo raggio spettrale, usiamo le seguenti istruzioni

```
β=eig(A)
rho=max(abs(β))
```

Esempio

```
» eig(A)
ans =

    0.9005
   21.6931
  -12.5936
```

Il polinomio caratteristico si ottiene con il comando

p=poly(A)

Esempio

```
» p=poly(A)
```

```
p =

    1.0000  -10.0000 -265.0000  246.0000
```

Si definisce numero di condizionamento di una matrice A, relativamente alla risoluzione di un sistema lineare, la quantità $k(A)= \|A\| \|A^{-1}\|$.
Il numero di condizionamento misura il massimo fattore di amplificazione dell'errore relativo sulla soluzione rispetto all'errore relativo sui dati. Il valore del numero di condizionamento è influenzato dalla norma matriciale adottata. Useremo un pedice quando vorremo porre in evidenza la norma usata per calcolarla.
Per calcolare il numero di condizionamento di una matrice in Matlab usiamo le seguenti istruzioni:
$k_2(A)$= cond(A)
$k_1(A)$= cond(A,1)
$k_{(\infty)}$= cond(A,inf)

Problema: calcolo degli autovalori

Problemi di calcolo degli autovalori compaiono in molte applicazioni:studio di vibrazioni e stabilità di strutture o circuiti, analisi di metodi iterativi , analisi di stabilità dei sistemi differenziali, chimica computazionale e altre. In ogni caso occorre valutare se, per il problema che si sta considerando, occorre calcolare l'intero spettro della matrice o solo alcuni autovalori.
Per calcolare autovalori e autovettori corrispondenti di una matrice A, si può trasformare la matrice A in una forma più semplice. Tale trasformazione è detta di similitudine e non deve alterare lo spettro della matrice.

Proposizione. Matrici simili hanno gli stessi autovalori
Dim. Siano A e B simili, ovvero $P^{-1}AP = B$, con P invertibile. Ora, se β è autovalore di A e $x \neq 0$ l'autovettore associato, allora si ha

$$BP^{-1}x = P^{-1}Ax = \beta P^{-1}x .$$

quindi β è autovalore di B con autovettore associato $P^{-1}x$.

Proposizione. Matrici simmetriche hanno autovalori reali.
Dim. Dapprima osserviamo che, per il coniugato del polinomio caratteristico di A, valgono le identità

$$\det(A - \beta I) = \det(A - \beta I)^* = \det(A^* - \beta_c I)$$

dove A* `e la matrice trasposta e coniugata di A. Si deduce allora che gli autovalori di A* sono i coniugati di quelli di A. Si conclude che $\beta=\beta$ se e solo se β appartiene ad R.

Prima di procedere nello studio del problema della determinazione di autovalori e autovettori ci interessa analizzare le matrici diagonalizzabili.
Una matrice A appartenente a R^{nxn} è diagonalizzabile se esiste una matrice X, appartenente a R^{nxn} non singolare tale che

$$X^{-1}AX=D= \text{diag}(\beta_{1,\ldots,}\beta_n)$$

con X che ha per colonne gli n autovettori di A (che formano una base per R^n).
Nel caso di matrice rettangolare non parliamo di autovalori ma di valori singolari e vale il seguente risultato, noto come decomposizione ai valori singolari (o SVD):
Sia A appartenente a $R^{m\times n}$. Allora esistono due matrici ortogonali U, appartenente a $R^{m\times m}$, e V, appartenente a $R^{n\times n}$ tali che

$$U^TAV = \sum$$

con $\sum = \text{diag}(\sigma_1, \ldots, \sigma_p)$ appartenente a $R^{m\times n}$, $p = \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.
I numeri σ_i sono i valori singolari di A.

Una matrice diagonalizzabile è simile a una matrice diagonale. Quando X è una matrice unitaria, diremo che A è unitariamente diagonalizzabile. Bisogna notare che non tutte le matrici sono diagonalizzabili.
Una matrice A di dimensione n è diagonalizzabile se e solo se ammette n autovettori indipendenti. Sia $Ax_i=\beta_i x_i$, i=1,….,n. In forma matriciale questa relazione può essere riscritta come

$$A[x_1 \ldots\ldots\ldots x_n]= [x_1 \ldots\ldots\ldots x_n]\begin{bmatrix} \beta 1 & \ldots & \ldots \\ \vdots & \ddots & \vdots \\ \ldots & \cdots & \beta n \end{bmatrix}$$

ossia AX=XD, con $X=[x_1 \ldots\ldots x_n]$. Se gli autovettori sono indipendenti la matrice X è invertibile e quindi possiamo moltiplicare l'equazione precedente per X^{-1}, ottenendo

$$X^{-1}AX=D$$

Quest'espressione viene detta fattorizzazione spettrale della matrice A.

Teorema di Schur. Per ogni matrice complessa A di dimensione n esiste una matrice unitaria complessa U tale che

$$U^*AU=T=\begin{bmatrix}\beta & * & * \\ & \beta & * \\ & & \beta\end{bmatrix}$$

Essendo β_i, i=1,……,n, gli autovalori di A. Le matrici U e T non sono univocamente determinate. Tale fattorizzazione è detta decomposizione di Schur della matrice A.

Il calcolo di autovalori e autovettori di una matrice è un problema non lineare e quindi molto delicato. Questo problema viene affrontato usando algoritmi che si basano su idee molto differenti. La determinazione degli autovalori è computazionalmente molto onerosa e in alcuni casi porta a delle approssimazioni inaccurate. Per questo motivo spesso ci si accontenta solo di una risposta parziale:

- Determinando autovalori di modulo massimo e/o minimo (autovalori estremali);
- Operando una separazione degli insiemi in modo da individuare degli insiemi che contengono un solo autovalore;
- Localizzando gli autovalori, ossia trovando una regione del piano complesso che contenga tutti gli autovalori (teoremi di Gershgorin);
- Dato un autovalore trovare il corrispondente autovettore;
- Raffinando la stima di un autovalore ottenuta mediante un altro algoritmo;
- Nell'ipotesi che sia noto solo un autovalore trovando una matrice di dimensione n-1 che abbia gli stessi autovalori della matrice iniziale tranne quello noto.

