Serie 3 - Soluzione
Autovalori e Autovettori

Esercizio 1

Si ricorda che gli elementi $a_{i,j}$ della matrice di connessione $A \in \mathbb{R}^{N\times N}$ godono della seguente proprietà:

$$\sum_{i=1}^{N} a_{i,j} = 1. \tag{1}$$

1. Consideriamo i seguenti comandi Matlab® :

```
A=randi([0, 1],100,100);
s=sum(A);
for j=1:size(A,1)
  A(j,:)=A(j,:)./s;
end
```

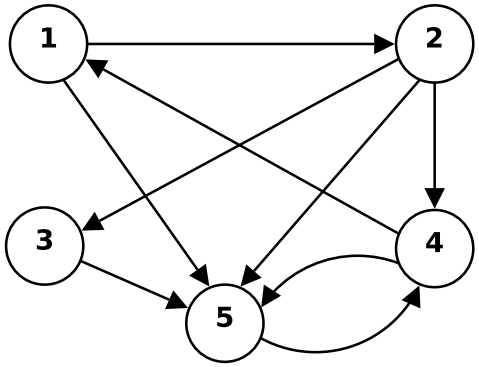

Figura 1: Schema di un web di 5 pagine. Ogni cerchio rappresenta una pagina web, ogni freccia un link.

2. La matrice si scrive facilmente se si procede pagina per pagina, cioè colonna per colonna:

```
B=[
  0   0   0   1   0
  1   0   0   0   0
  0   1   0   0   0
  0   1   0   0   1
  1   1   1   1   0];
s=sum(B);
for j=1:size(B,1)
  B(j,:)=B(j,:)./s;
end
```

3. Una possibile implementazione è la seguente:

```
function [lambda,x,iter]=eigpower(A,tol,nmax,x0)
%EIGPOWER    Approssima l'autovalore di modulo massimo
%            di una matrice.
%   LAMBDA = EIGPOWER(A) calcola con il metodo delle
%   potenze l'autovalore di una matrice A di modulo
%   massimo a partire da un dato iniziale pari al
%   vettore unitario.
%   LAMBDA = EIGPOWER(A,TOL,NMAX,X0) arresta il metodo
%   quando la differenza  fra due iterate consecutive
%   e' minore di TOL (il valore di default e' 1.E-06)
%   o quando il massimo numero di iterazioni NMAX (il
%   valore di default e' 100) e' stato raggiunto.
%   [LAMBDA,X,ITER] = EIGPOWER(A,TOL,NMAX,X0)
%   restituisce anche l'autovettore unitario X tale
%   che A*X=LAMBDA*X ed il numero di iterazioni
%   effettuate per calcolare X.

[n,m] = size(A);
if n ~= m
    error('Solo per matrici quadrate');
end

if nargin == 1
   tol = 1.e-06;   x0 = ones(n,1);   nmax = 100;
end

% iterazione zero fuori dal ciclo while
iter = 0;
y = x0/norm(x0); % y0
lambda = y'*A*y; % lambda0
err = tol*abs(lambda) + 1; % dobbiamo entrare nel ciclo

while err > tol * abs(lambda) && abs(lambda) ~= 0 && iter<nmax
   iter = iter + 1; % iter=1,2,3,...
   x = A*y;
   y= x/norm(x);
   lambdanew = y'*A*y;
   err = abs(lambdanew - lambda);
   lambda = lambdanew;
end

if (err <= tol*abs(lambda))
     fprintf('eigpower converge in %d iterazioni \n', iter);
else
     fprintf('eigpower non converge in %d iterazioni. \n', iter)
end

return
```

4. Consideriamo i seguenti comandi Matlab® :

```
[lA,xA,iterA]=eigpower(A,1.e-06,100,1/100*ones(100,1));
[lB,xB,iterB]=eigpower(B,1.e-06,100,1/5*ones(5,1));
lA
lB
xB
```

Queste istruzioni producono il seguente output su schermo:

```
Il metodo delle potenze converge in 5 iterazioni

lA =

    1.0000

Il metodo delle potenze converge in 23 iterazioni

lB =

    1.0000

xB =

    0.3402
    0.1701
    0.0567
    0.6804
    0.6238
```

