Sistemi lineari sovradeterminati e SVD

27 febbraio 2009

1 Sistemi lineari sovradeterminati

Si consideri il problema

$$\begin{aligned} x_1 + x_2 &= 1 \\ x_1 - x_2 &= 0 \\ x_1 + 3x_2 &= 0 \end{aligned}$$

Risulta chiaro che il sistema non ha soluzione. Infatti l'unica soluzione delle prime due equazioni è $x_1 = x_2 = 1/2$, che però non verifica $x_1 + 3x_2 = 0$.

Ritrascrivendo tutto in termini matriciali, possiamo dunque affermare che il problema $Ax = b$, con

$$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 3 \end{pmatrix}$$

e $b = (1,0,0)$ non ha soluzione.

In alternativa, risulta quindi ragionevole calcolare $x^* = (x_1^*, x_2^*)$ tale che

$$\gamma := \min_{x \in \mathscr{C}^n} \|b - Ax\|_2 = \|b - Ax^*\|_2 \tag{1}$$

dove

$$\|u\|_2 = \sqrt{\sum_i u_i^2}.$$

La prima questione riguarda l'esistenza e unicità di tale minimo x^*. A tal proposito citiamo il segue teorema (cf. [1], p. 432)

Teorema 1.1 *Sia X l'insieme dei vettori di $\mathscr{C}^n$ tali che $\hat{x} \in X$ se e solo se*

$$\min_{x \in \mathscr{C}^n} \|b - Ax\|_2 = \|b - A\hat{x}\|_2 \tag{2}$$

Supponiamo $A \in \mathscr{C}^{m \times n}$ con $m \geq n$ e $b \in \mathscr{C}^n$. Valgono le seguenti proprietà

1. *$x \in X$ se e solo se*

$$A^H Ax = A^H b, \tag{3}$$

   cioé x risolve il sistema delle equazioni normali.

2. *X è un insieme non vuoto, chiuso e convesso.*
3. *Esiste $x^* \in X$ tale che*
$$\|x^*\|_2 = \min_{x \in X} \|x\|_2.$$
Tale x^* è detto soluzione di minima norma.
4. *L'insieme X si riduce ad un solo elemento x^* se e solo se la matrice A ha rango massimo.*

In altre parole, se A ha rango n allora X ha un unico elemento, mentre se A ha rango minore di n allora X ha un unico elemento di minima norma 2.

2 Alcune fattorizzazioni di matrici rettangolari.

2.1 Fattorizzazione QR

Data una matrice $A \in \mathscr{C}^{m \times n}$ esistono $Q \in \mathscr{C}^{m \times n}$ unitaria (cioè $QQ^H = I_m$, $Q^H Q = I_n$) ed $R \in \mathscr{C}^{n \times n}$ triangolare superiore tali che $A = QR$. Si osservi che a seconda degli autori la fattorizzazione QR sopraindicata viene sostituita con la fattorizzazione $A = QR$ con $Q \in \mathscr{C}^{m \times m}$ unitaria ed $R \in \mathscr{C}^{m \times n}$ triangolare superiore cioè

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

con $R_1 \in \mathscr{C}^{n \times n}$ triangolare superiore.

2.2 Fattorizzazione SVD

Sussiste il seguente teorema [7]:

Teorema 2.1 *Sia $A \in \mathscr{C}^{m \times n}$. Allora esistono due matrici unitarie $U \in \mathscr{C}^{m \times m}$, $V \in \mathscr{C}^{n \times n}$ e una matrice diagonale $\Sigma \in \mathscr{R}_+^{m \times n}$ avente elementi σ_{ij} nulli per $i \neq j$ e uguali a $\sigma_i \in \mathscr{R}_+$ per $i = j$ con*
$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0 \tag{4}$$
tali che
$$A = U \Sigma V^H.$$

Il termine SVD sta per *singular value decomposition* e sottolinea la presenza dei valori singolari σ_i. Se λ_i e σ_i sono rispettivamente gli autovalori di $A^T A$ in ordine decrescente e i valori singolari di A in ordine decrescente, allora

$$\lambda_i = \sqrt{\sigma_i},\ i = 1, \ldots, n.$$

3 Fattorizzazione QR ed SVD in Matlab

Vediamo di seguito alcuni dettagli di come tali fattorizzazioni sono implementate in Matlab.