In Matlab autovalori e autovettori sono calcolati con le seguenti funzioni:

- Lambda=eig(A) restituisce gli autovalori di A

```
» A = [1 2 3; 4 9 2; 6 7 6]

A =
     1     2     3
     4     9     2
     6     7     6

» lambda = eig(A)

lambda =
   -1.2336
   13.5147
    3.7189
```

  [V,D]= eig(A) restituisce la matrice V, le cui colonne sono gli autovettori, e la matrice diagonale D, i cui elementi diagonali sono gli autovalori.
  Se A è diagonalizzabile, le matrici V e D costituiscono la sua fattorizzazione spettrale.

```
» [V,D]= eig(A)

V =
   -0.8484    0.2763    0.3854
    0.2394    0.5832   -0.5675
    0.4721    0.7639    0.7276

D =
   -1.2336         0         0
         0   13.5147         0
         0         0    3.7189
```

- [U,T] = schur(A) esegue la fattorizzazione di Schur

```
» [U,T]= schur(A)
```

```
U =
   -0.8484    0.5205    0.0959
    0.2394    0.5390   -0.8076
    0.4721    0.6622    0.5819

T =
   -1.2336    4.0663   -2.3103
    0.0000   13.5147   -4.0160
         0         0    3.7189
```

Ricordiamo che una matrice è hermitiana se coincide con la sua aggiunta (A=A*)
Il teorema di Schur appena enunciato ci permette di dimostrare un'importante proprietà delle matrici hermitiane: ogni matrice hermitiana è unitariamente diagonalizzabile e i suoi autovalori sono reali.
Dire che ogni matrice hermitiana è unitariamente diagonalizzabile equivale ad affermare che i suoi autovettori sono ortonormali. Infatti, se poniamo

$$U=[\, v_1, \ldots\ldots, v_n],$$

con $Av_i=\beta v$, allora

$$U^* U=I$$

Le matrici unitariamente diagonalizzabili sono più numerose delle matrici hermitiane.

Per valutare la stabilità del calcolo è fondamentale il seguente teorema:
Teorema di Bauer-Fike. Supponiamo che A appartenente a C^{nxn} sia diagonalizzabile, cioè che esista una matrice non singolare X tale che

$$X^{-1}AX=D= \mathrm{diag}(\beta_1,\ldots\ldots, \beta_n).$$

Posto che α sia un autovalore di A+E, con E matrice di perturbazione si ha che

$$\min_{(\beta \text{ appartente a } \sigma(A))} |\beta-\alpha| \leq k_2(X)\bullet \|E\|_2$$

dove σ(A) è lo spettro della matrice A e k_2 è il numero di condizionamento in norma 2.
Tale teorema ci consente di dire che se A è una matrice diagonalizzabile allora $k_2(X)$ è il numero di condizionamento del problema agli autovalori.
Il teorema di Bauer-Fike riguarda il condizionamento assoluto però soprattutto in presenza di autovalori molto piccoli si possono commettere errori relativi abbastanza significativi. Esso vale solo per matrici diagonalizzabili. In presenza di autovalori multipli difettivi il problema degli autovalori può essere instabile.
Consideriamo due esempi di autovalori instabili.

Esempio 1
Digitando sulla shell di Matlab

```
» D=diag(ones(10,1))+diag(ones(9,1),1);
» Q=orth(rand(10));
» A=Q*D*Q';
```

La matrice A, così ottenuta, ha un unico autovalore pari a 1 con molteplicità 10. Se calcoliamo gli autovalori di A, con Matlab otteniamo:

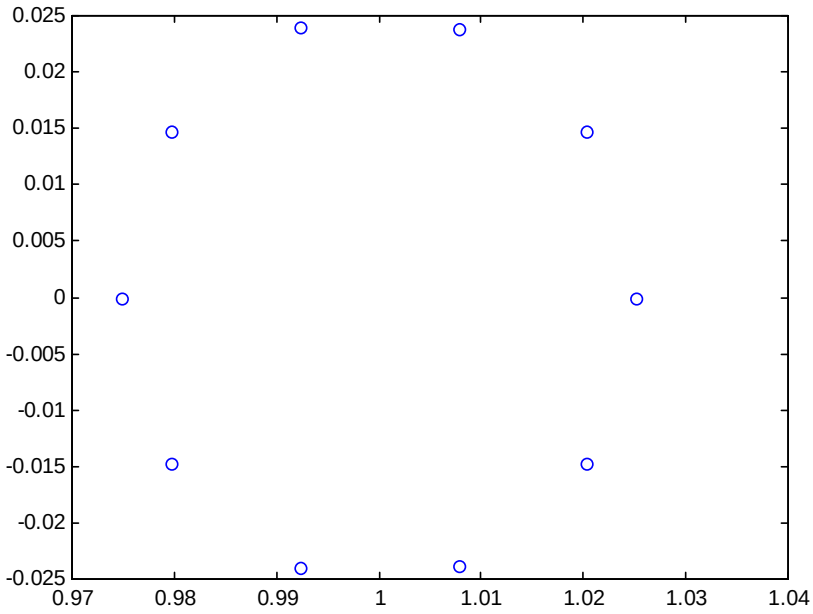
```
» eig(A)

ans =

   0.9737
   0.9787 + 0.0154i
   0.9787 - 0.0154i
```

```
  0.9918 + 0.0250i
  0.9918 - 0.0250i
  1.0081 + 0.0252i
  1.0081 - 0.0252i
  1.0214 + 0.0156i
  1.0214 - 0.0156i
  1.0265
```

Gli autovalori calcolati con Matlab sono affetti da errore e se noi li rappresentiamo graficamente tramite il comando:

```
» plot(eig(A),'o')
```

Notiamo che questi si allargano in un cerchio sul piano complesso.



Il numero di condizionamento di tale matrice lo otteniamo tramite l'istruzione Matlab

```
» [X , D] =eig(A);

»  cond(X)

ans =

  2.4972e+014
```

Esempio 2
Consideriamo adesso la seguente matrice B

```
» B = compan(poly(1:25))
```

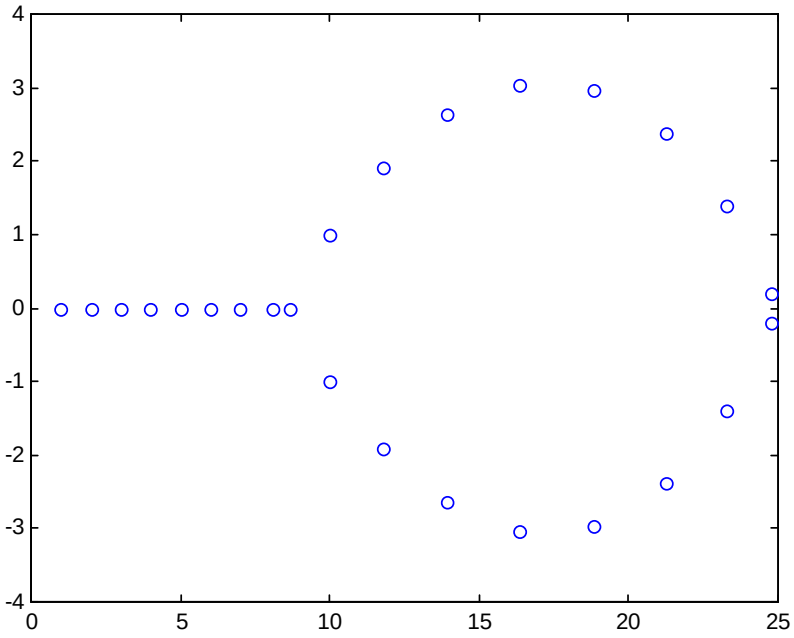
I suoi autovalori sono 1,2....25 ma anche in questo caso se noi li calcoliamo tramite l'ausilio dei comandi Matlab otteniamo:

```
»  eig(B)

ans =

  24.7729 + 0.1993i
  24.7729 - 0.1993i
```

```
  23.2548 + 1.4065i
  23.2548 - 1.4065i
  21.2278 + 2.3812i
  21.2278 - 2.3812i
  18.8483 + 2.9600i
  18.8483 - 2.9600i
  16.3331 + 3.0389i
  16.3331 - 3.0389i
  13.9268 + 2.6390i
  13.9268 - 2.6390i
  11.7939 + 1.9057i
  11.7939 - 1.9057i
   9.9792 + 1.0075i
   9.9792 - 1.0075i
   8.6395
   8.0927
   6.9933
   6.0004
   5.0000
   4.0000
   3.0000
   2.0000
   1.0000
```

```
» [X,D]=eig(A);

» cond(X)

ans =

  3.1995e+026
```