È interessante analizzare il vettore `xB`, cioè il *PageRank* di un web composto dalle 5 pagine di Fig. 1:

- come ci si potrebbe aspettare, la pagina 5 ha un ranking relativamente alto (0.6238), perché riceve un link da tutte le altre pagine;
- la pagina 4 ha il ranking maggiore (0.6804), nonostante riceva solo due link, infatti può godere interamente del ranking dato dal link proveniente da 5, più un terzo del ranking di 2;
- la pagina 1 riceve un solo link, proveniente da 4, e guadagna un punteggio (0.3402) pari a metà del ranking di 4;
- anche la pagina 2 riceve un solo link, proveniente da 1, ed ha un ranking (0.1701) pari alla metà di quello di 1, cioè un quarto di quello di 4;
- infine la pagina 3 riceve un solo link da 2 e ne eredita un terzo del ranking (0.0567).

I *PageRank*, cioè gli elementi dell'autovettore, sono compresi tra zero e uno; si sottolinea che i valori pubblicati da Google sono in realtà valori da 1 a 10: i punteggi infatti sono riportati in una opportuna scala logaritmica. Ad esempio `www.mate.polimi.it` ha attualmente un *PageRank* pari a 6, `www.gnu.org/software/octave/` 7, `www.mathworks.com` e `kernel.org` 8, `www.nytimes.com` 9, `google.com` 10.

Esercizio 2

1. Si consideri la seguente implementazione:

```
function [mu,x,iter]=invpower(A,tol,nmax,x0)
%INVPOWER    Approssima l'autovalore di modulo minimo
%            di una matrice.
%   [MU,X,ITER] = INVPOWER(A, TOL, NMAX, X0)
%   calcola con il metodo delle potenze inverse l'autovalore
%   di una matrice A di modulo minimo a partire da un dato
%   iniziale pari al vettore unitario, arrestando il metodo
%   quando la differenza  fra due iterate consecutive
%   e' minore di TOL (il valore di default e' 1.E-06)
%   o quando il massimo numero di iterazioni NMAX (il
%   valore di default e' 100) e' stato raggiunto.
%   La funzione restituisce anche l'autovettore unitario
%   X tale che A*X=MU*X ed il numero di iterazioni
%   effettuate per calcolare X.

[n,m] = size(A);
if n ~= m, error('Solo per matrici quadrate'); end
if nargin == 1
   tol = 1.e-06;   x0 = ones(n,1);   nmax = 100;
end

% calcolo la fattorizzazione LU una volta per tutte
[L,U,P]=lu(A);

% iterazione zero fuori dal ciclo while
iter = 0;
y = x0/norm(x0); % y0
mu = y'*A*y; % mu0
err = tol*abs(mu) + 1; % dobbiamo entrare nel ciclo

while err > tol*abs(mu) && abs(mu) ~= 0 && iter<nmax
   iter = iter + 1; % iter=1,2,3,...
   % risolvo Ax^{(k)}=y^{(k-1)}
   z=fwsub(L,P*y);
   x=bksub(U,z);
   y= x/norm(x)
   munew = y'*A*y;
   err = abs(munew - mu);
   mu = munew;
end

if (err <= tol*abs(mu))
     fprintf('invpower converge in %d iterazioni \n', iter);
else
     fprintf('invpower non converge in %d iterazioni \n', iter)
end

return
```

2. Le istruzioni

```
A = toeplitz (1:4);
```

```
invpower (A, 1e−6, 100, [1 2 3 4]')
```

danno come output

```
invpower converge in 14 iterazioni

ans =

   -0.5858
```

Chiamando la funzione con input $\mathbf{x}^{(0)} = (1, 1, 1, 1)^T$ otteniamo invece

```
>> invpower (A, 1e-6, 100, ones (4, 1))
invpower converge in 6 iterazioni

ans =

   -1.0990
```

L'autovalore trovato non è quello di modulo minimo all'interno dello spettro di A. Questo è dovuto al fatto che il vettore iniziale $\mathbf{x}^{(0)}$ è ortogonale all'autovettore relativo all'autovalore di modulo minimo (si veda l'ananlisi di convergenza del metodo delle potenze sul libro di testo per maggiori dettagli). L'autovalore trovato è quindi quello di modulo minimo tra quelli che hanno autovettori non ortogonali a $\mathbf{x}^{(0)}$.