3.1 QR in Matlab

In merito alla fattorizzazione QR, l'help di Matlab fornisce le seguenti indicazioni:

```
>> help qr

 QR     Orthogonal-triangular decomposition.
    [Q,R] = QR(A) produces an upper triangular matrix R of the same
    dimension as A and a unitary matrix Q so that A = Q*R.

    [Q,R,E] = QR(A) produces a permutation matrix E, an upper
    triangular R and a unitary Q so that A*E = Q*R.  The column
    permutation E is chosen so that abs(diag(R)) is decreasing.

    [Q,R] = QR(A,0) produces the "economy size" decomposition.
    If A is m-by-n with m > n, then only the first n columns of Q
    are computed.

    [Q,R,E] = QR(A,0) produces an "economy size" decomposition in
    which E is a permutation vector, so that Q*R = A(:,E).  The column
    permutation E is chosen so that abs(diag(R)) is decreasing.

    By itself, QR(A) is the output of LAPACK'S DGEQRF or ZGEQRF routine.
    TRIU(QR(A)) is R.

    For sparse matrices, QR can compute a "Q-less QR decomposition",
    which has the following slightly different behavior.

    R = QR(A) returns only R.  Note that R = chol(A'*A).
    [Q,R] = QR(A) returns both Q and R, but Q is often nearly full.
    [C,R] = QR(A,B), where B has as many rows as A, returns C = Q'*B.
    R = QR(A,0) and [C,R] = QR(A,B,0) produce economy size results.

    The sparse version of QR does not do column permutations.
    The full version of QR does not return C.

    The least squares approximate solution to A*x = b can be found
    with the Q-less QR decomposition and one step of iterative refinement:

        x = R\(R'\(A'*b))
        r = b - A*x
        e = R\(R'\(A'*r))
        x = x + e;

    See also LU, NULL, ORTH, QRDELETE, QRINSERT, QRUPDATE.

>>
```

Nel caso della matrice

$$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 3 \end{pmatrix}$$

vediamo alcune fattorizzazioni QR.

```
>> A=[1 1; 1 -1; 1 3]

A =

     1     1
     1    -1
     1     3
>> % PRIMA FATTORIZZAZIONE QR.
>> [Q,R]=qr(A)
Q =
   -0.5774    0.0000   -0.8165
   -0.5774   -0.7071    0.4082
   -0.5774    0.7071    0.4082
R =
   -1.7321   -1.7321
         0    2.8284
         0         0
>> Q'*Q
ans =
    1.0000         0   -0.0000
         0    1.0000   -0.0000
   -0.0000   -0.0000    1.0000
>> norm(A-Q*R)
ans =
  4.9938e-016
>> % SECONDA FATTORIZZAZIONE QR.
>> [Q,R]=qr(A,0)
Q =
   -0.5774    0.0000
   -0.5774   -0.7071
   -0.5774    0.7071
R =
   -1.7321   -1.7321
         0    2.8284
>> Q'*Q
ans =
    1.0000         0
         0    1.0000
>> norm(A-Q*R)
ans =
  4.9938e-016
>> % TERZA FATTORIZZAZIONE QR.
```

```
>> [Q,R,E]=qr(A)
Q =
   -0.3015   -0.4924   -0.8165
    0.3015   -0.8616    0.4082
   -0.9045   -0.1231    0.4082
R =
   -3.3166   -0.9045
         0   -1.4771
         0         0
E =
     0     1
     1     0
>> Q'*Q
ans =
    1.0000         0    0.0000
         0    1.0000    0.0000
    0.0000    0.0000    1.0000
>> norm(A*E-Q*R)
ans =
  5.6322e-016
>> A*E-Q*R

ans =

  1.0e-015 *

    0.2220    0.1110
    0.2220    0.2220
   -0.4441         0

>>
```