Gli autovalori sono affetti da un forte errore infatti, se rappresentati graficamente, si spostano sul piano complesso come in figura:



Il condizionamento della matrice X degli autovettori, che in base al Teorema di Bauer-Fike, esprime il condizionamento del problema sarà:

In questi due esempi, si nota come il problema degli autovalori, in base al tipo di matrice assegnata può essere estremamente instabile

Algoritmo QR

Se si desiderano tutti gli autovalori di una matrice, bisogna ricorrere a tecniche che consentono dapprima di ridurre la matrice ad una forma più semplice mediante trasformazioni per similitudine pervenendo a una forma triangolare superiore o diagonale: il calcolo degli autovalori diventa così notevolmente semplificato. Questa è la filosofia delle trasformazioni con matrici ortogonali di Householder o Givens. Su tale filosofia si basa infatti il metodo QR e le sue varianti.
Il metodo QR fa uso dei concetti di matrici simili e di decomposizione A=QR è l'algoritmo QR.
Sia *A* una matrice quadrata di ordine *n*. Utilizzando il metodo di Householder è possibile fattorizzare la matrice A come prodotto di due matrici *Q* ed *R* con *Q* unitaria (cioè $Q^TQ=QQ^T = I$) ed *R* triangolare superiore.
Citiamo alcune cose:
1. La matrice *A* ha quale sola particolarità di essere quadrata.
2. Tale fattorizzazione non è unica (i segni delle componenti sulla diagonale della matrice *A* possono essere scelti arbitrariamente).
3. La routine Matlab qr effettua tale fattorizzazione.
4. Se la matrice *H* è simile a *K* allora *H* e *K* hanno gli stessi autovalori. Si può vedere facilmente che la relazione di similitudine è transitiva, cioè se *H*1 è simile ad *H*2 e *H*2 è simile ad *H*3 allora *H*1 è simile ad *H*3.

Lo schema di tale algoritmo è molto semplice. Si parte mettendo $A=A_0$. Nella generica iterazione si effettua la fattorizzazione QR della nuova matrice A_k e si calcola la nuova iterata rimoltiplicando i fattori in ordine inverso:

1. $A_0=A$
2. for k=0,1…fino alla convergenza
    1. fattorizza $A_k=Q_kR_k$
    2. calcola $A^{(k+1)}=R_kQ_k$

Tutte le matrici della successione sono simili e come tali hanno gli stessi autovalori.
Se la matrice A $R^{n\times n}$ è qualsiasi, ma regolare, e con autovalori tutti distinti in modulo tali che

$$|\beta_1| > \ldots\ldots > |\beta_n|$$

allora l'algoritmo QR converge a una matrice T triangolare superiore.
Inoltre essendo $A_{k+1}{}^T= Q_k{}^TA_K{}^TQ_K$, se la matrice A è simmetrica, lo saranno tutte le matrici della successione, con la conseguenza che T coinciderà necessariamente con una matrice diagonale

$$T=D=\mathrm{diag}(\beta_1, \ldots\ldots, \beta_n)$$

L'algoritmo fornisce anche un approssimazione della forma di Schur di A. Infatti

$$A_{k+1}=Q_k{}^TA_kQ_k=Q_kQ_{k-1}{}^T A_{k-1}Q_{k-1}Q_k=\ldots.$$
$$=Q_k{}^T\ldots Q_0{}^TAQ_0\ldots Q_k=U_k{}^TAU_k$$

dove $U_k=Q_0\ldots Q_k$, da cui segue che

$$\lim_{k\to\infty} A_k=T=U^TAU$$

con $U=\prod_{j=0}^{\infty} Q_j$.
Se A è simmetrica si ottiene la sua fattorizzazione spettrale

$$A=UDU^T$$

In questo caso U è la matrice degli autovettori.
Abbiamo visto che l'ipotesi per la convergenza è che tutti gli autovalori siano distinti in modulo.
Quest'ipotesi, apparentemente molto stingente, non crea grandi problemi perché anche quando non è verificata esiste un modo per approssimare comunque gli autovalori.

Supponiamo ad esempio che β=β. In base a tale ipotesi la successione delle matrici A_k converge ad una matrice del tipo

$$\begin{bmatrix} * & * & * & \dots & \dots & * \\ * & * & * & \dots & \dots & * \\ & & \beta 3 & & & \\ & & & \ddots & \vdots & \\ & & & & & \beta n \end{bmatrix}$$

In cui quasi tutto il triangolo inferiore si annulla tranne l'elemento in posizione (2,1).
Il blocco costituito dagli elementi in posizione (1,1), (1,2), (2,1), (2,2), pur non stabilizzandosi ha autovalori che tendono a β e β. Questo ci porta a dire che ogni volta che il metodo non converge è sufficiente calcolare separatamente gli autovalori di ciascuno dei blocchi diagonali che non si stabilizzano. Questo si può fare semplicemente calcolando gli zeri del polinomio caratteristico corrispondente.
Per ridurre il numero di iterazioni nell'applicazione dell'algoritmo QR, e quindi la complessità computazionale, possiamo trasformare la matrice A, mediante una trasformazione di similitudine, in una matrice con strutture di Hessemberg, cioè del tipo

$$A = \begin{bmatrix} * & & * \\ & \cdots & \\ * & & * \\ & \ddots & \vdots \\ & & * \end{bmatrix}$$