Consideriamo ora la funzione `invpower_mod` (allegata) che esegue un dato numero di iterazioni (scelto dall'utente) del metodo delle potenze inverse. Detta $\mathbf{y}^{(i)}$ l'approssimazione dell'autovettore ottenuta nell'i-esima iterazione, `invpower_mod` calcola anche il valore assoluto della componente di $\mathbf{y}^{(i)}$ lungo la direzione definita dall'autovettore associato al minimo autovalore (in modulo) della matrice in input. Chiamiamo $\mathbf{x}_n$ questo autovettore. La lista delle componenti calcolate in ciascuna iterazione è contenuta nell'output `min_eigvec_comp`.

```
>> [lambda, x, min_eigenvec_comp] = invpower_mod (A, 100, ones (4, 1));
>> lambda
lambda =

   -0.5858
```

Vediamo che l'autovalore calcolato è effettivamente quello più piccolo in modulo. Disegnando il grafico in scala semilogaritmica di `min_eigvec_comp` in funzione del numero di iterazioni (vedi Figura 2), osserviamo una curva che tende a 1. Questo significa che gli y_i calcolati si stanno allineando a $\mathbf{x}_n$. Il motivo alla base di ciò è il seguente: se si sceglie come dato iniziale $(1, 1, 1, 1)^{\mathrm{T}}$, che è ortogonale a $\mathbf{x}_n$, si ottiene un'iniziale convergenza all'autovettore associato a -1.0990. Questo è ciò che ci aspetteremmo anche dalla teoria. Se però si forza il metodo ad eseguire più iterazioni, gli errori numerici fanno sì che si generi negli $\mathbf{y}^{(i)}$ una componente nella direzione di $\mathbf{x}_n$ che porta rapidamente ad una convergenza a $\mathbf{x}_n$ stesso.

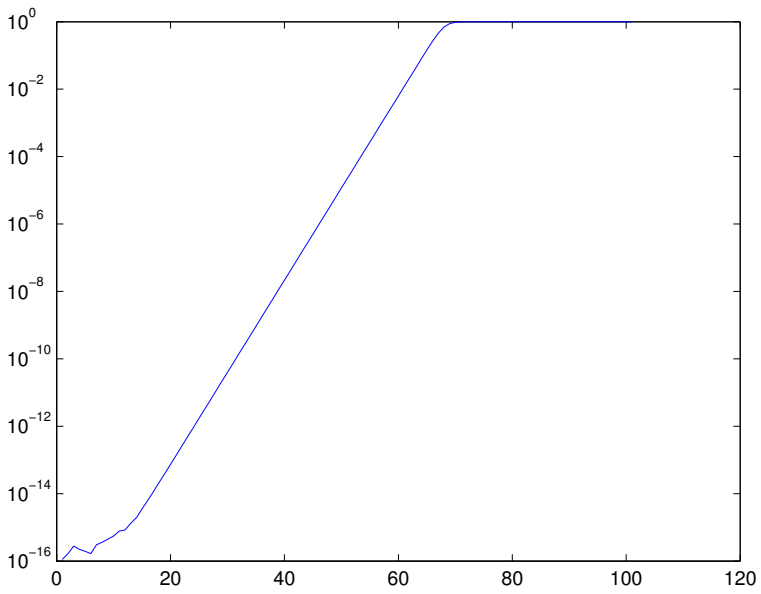

Figura 2: Andamento di $\dfrac{\mathbf{y}^{(i)} \cdot \mathbf{x}_n}{\|\mathbf{x}_n\|^2}$