Nota facoltativa. Dal punto di vista implementativo è facile calcolare la fattorizzazione QR di A anche quando il rango A non è massimo, utilizzando `[Q,R,E]=qr(A)`. Si stabilisce facilmente il rango r di A cercando il numero di elementi diagonali di R non nulli.

3.2 SVD in Matlab

Osserviamo che Matlab dispone del comando `svd`. La descrizione dell'help è la seguente:

```
>> help svd

 SVD    Singular value decomposition.
    [U,S,V] = SVD(X) produces a diagonal matrix S, of the same
    dimension as X and with nonnegative diagonal elements in
```

```
    decreasing order, and unitary matrices U and V so that
    X = U*S*V'.

    S = SVD(X) returns a vector containing the singular values.

    [U,S,V] = SVD(X,0) produces the "economy size"
    decomposition. If X is m-by-n with m > n, then only the
    first n columns of U are computed and S is n-by-n.

    See also SVDS, GSVD.

 Overloaded methods
    help sym/svd.m

>>
```

Per fare pratica con questo comando sia

$$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 3 \end{pmatrix}$$

e utilizziamo `[U,S,V] = svd(A)` per avere la decomposizione SVD della matrice A. Il risultato è

```
>> A=[1 1; 1 -1; 1 3]

A =

     1     1
     1    -1
     1     3

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

U =

    0.3651    0.4472   -0.8165
   -0.1826    0.8944    0.4082
    0.9129   -0.0000    0.4082


S =

    3.4641         0
         0    1.4142
         0         0
```

```
V =

    0.3162    0.9487
    0.9487   -0.3162
>> % U, V SONO UNITARIE.
>> U'*U
ans =
    1.0000   -0.0000   -0.0000
   -0.0000    1.0000    0.0000
   -0.0000    0.0000    1.0000

>> V'*V
ans =
     1     0
     0     1
>>
>> X = U*S*V'

X =

    1.0000    1.0000
    1.0000   -1.0000
    1.0000    3.0000

>>
```

Si puó dimostrare che il rango della matrice è esattamente il numero di σ_i non nulli. In realtá la tentazione di determinarlo in virtú dei termini diagonali di Σ puó risultare fallace in quanto il computer fornisce solo un'approssimazione dei σ_i e qualora questi siano molto piccoli non si puó determinare con sicurezza il rango della matrice.

Nota Facoltativa. Esiste un legame tra valori singolari e autovalori. Più precisamente se $\{\lambda_i\}$ sono gli autovalori di $A^H A$ allora $\sigma_i = \sqrt{\lambda_i}$. Vediamolo con un esempio:

```
>> A=[1 1; 1 -1; 1 3]
A =
     1     1
     1    -1
     1     3
>> [U,S,V]=svd(A);
>> S
S =
    3.4641         0
         0    1.4142
         0         0
>> % LEGAME VALORI SINGOLARI E AUTOVETTORI.
```

```
>> lambda=eig(A'*A)
lambda =
     2
    12
>> sqrt(12)
ans =
    3.4641
>> sqrt(2)
ans =
    1.4142
>>
```

4 Calcolo della soluzione ai minimi quadrati

Mostriamo ora tre metodi per risolvere il problema sopra descritto, cioè relativo al calcolo di x^* che minimizza la norma 2 del residuo, cioè $\|b - Ax\|_2$.