Si usa tale metodo perché è invariante per la trasformazione QR: se la matrice a cui viene applicato il metodo QR è di Hessemberg, tali saranno tutte le matrici della successione da esso generata. Questo fatto accelera la convergenza del metodo, in quanto l'algoritmo deve azzerare solo gli n-1 elementi sotto diagonale.
Un altro vantaggio nel trasformare la matrice A in forma di Hessemberg è che si riduce notevolmente il costo computazionale di ogni iterazione del metodo QR.
Per determinare la fattorizzazione QR di una matrice di Hessemberg è sufficiente applicare le n-1 trasformazioni di Givens che annullano gli elementi sottodiagonali.
Per passare in forma di Hessemberg sono necessarie n-2 opportune trasformazioni di Householder che dovranno essere applicate in modo simultaneo a destra e a sinistra di A per garantire la similitudine. Al primo passo indichiamo con a_1 un vettore che contiene tutti gli elementi della prima colonna della matrice $A^{(1)}$=A aventi indice compreso tra 2 e n. H_1 è la matrice elementare di Householder di ordine n-1 che consente di soddisfare l'uguaglianza $H_1 * a_1 = k_1 * e_1$. H_1 è la matrice ortogonale ottenuta orlando H_1 fino a farle raggiungere la dimensione n.
La matrice

$$A^{(2)} := H_1 A^{(1)} H_1 = \begin{bmatrix} * & * & \cdots & * \\ k1 & * & & * \\ \vdots & & \ddots & \vdots \\ 0 & & & * \\ 0 & * & \cdots & * \end{bmatrix}$$

è simile ad A e ha quindi gli stessi autovalori.
Al passo i si ha la seguente matrice

$$A^{(i)} = \begin{bmatrix} * & * & * & & * & * & * \\ k1 & * & * & \cdots & * & * & * \\ * & * & * & & * & * & * \\ & & & \ddots & \vdots & & \\ & ki-1 & & & ** & * & * \\ & & & & ** & * & * \\ & & & & ** & * & * \end{bmatrix}$$

e il vettore individuato dagli elementi con due asterischi , contiene gli elementi della i-esima colonna di $A^{(i)}$ con indice compreso tra i+1 e n.
Se H_i è la matrice di Householder di ordine n-i e H_i è la matrice di dimensione n ottenuta orlando H_i, si ha

$$A^{(i+1)}=H_iA^{(i)}H_i$$

Iterando questo procedimento otteniamo come matrice finale

$$A^{(n-1)}=H_{n-2}.....H_2H_1AH_1H2....H_{n-2}$$

Questa matrice è simile alla matrice iniziale A ed è in forma di Hessemberg: a tale matrice possiamo applicare l'algoritmo QR.
Se la matrice ha tutti autovalori distinti per arrestare l'iterazione dell'algoritmo QR dovremo verificare che gli elementi sotto la diagonale siano inferiori in modulo a una certa tolleranza T

$$|a_{i+1,i}|<T \quad i=1,.....,n-1$$

e che non si superi il numero massimo di interazioni N.
Se A è simmetrica lo sarebbe anche $A^{(n-1)}$. In questo caso la struttura di Hessemberg non può coincidere con quella tridiagonale ed è immediato dimostrare che tale struttura viene conservata da tutte le matrici A(k) generate dall'algoritmo QR.
Vediamo come implementare un programma in Matlab che ci consenta di ricondurre una matrice alla sua forma equivalente di Hessemberg:

```
function [H,Q]=houshess(A)

% trasformazione di una matrice nella forma equivalente di Hessemberg.

n=max(size(A)); Q=eye(n); H=A;
for k=1:(n-2)
[v,beta]=vhouse(H(k+1:n,k));
I=eye(k);
N=zeros(k, n-k);
m=length(v);
R=eye(m)-beta*v*v';
H(k+1:n,k:n)=R*H(k+1:n,k:n);
H(1:n,k+1:n)=H(1:n,k+1:n)*R;
P=[I,N; N',R];
Q=Q*P;
end
```

ove vhouse.m è definito da

```
function [v, beta]= vhouse(x)

% costruzione del vettore di Householder.

n=length(x);
x=x/norm(x);
s=x(2:n)'*x(2:n);
v=[1; x(2:n)];
if (s==0)
beta=0;
else
mu=sqrt(x(1)^2+s);
if (x(1) <= 0)
v(1)=x(1)-mu;
else
v(1)=-s/(x(1)+mu);
```

```
end
beta=2*v(1)^2/(s+v(1)^2);
v=v/v(1);
end
```

quindi QRbasicmethod.m che calcola la matrice triangolare *T* relativa a QR:

```
function [T,hist]=QRbasicmethod(T_input,maxit)

% Metodo QR per una matrice simmetrica tridiagonale "T_input".

T=T_input;
hist=sort(diag(T));
for index=1:maxit
[Q,R]=qr(T);
T=R*Q;                    % nuova matrice equivalente.
hist=[hist sort(diag(T))];        % ripristina gli elementi diagonali dell'indice di interazione
```

Esempio: data la seguente matrice, e sfruttando la precedente implementazione Matlab,

```
» A= [ 2 3 4; 4 5 8; 9 3 11]

A =

    2    3    4

    4    5    8

    9    3   11
```

otteniamo la seguente matrice di Hessemberg:

```
» houshess(A)

ans =

   2.0000    4.8737    1.1169

   9.8489   14.0928   -1.0412

   0.0000    3.9588    1.9072
```

Teoremi di Gershgorin

Come abbiamo precedentemente detto, il calcolo di autovalori e autovettori è un problema molto delicato, sia perché computazionalmente oneroso sia perché in alcuni casi è molto instabile e porta a una forte propagazione degli errori.
Spesso non è indispensabile conoscere tutti gli autovalori ma ci si può accontentare di una risposta parziale consistente, come ad esempio conoscere solo alcuni autovalori o individuare un sottoinsieme del piano complesso C che contenga tutti gli autovalori, o che non li contenga. I risultati di quest'ultimo tipo vengono detti teoremi di localizzazione e un esempio è fornito dal seguente enunciato:
Sia $\|\bullet\|$ una norma consistente. Allora per ogni matrice quadrata A si ha

$$\text{rho}(A) \leq \|A\|$$

Tale teorema afferma che un cerchio con centro nell'origine e raggio pari a una qualunque norma matriciale consistente di una matrice A contiene tutti gli autovalori di A
I teoremi dei cerchi di Gershgorin consentono di localizzare gli autovalori di una matrice in modo più stringente.