Esercizio 3

1. I comandi

```
>> B = [10 −1 1 0; 1 1 −1 3; 2 0 2 −1; 3 0 1 5]
>> gershcircles(B)
```

generano, in due grafici distinti, le zone del piano determinate dall'unione dei cerchi riga $\mathcal{S}_\mathcal{R}$ e colonna $\mathcal{S}_\mathcal{C}$ della matrice B (Figura 3). Vengono inoltre disegnate, in un terzo grafico (si veda l'output in Figura 4), le due zone del piano $\mathcal{S}_\mathcal{R}$ e $\mathcal{S}_\mathcal{C}$, cosa che rende più agevole l'individuazione di $\mathcal{S}_\mathcal{R} \bigcap \mathcal{S}_\mathcal{C}$. In referimento alla Figura 4, lo studio dei cerchi di Gershgorin non fornisce nessun limite inferiore per l'autovalore di modulo minimo, infatti il punto $\{0, 0\}$ del piano complesso è compreso nella zona $\mathcal{S}_\mathcal{R} \bigcap \mathcal{S}_\mathcal{C}$.
Si ricava invece qualche informazione sull'autovalore di modulo massimo: dato che non potrà essere localizzato esternamente al cerchio riga di centro $\{10, 0\}$ e raggio 2, possiamo dedurre che sicuramente $|\lambda_{\max}(B)| \leq 12$.

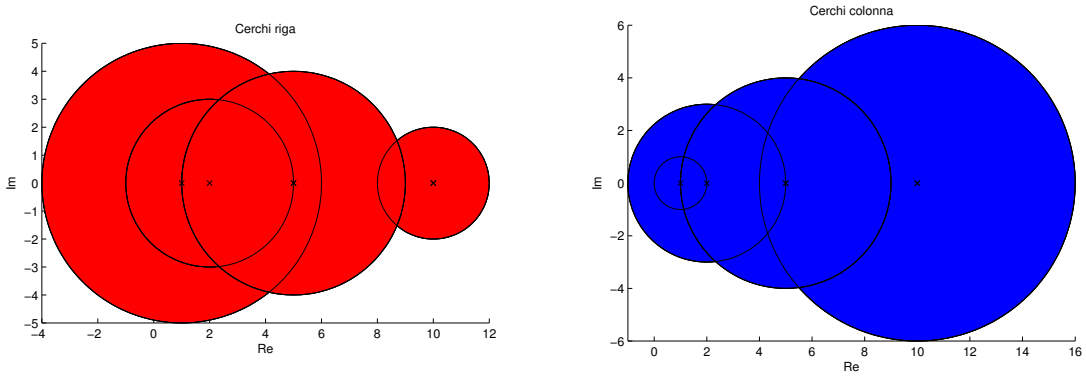

Figura 3: Cerchi riga e colonna relativi alla matrice B.

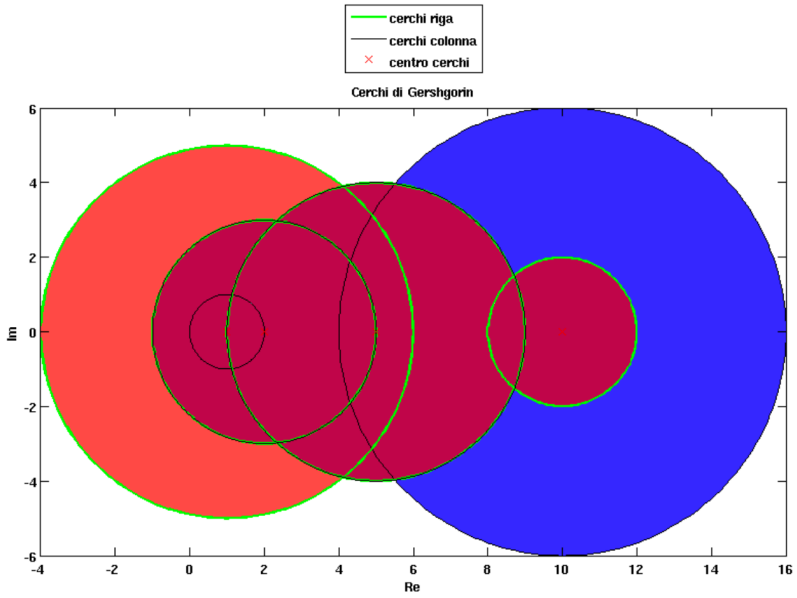

Figura 4: Cerchi di Gershgorin relativi alla matrice B. I colori sono aggiunti a mano, in modo da evidenziare la regione $\mathcal{S}_{\mathcal{R}} \bigcap \mathcal{S}_{\mathcal{C}}$ (colore viola).