4.1 Risoluzione equazioni normali

Supponiamo $A \in \mathbb{R}^{m\times n}$ abbia rango n. Dal teorema 1.1 sappiamo che la soluzione ai minimi quadrati esiste, è unica e risolve $A^H Ax = A^H b$. Nel caso A abbia coefficienti in $\mathscr{R}$, ció si riduce a risolvere $A^T Ax = A^T b$. In questo caso $A^H A$ è definita positiva e si pò utilizzare il metodo di Cholesky. Se $LL^H = A^H A$ per L triangolare inferiore (fattorizzazione di Cholesky) basta risolvere

$$Ly = A^H b,$$
$$L^H x = y.$$

Il costo computazionale è di $n^2 m/2$ operazioni moltiplicative per la costruzione di $A^H A$ e di $n^3/6$ moltiplicative per la soluzione del sistema, e quindi il costo totale è di

$$n^2 m/2 + n^3/6$$

operazioni moltiplicative. Osserviamo che se A non ha rango massimo non si puó applicare il metodo di Cholesky ma il metodo di Gauss con variante di massimo pivot.
Nota facoltativa. Se A non ha rango massimo non si può applicare il metodo di Cholesky, ma l'eliminazione gaussiana con la variante del massimo pivot.

4.2 Metodo QR

Supponiamo il rango di $A \in \mathbb{R}^{m\times n}$ sia $r = n$. Nota la fattorizzazione QR si deduce che $Rx = Q^H QRx = Q^H b \in \mathscr{R}^n$.

Osserviamo infatti che essendo $Q^H Q = I_n$

$$A = QR,\ Ax = b \Rightarrow QRx = b \Rightarrow Rx^* = Q^H b$$

e quindi prima calcoliamo $c = Q^H b$ e poi risolviamo $Rx^* = c$. Il passaggio

$$QRx = b \Rightarrow Rx^* = Q^H b$$

è delicato, in quanto $Q \in \mathscr{C}^{m\times n}$ è unitaria (cioè $QQ^H = I_m$, $Q^H Q = I_n$) ma rettangolare. Dice che posto $y = Rx^*$, l'equazione diventa $Qy = b$. Infatti, le corrispettive equazioni normali (si ricordi che $Q \in \mathscr{C}^{m\times n}$ con $m \geq n$ è unitaria) sono

$$Qy = b \Rightarrow Q^H Q y = Q^H b \Rightarrow y = Q^H b \Rightarrow Rx^* = Q^H b.$$

Se A ha *rango massimo* allora R è non singolare e quindi il problema $Rx = Q^H b$ risolto facilmente per sostituzione all'indietro. Il costo computazionale per la fattorizzazione QR è di $n^2(m - n/3)$, il costo computazionale della risoluzione del sistema triangolare è $n^2/2$ operazioni moltiplicative. Quindi il costo totale è

$$n^2(m - n/3) + n^2/2.$$

Nota facoltativa. Se A non ha *rango massimo* allora $AE = QR$ con E matrice di permutazione,

$$R = \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix}$$

dove $R \in \mathscr{C}^{n\times n}$, $R_1 \in \mathscr{C}^{r\times r}$ sono matrici triangolari superiori. Si dimostra (non facile) che posto

$$c = Q^H b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, c_1 \in \mathscr{C}^n, c_2 \in \mathscr{C}^{n-m}$$

la soluzione ai minimi quadrati di minima norma risolve il problema

$$R_1 x = c_1.$$

4.3 Metodo SVD

Siano u_i, v_i le i-sime righe rispettivamente di U e V. Per capire come risolvere un sistema lineare sovradeterminato tramite questa fattorizzazione ci affidiamo al seguente teorema

Teorema 4.1 *Sia $A \in \mathscr{C}^{m\times n}$ di rango r con $m \geq n$ e sia*

$$A = U\Sigma V^H$$

la decomposizione ai valori singolari di A. Allora la soluzione di minima norma è data da

$$x^* = \sum_{i=1}^{r} \frac{u_i^H b}{\sigma_i} v_i$$

e

$$\gamma^2 = \sum_{i=r+1}^{m} |u_i^H b|^2$$

dove al solito

$$\gamma := \min_{x\in\mathscr{C}^n} \|b - Ax\|_2 = \|b - Ax^*\|_2. \tag{5}$$

5 Un esempio: soluzione di sistemi sovradeterminati.

Proviamo a risolvere il problema iniziale

$$\begin{aligned} x_1 + x_2 &= 1 \\ x_1 - x_2 &= 0 \\ x_1 + 3x_2 &= 0 \end{aligned}$$

coi metodi sopracitati.