Primo teorema di Gershgorin
Data una matrice quadrata A di ordine n, definiamo come cerchi riga:

$$K_i^{(r)} = \{z \in C : |z - a_{i,j}| \leq \sum_{j=1, j \text{ diverso da } i}^{n} |aij| \} \quad i=1....n$$

allora gli autovalori di A sono tutti contenuti nell'unione dei cerchi riga di Gershgorin

$$\sigma(A) \subseteq \bigcup_{i=1}^{n} K_i^{(r)}$$

Data una matrice quadrata A di ordine n, definiamo come cerchi colonna:

$$K_j^{(c)} = \{z \in C : |z - a_{i,j}| \leq \sum_{j=1, j \text{ diverso da } i}^{n} |aij| \} \quad j=1....n$$

allora gli autovalori di A sono tutti contenuti nell'unione dei cerchi colonna di Gershgorin

$$\sigma(A) \subseteq \bigcup_{i=1}^{n} K_i^{(c)}$$

Se indichiamo con C_r e C_c, rispettivamente, l'unione dei cerchi riga e dei cerchi colonna avendo

$$C_r \subseteq \bigcup_{i=1}^{n} K_i^{(r)}$$

$$C_c \subseteq \bigcup_{j=1}^{n} K_j^{(c)}$$

avremo che $\sigma(A) \subseteq C_r \cap C_c$

Consideriamo come esempio la seguente matrice A

$$A = \begin{matrix} 4 & -1 & 1 & 0 & 0 \\ 1 & 3 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 & 8 \end{matrix}$$

i cui autovalori sono $\beta_1 = 5 + \sqrt{10}$, $\beta_2 = \beta_3 = 3$, $\beta_4 = 2$ e $\beta_5 = 5 - \sqrt{10}$.

I cerchi riga sono:
R1 = {z : |z − 4| ≤ 2},
R2 = {z : |z − 3| ≤ 2},
R3 = {z : |z − 1| ≤ 1},
R4 = {z : |z − 2| ≤ 1},
R5 = {z : |z − 8| ≤ 1} .

Quelli colonna sono:

C1 = {z : |z − 4| ≤ 1},
C2 = {z : |z − 3| ≤ 2},
C3 = {z : |z − 1| ≤ 2},
C4 = {z : |z − 2| ≤ 1},
C5 = {z : |z − 8| ≤ 1} .

E' chiaro osservare che gli autovalori stanno nell'insieme:

R2 U R3 U R4 U R5

poiché R2 = C2, R4 incluso in R2; C1,C4 incluso in C2 e R5 = C5.

Secondo teorema di Gershgorin

Se l'unione C_1 di k cerchi di Gershgorin è disgiunta dall'unione C_2 dei rimanenti n − k, cioè se

$C_1 = \bigcup_{i=1}^{k} K_i$ $C_2 = \bigcup_{i=k+1}^{n} K_i$ $C_1 \cap C_2 = 0$

allora k autovalori appartengono a C_1 e n −k appartengono a C_2 ossia i due insiemi sono disgiunti.

Terzo teorema di Gershgorin

Se la matrice di ordine n è irriducibile e un autovalore β sta sulla frontiera dell'unione dei cerchi di Gershgorin, allora sta sulla frontiera di ogni cerchio di Gershgorin K_i, i=1….n.
Questo teorema è utile per dimostrare che le matrici di Poisson sono non singolari.

Ricordiamo che una matrice si dice irriducibile se non è riducibile e che una matrice di ordine n ≥ 2 è riducibile se esiste una matrice di permutazione $\prod$ e un intero k, 0 < k < n, tale che

$$B = \prod A \prod^{T} = \begin{bmatrix} A11 & A12 \\ 0 & A22 \end{bmatrix}$$ in cui $A_{1,1} \in C^{k\times k}$, $A_{12,2} \in C^{(n-k)\times(n-k)}$.

Vediamo ora in Matlab quali sono effettivamente gli autovalori della matrice A:

```
>> A = [15 -2 2; 1 10 -3; -2 1 0]

A =
15 -2 2
1 10 -3
-2 1 0
```

A conferma di quanto stabilito dai primi due teoremi di Gershgorin otteniamo che gli autovalori di A sono:

```
>> eig(A)

ans =
0.5121
14.1026
10.3854
>>
```

Per scrivere una funzione Matlab, che consenta la rappresentazione di tali cerchi di Gershgorin basta selezionare i centri e i raggi per rappresentare poi le circonferenze corrispondenti ai cerchi in modo parametrico

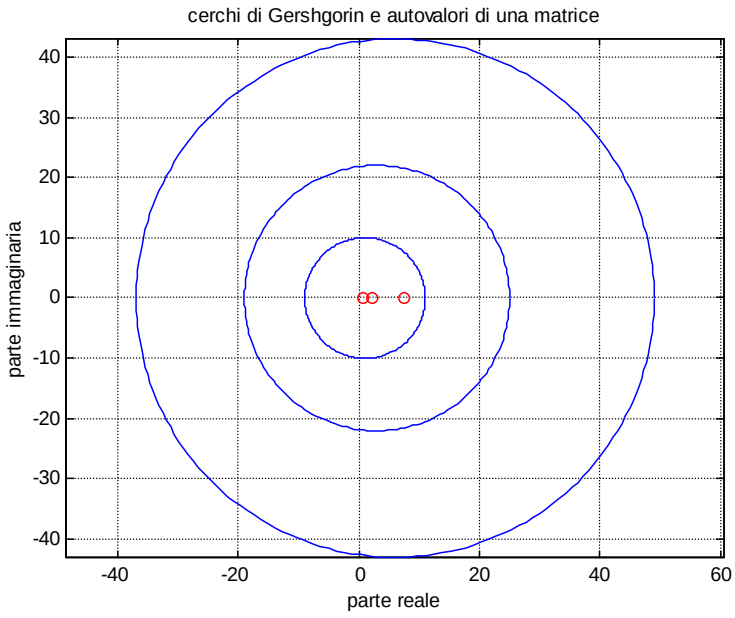
$$x_i(t)=\text{Real}(a_{ii})+r_i\cos(t)$$
$$y_i(t)=\text{Imag}(a_{ii})+r_i\text{sen}(t) \quad t \text{ appartiene } [0,2*\text{pi Greco})$$

Assegnata la matrice

$$A = \begin{matrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{matrix}$$

il listato Matlab che ci consente di rappresentare i cerchi di Gershgorin, ad essa associati, è il seguente:

```
d = diag(A);                        % centri
c_real=real(d);                   % parte reale
c_imag=imag(d);                % parte immaginaria
B=A*diag(d);                      % elementi fuori diagonale
[m,n]= size(A);
r=sum(abs(B'));                  % raggi dei cerchi
t=linspace(0,2*pi,200);  % discretizzazione del grafico
co=cos(t);
si=sin(t);
[v,d]= eig(A);
d = diag(d);                        % autovalori
figure
h=plot(real(d),imag(d), 'or');   % disegna autovalori
hold on;
grid on;
axis equal;
xlabel ('parte reale');
ylabel ('parte immaginaria');
set(h, 'LineWidth',1.5);
    for i=1:n;
        x=c_real(i)+r(i)*co;
        y=c_imag(i)+r(i)*si;
        h2=plot(x,y);
        set(h2,'LineWidth',1,2);
        end
 hold off;
 title('Cerchi di Gorshgorin e autovalori di una matrice');
```