2. Il comando

```
>> eig(B)
ans =
   9.8567
   5.4415
   1.3509 + 0.4554i
   1.3509 - 0.4554i
```

restituisce gli autovalori della matrice B. Notiamo che siamo in presenza di due autovalori complessi coniugati di modulo minimo.

3. Dato che gli autovalori di modulo minimo sono complessi coniugati, la convergenza del metodo delle potenze inverse non è assicurata. Verifichiamo con il comando:

```
>> [lambda,x,iter]=invpower(B,1e-6,1000,ones(4,1));
Il metodo delle potenze inverse non converge in 1000 iterazioni.
```

   Ad analoga conclusione si perviene anche se si aumenta il numero massimo di iterazioni.

4. La funzione è una immediata modifica della `function invpower`: è infatti necessario aggiungere un valore in input alla funzione che rappresenta il valore dello *shift*, e creare la matrice $M = A - \mu I$, che sarà fattorizzata e utilizzata per la risoluzione dei sistemi lineari per il primo passo dell'algoritmo.
   Si noti che il quoziente di Rayleigh $\lambda^{(k)} = (\mathbf{y}^{(k)})^H A\mathbf{y}^{(k)}$ viene sempre calcolato con la matrice A, in modo che l'algoritmo converga direttamente all'autovalore $\lambda_\mu(A)$.

```
function [lambda,x,iter]=invpowershift(A,mu,tol,nmax,x0)
%   INVPOWERSHIFT Approssima l'autovalore di una matrice
%                 piu' vicino al numero (complesso) mu
%   LAMBDA = INVPOWERSHIFT(A,MU) calcola con il metodo delle
%   potenze inverse con shift l'autovalore di una matrice A
%   piu vicino a mu a partire da un dato iniziale pari
%   al vettore unitario.
%   LAMBDA = INVPOWERSHIFT(A,MU,TOL,NMAX,X0) arresta il metodo
%   quando la differenza  fra due iterate consecutive
%   e' minore di TOL (il valore di default e' 1.E-06)
%   o quando il massimo numero di iterazioni NMAX (il
%   valore di default e' 100) e' stato raggiunto.
%   [LAMBDA,X,ITER] = INVPOWERSHIFT(A,TOL,NMAX,X0)
%   restituisce anche l'autovettore unitario X tale
%   che A*X=LAMBDA*X ed il numero di iterazioni
%   effettuate per calcolare X.

[n,m] = size(A);
if n ~= m, error('Solo per matrici quadrate'); end
if nargin == 2
   tol = 1.e-06;   x0 = ones(n,1);   nmax = 100;
end
M = A - mu*eye(n);
% calcolo la fattorizzazione LU una volta per tutte
[L,U,P]=lu(M);
% iterazione zero fuori dal ciclo while
iter = 0;
y = x0/norm(x0); % y0
lambda = y'*A*y; % lambda0
err = tol*abs(lambda) + 1; % dobbiamo entrare nel ciclo

while (err>tol*abs(lambda)) && (abs(lambda)~=0) && (iter<nmax)
   iter = iter + 1; % iter=1,2,3,...
   % risolvo Mx^{(k)}=y^{(k-1)}
   z=fwsub(L,P*y);
   x=bksub(U,z);
```

```
    y= x/norm(x);
    lambdanew = y'*A*y;
    err = abs(lambdanew - lambda);
    lambda = lambdanew;
end
if (iter < nmax)
      fprintf(['Il metodo delle potenze inverse con shift converge '...
               'in %d iterazioni all''autovalore \n'], iter);
      lambda
else
      fprintf(['Il metodo delle potenze inverse con shift non converge '...
               'in %d iterazioni. \n'], iter);
end
return
```