Metodo delle potenze

Il metodo delle potenze ci permette di determinare un autovalore β_1 di modulo massimo di una matrice A e l'autovettore ad esso corrispondente. Questo metodo converge se sono verificate tre ipotesi:

- A è diagonalizzabile;
- L'autovalore di modulo massimo è separato dagli altri ossia
  $|\beta_1|>|\beta_2|>\ldots..>|\beta_n|$;
- Il vettore iniziale x_0 ha una componente non nulla lungo l'autovettore v_1, corrispondente a β_1;

Il metodo delle potenze dato un vettore iniziale x_0 ci consente di calcolare la successione di vettori il cui termine k-esimo è

$$X_k=A_kx_0$$

Essendo A una matrice diagonalizzabile, esiste una base di autovettori e quindi il vettore iniziale può essere visto come una loro combinazione lineare

$$X_0=\sum_{i=1}^{n} \alpha_i v_i$$

dove $Av_i=\beta_i v_i$, i=1,……,n. Sfruttando la relazione precedente si ottiene

$$x_k=A_kx^{(0)}=\sum_{i=1}^{n} \alpha_i A^k v_i = \sum_{i=1}^{n} \alpha_i \beta_i^k v_i$$

applicando la terza ipotesi ($\alpha\neq 0$)

$$x^{(k)}=\alpha_1\beta_1^k v_1 + \sum_{i=2}^{n} \alpha_i\beta_i^k v_i = \alpha_1\beta_1^k(v_1 + \sum_{i=2}^{n} (\alpha_i/\alpha_1)(\beta_i/\beta_1)^k v_i)$$

Per la seconda ipotesi, il rapporto $(\beta_i/\beta_1)^k$ converge a 0 per valori di k tendenti a infinito. Sulla base di ciò, possiamo dire che il vettore x^k tende ad essere parallelo all'autovettore v_1, o meglio a coincidere con esso.
Per aumentare la stabilità del calcolo e diminuire il costo computazionale di ogni iterazione potremo valutare x_k come

$$x_k=Ax^{(k-1)}$$

Tale procedimento porta a due gravi problemi: underflow e overflow: la norma di β^k converge a zero quando $|\beta_1|<1$ e all'infinito quando $|\beta_1|>1$.
Per evitare questi tipi di problemi possiamo normalizzare il vettore x_k ad ogni iterazione, per evitare che la sua norma cresca o decresca eccessivamente, come segue

$$x_k=Aq^{(k-1)}$$

$$q^k=x^k/\|x^k\|$$

Ad ogni passo l'autovalore β_1 può essere approssimato tramite quoziente di Rayleigh

$$\beta_1^k=((q^{(k)})^TAq^{(k)})/((q^{(k)})^Tq^{(k)})$$

Se applichiamo il metodo delle potenze con normalizzazione in norma due, posso trascurare il denominatore.
Dobbiamo inoltre fissare un criterio di stop del tipo

$$|\beta_1^K - \beta_1^{(k-1)}|< T\ |\beta_1^k|$$

dove T è la tolleranza, che dev'essere maggiore di zero, e N è il numero massimo di iterazioni.

L'algoritmo che descrive i metodo delle potenze con normalizzazione in norma 2 è il seguente:

1. Scegli x_0,T e N
2. $q_0=x_0/\|x_0\|_2$
3. k=0;
4. $\beta_0=0$;
5. repeat
   1. k=k+1;
   2. $x_k=Aq^{(k-1)}$
   3. $q^k=x^k/\|x^k\|_2$
   4. $\beta_k=(q^k)^T Aq^k$
6. until $|\beta^k-\beta^{(k-1)}|<T|\beta^k|$ or k>N

x_0 è un vettore iniziale che è conveniente sia costituito da numeri reali casuali per evitare di violare la terza ipotesi.
Se le ipotesi sono verificate β^k converge all'autovalore di modulo massimo e q^k al corrispondente autovettore normalizzato.
Implementiamo in Matlab il metodo delle potenze

```
function [lambda1, x1, niter, err]=power_basic(A,z0,toll,nmax)

% INPUT:
% A : MATRICE DI CUI VOGLIAMO CALCOLARE L'AUTOVALORE DI MASSIMO MODULO.
% z0 : VETTORE INIZIALE (NON NULLO).
% toll: TOLLERANZA.
% nmax: NUMERO MASSIMO DI ITERAZIONI.
%
% OUTPUT:
% lambda1 : VETTORE DELLE APPROSSIMAZIONI DELL'AUTOVALORE DI MASSIMO MODULO.
% x1 : AUTOVETTORE RELATIVO ALL'AUTOVALORE DI MASSIMO MODULO.
% niter : NUMERO DI ITERAZIONI.
% err : VETTORE DEI RESIDUI PESATI RELATIVI A "lambda1".

q=z0/norm(z0);
q2=q;
 err=[];
 lambda1=[];
res=toll+1;
 niter=0;
 z=A*q;
while (res >= toll & niter <= nmax)
q=z/norm(z);
z=A*q;
lam=q'*z;
 x1=q;
z2=q2'*A;
q2=z2/norm(z2);
 q2=q2'; y1=q2;
 costheta=abs(y1'*x1);
niter=niter+1;
 res=norm(z-lam*q)/costheta;
err=[err; res];
lambda1=[lambda1; lam];
end
```

Qualche nota:

1. il vettore iniziale z_0 e' normalizzato ed in err, lambda1 vengono memorizzati rispettivamente i valori dell'errore compiuto e dell'autovalore di massimo modulo β_{max};
2. l'assegnazione res=toll+1; forza l'algoritmo ad entrare nel ciclo while,mentre z=A*q; è una quantità da utilizzarsi per il calcolo dell'autovalore β_{max};
3. nel ciclo while, q è un'approssimazione di un autoversore relativo a β_{max}, mentre lam di β_{max};
4. il ciclo si interrompe se un numero massimo di iterazioni niter è raggiunto oppure $(\|Aq_k-\beta_k\|_2/|\cos(\theta\beta_k)|)<$ tol

dove $\theta\beta_k$ è l'angolo formato tra (un'approssimazione del)l'autovalore destro x1 e sinistro y1 associati a lam.