Per ottenere la convergenza agli autovalori di modulo minimo, che sono complessi coniugati, è necessario fornire un valore complesso di *shift*. Ad esempio, fornendo $\mu = i$, l'algoritmo converge correttamente all'autovalore con parte immaginaria positiva; viceversa, fornendo lo shift $\mu = -i$, l'algoritmo converge all'autovalore con parte immaginaria negativa:

```
>> [lambda,x,iter]=invpowershift(B,i,1e-6,1000,ones(4,1));
Il metodo delle potenze inverse con shift converge in 42 iterazioni
all'autovalore lambda = 1.3509 + 0.4554i

>> [lambda,x,iter]=invpowershift(B,-i,1e-6,1000,ones(4,1));
Il metodo delle potenze inverse con shift converge in 42 iterazioni
all'autovalore lambda = 1.3509 - 0.4554i
```

Dall'analisi dei cerchi di Gershgorin avevamo dedotto che $|\lambda_{\max}(B)| \leq 12$; fornendo il valore $\mu = 12$ come *shift* per il metodo si converge all'autovalore di modulo massimo:

```
>> [lambda,x,iter]=invpowershift(B,12,1e-6,1000,ones(4,1));
Il metodo delle potenze inverse con shift converge in 12 iterazioni
all'autovalore lambda = 9.8567
```

Tale autovalore può essere correttamente stimato anche con il metodo delle potenze dirette, in un numero di iterazioni maggiore:

```
>> [lambda,x,iter]=eigpower(B,1e-6,1000,ones(4,1));
Il metodo delle potenze converge in 20 iterazioni all'autovalore
lambda = 9.8567
```

Esercizio 4

1. L'implementazione della function è la seguente:

```
function D=qrbasic(A,tol,nmax)
%  QRBASIC calcola gli autovalori di una matrice
%  D=QRBASIC(A,TOL,NMAX) calcola con il metodo delle
%  iterazioni QR tutti gli autovalori della matrice A
%  a meno di una tolleranza TOL ed in un numero massimo
%  di iterazioni NMAX. La convergenza di questo metodo
%  non e' in generale garantita.
[n,m]=size(A);
if n ~= m
  error('La matrice deve essere quadrata')
end
T = A;
niter = 0;
test = max(max(abs(tril(T,-1))));
while niter < nmax & test > tol
  [Q,R]=qr(T);
  T = R*Q;
  niter = niter + 1;
  test = max(max(abs(tril(T,-1))));
end
if niter > nmax
 fprintf(['Il metodo non converge nel massimo',...
            ' numero di iterazioni permesso']);
else
 fprintf('Il metodo converge in %d iterazioni\n',niter)
end
D = diag(T);
return
```

   Si noti come grazie al comando `tril(T,-1)` che estrae la parte sottodiagonale della matrice `T` il valore `test` che viene utilizzato per testare la convergenza del codice viene ottenuto con il semplice comando:

```
test = max(max(abs(tril(T,-1))));
```

2. Per la prima matrice:

```
>> A1 = [30 2 3 13; 5 11 10 8; 9 7 6 12 ; 4 14 15 1];
>> D1 = qrbasic(A1, 1e-10, 1000)
Il metodo converge in 41 iterazioni
D1 =
   39.3960
   17.8208
   -9.5022
    0.2854
```

   Il metodo converge in 41 iterazioni, mentre per la seconda matrice:

```
>> A2 = [-30 2 3 13; 5 11 10 8; 9 7 6 12 ; 4 14 15 1];
>> D2 = qrbasic(A2, 1e-10, 1000)
Il metodo converge in 843 iterazioni
D2 =
  -30.6430
   29.7359
  -11.6806
    0.5878
```

la convergenza è ottenuta in 843 iterazioni. Questo perché la velocità di convergenza del metodo dipende da $\max_i |\lambda_{i+1}/\lambda_i|$; per la prima matrice questa quantità vale

```
>> for i=1:length(D1)-1
    v1(i) = abs(D1(i+1)/D1(i));
   end
>> V1 = max(v1)
V1 =
    0.5332
```

mentre per la seconda matrice si ha:

```
>> for i=1:length(D2)-1
    v2(i) = abs(D2(i+1)/D2(i));
   end
>> V2 = max(v2)
V2 =
    0.9704
```

Questo valore così prossimo ad 1 rallenta notevolmente la convergenza dell'algoritmo.

Esercizio 5

1. Carichiamo l'immagine `street1.jpg`:

```
>> IM = imread( "street1.jpg" );
>> IM = rgb2gray( IM );
>> A = double( IM );
>> [ m, n ] = size( A );
```

   Notiamo che il rango di A è:

```
>> p = min( m, n )
p =
    480
```

2. Otteniamo i fattori della SVD con il comando:

```
>> [ U, S, V ] = svd( A );
```

   Per ottenere A_k, è sufficiente moltiplicare i termini della SVD ristretti al rango k scelto. Nel nostro caso:

```
>> k = 10;
>> Ak = U( :, 1 : k ) * S( 1 : k, 1 : k ) * ( V( :, 1 : k ) )';
```

   La matrice A_k ha le stesse dimensioni della matrice A, ma $\text{rank}(A_k) = 10$.

3. Per visualizzare le due immagini a confronto, insieme all'andamento dei valori singolari della matrice A, usiamo i comandi (vedi Figura 5):

```
>> figure
>> subplot( 2, 2, 1 );    imshow( IM );   title('uncompressed image');
>> subplot( 2, 2, 3 );   imshow( uint8( Ak ) )
>> title('compressed image - rank k')
>> subplot( 1, 2, 2 );
>> loglog( 1 : p, diag( S ), '-', [ k k ], [ S( p, p) S( 1, 1 ) ], '--r', ...
           'LineWidth', 2 );
>> grid on;   xlabel( 'i' );   ylabel('\sigma_i');   title( '\sigma_i vs. i' );
```

4. L'immagine originale richiede di memorizzare il seguente numero di elementi in A:

```
>> uncompressed_size = m * n;
```

   Per salvare l'immagine compressa, invece, è sufficiente salvare gli elementi dei fattori U, Σ e V della SVD troncati al valore k del rango ridotto, ovvero:

```
>> compressed_size = m * k + k + k * n;
```

   Otteniamo quindi il rapporto di compressione:

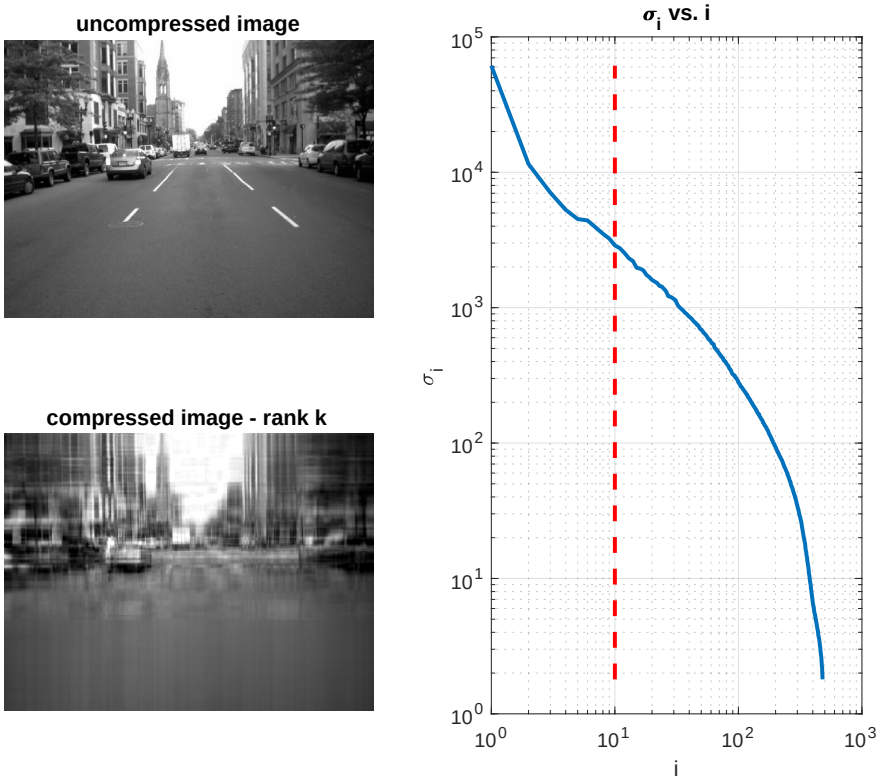

Figura 5: Esercizio 5. A sinistra, l'immagine `street1.jpg` e la sua approssimazione di rango $k = 10$. A destra i valori singolari σ_i di A vs i.

```
>> compression_ratio = uncompressed_size / compressed_size
compression_ratio =
    27.4041
```

Osserviamo che, sulla base del Teorema di Schmidt–Eckart–Young, l'errore relativo commesso dall'approssimazione A_k di A in norma di Frobenious è:

$$\frac{\|A - A_k\|_F}{\|A\|_F} = \frac{\sqrt{\sum_{i=k+1}^{p} \sigma_i^2}}{\sqrt{\sum_{i=1}^{p} \sigma_i^2}}.$$

Otteniamo il seguente errore relativo per $k = 10$:

```
>> relative_error = sqrt( sum( diag( S( k + 1 : end, k + 1 : end ) ).^2 )  ...
                 / sum( diag( S ).^2 ) )
relative_error =
    0.1482
```

Esercizio 6

1. Si veda la funzione `sdpcond.m`, anche riportata di seguito.

```
function K = sdpcond( A, tol, nmax )
%
% SDPCOND Metodo per il calcolo del numero di condizionamento spettrale di matrici
% simmetriche e definite positive
%
% Parametri di ingresso:
%
% A: matrice di sistema
% tol:  tolleranza criterio d'arresto
% nmax: numero massimo di iterazioni ammesse
%
% Parametri di uscita:
%
% K: numero di condizionamento spettrale di A
%

% Inizializzazione
n = size( A, 2 );
x0 = rand( n, 1 );
iter = 0;
y = x0 / norm( x0 );
y_mu = y;
lambda = y' * A * y;
mu = lambda;
K = 1;
err = tol + 1; % dobbiamo entrare nel ciclo
[ L, U, P ] = lu( A ); % O(2/3 n^3)

while err > tol && iter<nmax
   iter = iter + 1;

   x = A * y;             % 2n^2-n ops
   y = x / norm( x );     % 2n+1 ops
   lambda = y' * A * y;   % 2n^2+n-1 ops

   z = fwsub( L, P * y_mu ); % n^2 ops
   x = bksub( U, z );        % n^2 ops
   y_mu = x / norm( x ); % 2n+1 ops
   mu = y_mu' * A * y_mu; % 2n^2+n-1 ops

   K_new = lambda / mu;
   err = abs( K - K_new );
   K = K_new;
end

end
```

2. Consideriamo i comandi:

```
A = [-2 1 0 0; 1 -2 1 0; 0 1 -2 1; 0 0 1 -2];
K = sdpcond(A, 1e-8, 2000)
```

ottenendo come risultato

```
K =
    9.4721
```

3. La fattorizzazione LU richiede $O\left(\frac{2}{3}n^3\right)$ operazioni. All'interno del ciclo for, effettuiamo 3 moltiplicazioni matrice–vettore ($O(2n^2)$ di operazioni ciascuna) e due chiamate alle funzioni `fwsub` e `bksub` per gli algoritmi delle sostituzioni in avanti e all'indietro (n^2 operazioni ciascuna), mentre i comandi rimanenti hanno un costo al più lineare rispetto ad n. Essendo k il numero di iterazioni effettuate otteniamo il totale di $O\left(\frac{2}{3}n^3 + 8n^2\,k\right)$ operazioni.