Esempio 1
Testiamo il codice relativamente al calcolo dell'autovalore di massimo modulo di

$$A = \begin{matrix} -15.5 & 7.5 & 1.5 \\ -51 & 25 & 3 \\ -25.5 & 7.5 & 11.5 \end{matrix}$$

$$= \begin{matrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 7 & 9 & 3 \end{matrix} * \begin{matrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{matrix} * \begin{matrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 7 & 9 & 3 \end{matrix}^{-1}$$

La matrice A è diagonalizzabile e ha autovalori β_1=10, β_2=10, β_3=1.
Si può vedere che una base di autovettori relativa agli autovalori 10, 10, 1 è composta da (1,2,7), (2,5,9), (3,6,3). Quale vettore iniziale del metodo delle potenze consideriamo z_0 = (1,1,1) = (7/6) · (1,2,7)−1· (2,5,9)+(11/18) · (3,6,3) e quindi il metodo delle potenze applicato ad A, e avente quale punto iniziale z_0 può essere utilizzato per il calcolo dell'autovalore di massimo modulo di A, poiché $a_1 = 7/6 \neq 0$.
Dalla shell di Matlab

```
>> S=[1 2 3; 2 5 6; 7 9 3];
>> D=diag([10 10 1]);
>> A=S*D*inv(S)

A =
-15.5000 7.5000 1.5000
-51.0000 25.0000 3.0000
-25.5000 7.5000 11.5000

>> z0=[1 1 1]';
>> toll=10^(-8);
>> nmax=10;
>> format short e;
>> [lambda1, x1, niter, err]=power_basic(A,z0,toll,nmax)

lambda1 =
1.1587e+001
1.0138e+001
1.0014e+001
1.0001e+001
1.0000e+001
1.0000e+001
1.0000e+001
1.0000e+001
1.0000e+001
1.0000e+001

x1 =
-2.8583e-001
-9.1466e-001
-2.8583e-001

niter =
10

err =
2.2466e+000
2.1028e-001
2.0934e-002
```

```
2.0925e-003
2.0924e-004
2.0924e-005
2.0924e-006
2.0924e-007
2.0924e-008
2.0924e-009
>>
```

Dall'analisi della quantità err, che consiste in un particolare residuo pesato, si nota chela convergenza è abbastanza veloce.
Una questione sorge spontanea: cosa sarebbe successo se avessimo utilizzato l'algoritmo senza normalizzazione ossia se al posto della funzione power_basic avessimo usato quella power_method definita come segue:

```
function [lambda,v]=power_method(A,x0,maxit)
v=x0;
 for index=1:maxit
v_old=v;
v=A*v_old;
lambda=(v_old'*v)/(v_old'*v_old);
end
```

Proviamo il test, facendo iterare il metodo prima 5, poi 100 volte e alla fine 1000 volte(si noti il settaggio della variabile maxit relativa al numero di iterazioni da compiere):

```
>> x0=[1 1 1]'

x0 =
1
1
1

>> A=[-15.5 7.5 1.5; -51 25 3; -25.5 7.5 11.5]

A =
-15.5000 7.5000 1.5000
-51.0000 25.0000 3.0000
-25.5000 7.5000 11.5000

>> [lambda,v]=power_method(A,x0,5)

lambda =
10.0014

v =
1.0e+005 *
-0.8333
-2.6666
-0.8333

>> [lambda,v]=power_method(A,x0,100)

lambda =
10.0000

v =
1.0e+100 *
-0.8333
-2.6667
-0.8333

>> [lambda,v]=power_method(A,x0,1000)

lambda =
```

```
NaN

v =
NaN
NaN
NaN
```

La ragione di tali risultati è facilmente spiegabile. Per k relativamente piccolo si ha $A \cdot t_k \approx 10 \cdot t_k$ e quindi per $s \geq k$ $t_s \approx A^{s-k} \cdot t_k \approx 10 \cdot A^{s-k-1} \cdot t_k \approx \cdots \approx 10^{s-k} \cdot t_k$ da cui $\|t_s\|_2 \approx 10^{(s-k)} \|t_s\|_2$: questo causa problemi di overflow nell'applicazione dell'algoritmo di base.

Esempio 2
Proviamo un test diverso, questa volta con la matrice (diagonalizzabile)

$$A = \begin{matrix} 1 & 2 \\ 0 & -1 \end{matrix}$$

avente autovalori $\beta_1 = 1$ e $\beta_2 = -1$ e autovettori linearmente indipendenti $(1, 0),(-1, 1)$. Quale vettore iniziale poniamo $x_0 = (1,3) = 4 \cdot (1,0)+3 \cdot (-1,1)$ e quindi il metodo delle potenze , partendo da x_0, può essere sicuramente applicato alla matrice A.
Non è detto che il metodo converga in quanto, pur essendo $\beta_1 \neq \beta_2$, si ha $|\beta_1| = |\beta_2| = 1$.

Dalla shell di Matlab:

```
>> A=[  1 2; 0 -1]

A =
1 2
0 -1

>> [lambda1, x1, niter, err]=power_basic(A,[1; 3],10^(-8),15)

lambda1 =
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001
-3.4483e-002
-2.0000e-001

x1 =
3.1623e-001
9.4868e-001

niter =
16

err =
4.4567e-001
2.4000e+000
4.4567e-001
2.4000e+000
4.4567e-001
2.4000e+000
4.4567e-001
```

```
2.4000e+000
4.4567e-001
2.4000e+000
4.4567e-001
2.4000e+000
4.4567e-001
2.4000e+000
4.4567e-001
2.4000e+000
>>
```

Dall'analisi del residuo pesato è chiaro che il metodo non converge, e come anticipato il motivo è la presenza di autovalori distinti aventi modulo massimo identico.

Esempio 3
Per terminare, vediamo il caso della matrice diagonalizzabile (avente autovalori distinti)

$$A = \begin{matrix} 1 & 2 \\ 0 & 10 \end{matrix}$$

in cui il metodo funziona rapidamente, in quanto esiste un solo autovalore di modulo massimo, uguale a 10.

```
>> A=[1 2; 0 10]

A =
1 2
0 10

>> [lambda1, x1, niter, err]=power_basic(A,[1; 3],10^(-8),15)

lambda1 =
9.9779e+000
9.9979e+000
9.9998e+000
1.0000e+001
1.0000e+001
1.0000e+001
1.0000e+001
1.0000e+001

x1 =
2.1693e-001
9.7619e-001

niter =
8

err =
9.6726e-002
9.7529e-003
9.7610e-004
9.7618e-005
9.7619e-006
9.7619e-007
9.7619e-008
9.7619e-009
```

Si osservi che il metodo termina in quanto l'errore pesato err è minore della tolleranza
tol l = 10−8.

Metodo delle potenze inverse

Il metodo delle potenze inverse è un metodo utile per determinare l'autovalore di modulo minimo di una matrice non singolare.
Essendo $\sigma(A)= \{\beta_1,....,\beta_n\} \quad \Longrightarrow \quad \sigma(A^{-1})=\{\beta_1^{-1},.....,\beta_n^{-1}\}$, si può applicare l'algoritmo alla matrice inversa A^{-1} per approssimare il suo autovalore di modulo massimo β_n^{-1}. Quest'approssimazione è fattibile sulla base di alcune ipotesi:

1. A dev'essere diagonalizzabile;
2. X_0 deve avere componente non nulla rispetto a v_n
3. $1/|\beta_i| > 1/|\beta_n|$ con i=1,.....,n-1

Dobbiamo risolvere il sistema $Ax^k=x^{(k-1)}$. Per ridurre la complessità computazionale è conveniente fattorizzare la matrice A all'inizio del processo e risolvere ad ogni passo due sistemi triangolari.
L'algoritmo che ci descrive il metodo delle potenze inverse è il seguente:

1. Scegli x_0, T e N
2. $q^{(0)}=x^{(0)}/\|x^{(0)}\|_2$
3. fattorizza A=LU
4. k=0
5. $\beta^{(0)}=0$
6. repeat
   1. k=k+1
   2. risolvi $Ly=q^{(k-1)}$
   3. risolvi $Ux^k=y$
   4. $q^k=x^k/\|x^k\|_2$
   5. $\beta^k=(\beta^k)^T Aq^k$
7. until $|\beta^k-\beta^{(k-1)}<T|\beta^k|$ or k>N

se sono verificate le ipotesi di convergenza β^k tende a β_n^{-1} per k che tende a infinito.
Ricordiamo che se β è autovalore di *A* allora Ax = βx ----> $(A-\mu I)x = \beta x - \mu x = (\beta-\mu)x$ e quindi $\beta-\mu$ è autovalore di $A-\mu I$.
Il metodo delle potenze inverse applicato a $A-\mu I$ calcola il minimo autovalore $\sigma = \beta-\mu$ in modulo di $A-\mu I$ cioè il σ che rende minimo il valore di $|\sigma| = |\beta_i - \mu|$, dove β_i sono gli autovalori di A. Quindi essendo $\beta_i = \sigma_i - \mu$ si ottiene pure il β_i più vicino a μ.
Una possibile implementazione del metodo delle potenze inverse in Matlab è la seguente:

```
function [lambda, x, niter, err]=invpower(A,z0,mu,toll,nmax)

% DATO UN VALORE mu, SI CALCOLA L'AUTOVALORE "lambda_mu" PIU' VICINO A mu.
% INPUT:
% A : MATRICE DI CUI VOGLIAMO CALCOLARE L'AUTOVALORE "lambda_mu".
% z0 : VETTORE INIZIALE (NON NULLO).
% mu : VALORE DI CUI VOGLIAMO CALCOLARE L'AUTOVALORE PIU' VICINO.
% toll: TOLLERANZA.
% nmax: NUMERO MASSIMO DI ITERAZIONI.
%
% OUTPUT:
% lambda : VETTORE DELLE APPROSSIMAZIONI DELL'AUTOVALORE DI MINIMO MODULO.
% x : AUTOVETTORE RELATIVO ALL'AUTOVALORE DI MINIMO MODULO.
% niter : NUMERO DI ITERAZIONI.
% err : VETTORE DEI RESIDUI PESATI RELATIVI A "lambda".
%

n=max(size(A));
M=A-mu*eye(n);
 [L,U,P]=lu(M);
q=z0/norm(z0);
 q2=q';
 err=[];
lambda=[];
res=toll+1;
```

```
 niter=0;
while (res >= toll & niter <= nmax)
niter=niter+1;
b=P*q;
y=L\b;
 z=U\y;
q=z/norm(z);
 z=A*q;
lam=q'*z;
b=q2';
 y=U'\b;
 w=L'\y;
q2=(P'*w)';
q2=q2/norm(q2);
costheta=abs(q2*q);
if (costheta > 5e-2)
res=norm(z-lam*q)/costheta;
 err=[err; res];
 lambda=[lambda; lam];
else
disp('\n \t [ATTENZIONE]: AUTOVALORE MULTIPLO');
 break;
end
x=q;
```

Forniamo ora alcune spiegazioni del codice in invpower.
1. Per risolvere il sistema lineare si effettua una fattorizzazione PM = LU della matrice M = A−μI ;
2. All'interno del ciclo while, nella prima riga si calcola z_k , mentre nella successiva un suo versore q_k , e σ_k è immagazzinato in lam;
3. Similmente al metodo diretto si effettua il prodotto scalare di un'autovalore sinistro con uno destro.

Applichiamo il metodo delle potenze inverse per il calcolo dell'autovalore più piccolo in modulo della matrice

$$A = \begin{matrix} -15.5 & 7.5 & 1.5 \\ -51 & 25 & 3 \\ -25.5 & 7.5 & 11.5 \end{matrix}$$

$$= \begin{matrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 7 & 9 & 3 \end{matrix} * \begin{matrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{matrix} * \begin{matrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 7 & 9 & 3 \end{matrix}^{-1}$$

Come visto la matrice A è quindi diagonalizzabile, ha autovalori 10, 10, 1 e relativi autovettori è (1,2,7), (2,5,9), (3,6,3) formanti una base di R3.
Quale vettore iniziale del metodo delle potenze consideriamo z_0 = (1,1,1) = (7/6) · (1,2,7)−1· (2,5,9)+(11/18) · (3,6,3) e quindi il metodo delle potenze inverse applicato ad A, e avente quale punto iniziale z_0 può essere utilizzato per il calcolo dell'autovalore di minimo modulo di A.

```
» z0=[1;1;1];
» mu=0;
» toll=10^(-8);
» nmax=10;

» A=[-15.5 7.5 1.5; -51 25 3; -25.5 7.5 11.5]

A =
-15.50000000000000 7.50000000000000 1.50000000000000
-51.00000000000000 25.00000000000000 3.00000000000000
-25.50000000000000 7.50000000000000 11.50000000000000
```

```
» [lambda, x, niter, err]=invpower(A,z0,mu,toll,nmax)

lambda =
0.39016115351993
0.94237563941268
0.99426922936854
0.99942723776656
0.99994272692315
0.99999427272378
0.99999942727270
0.99999994272728
0.99999999427273

x =
0.40824829053809
0.81649658085350
0.40824829053809

niter =
9

err =
0.81535216507377
0.08358101289062
0.00838126258396
0.00083836078891
0.00008383842712
0.00000838386620
0.00000083838685
0.00000008383868
0.00000000838387
>>
```

La convergenza è lineare (come si intuisce dalle approssimazioni contenute nel vettore lambda). Per vederlo, dalla shell di Matlab calcoliamo l'errore assoluto/relativo relativo all'autovalore 1:

```
>> s=1-lambda
s =
0.60983884648007
0.05762436058732
0.00573077063146
0.00057276223344
0.00005727307685
0.00000572727622
0.00000057272730
0.00000005727272
0.00000000572727
```