Lavoriamo direttamente nella shell di Matlab/Octave:

```
>> % EQUAZIONI NORMALI.
>> A=[1 1; 1 -1; 1 3];
>> b=[1 0 0]';
>> C=A'*A;
>> b1=A'*b;
>> format long
>> x=C\b1
x =
   0.33333333333333
  -0.00000000000000
>> % METODO QR
>> [Q,R]=qr(A);
>> size(Q)
ans =
     3     3
>> size(R)
ans =
     3     2
>> % COMMENTO: NON E'LA FATTORIZZAZIONE CERCATA
>> % "R" DEVE ESSERE QUADRATA.
>> [Q,R]=qr(A,0);
>> size(Q)
ans =
     3     2
>> size(R)
ans =
     2     2
>> b1=Q'*b;
>> size(b1)
ans =
     2     1
>> x=R\b1
x =
   0.33333333333333
   0.00000000000000
>> % SOLUZIONE VIA SVD.
>> [U,S,V]=svd(A);
>> size(U)
ans =
```

```
     3     3
>> size(V)
ans =
     2     2
>> size(b)
ans =
     3     1
>> c1=U'*b;
>> size(c1)
ans =
     3     1
>> c2=c1(1:size(V,1));
>> s=c2./diag(S);
>> x=V*s
x =
   0.33333333333333
   0.00000000000000
>> gamma_value=norm(A*x-b,2)
gamma_value =
   0.81649658092773
>> abs((U(:,3))'*b)
ans =
   0.81649658092773
>>
```

Commentiamo i risultati a partire da quanto visto per il metodo QR, visto che la risoluzione via equazioni normali è del tutto ovvia.

1. L'implementazione in Matlab di QR per matrici rettangolari A richiede una precisazione. Sia x^* la soluzione del problema sovradeterminato. Il comando `[Q,R]=qr(A)` comporta che Q sia quadrata, mentre `[Q,R]=qr(A,0)` implica che R è quadrata, come da noi richiesto.

2. Per quanto riguarda la soluzione via SVD, l'unico punto da osservare è che in un precedente teorema si dice che

   $$x^* = \sum_{i=1}^{k} \frac{u_i^H b}{\sigma_i} v_i$$

   dove u_i, v_i sono le righe di U e V. La sommatoria è limitata alle prime k righe e per questo abbiamo effettuato

   ```
   >> c1=U'*b;
   >> c2=c1(1:size(V,1));
   >> s=c2./diag(S);
   ```

   Si osservi che il rango k per la matrice in questione è uguale a n cioè 2, che altri non è che $size(V,1)$. Se il rango non fosse stato massimo, cioè $r \neq n$, prima sarebbe stato opportuno calcolare il rango r e poi sostituire `c1(1:size(V,1),:);` con `c1(1:r,:);`

Ricordiamo che `1:r` genera il vettore riga $[1\,2\,\ldots\,r]$ e che `c2=c1(1:r);` assegna a `c2` le righe dalla prima all' r-sima del vettore `c1`

```
>> r=5;
>> 1:r
ans =
     1     2     3     4     5
>> c1=[1 4 2 6 4 8 4]
c1 =
     1     4     2     6     4     8     4
>> c1(1:r)
ans =
     1     4     2     6     4
```

Essendo v_i le righe di V, posto $s_i = u_i^H b/\sigma_i$ risulta che

$$x^* = \sum_{i=1}^{r} \frac{u_i^H b}{\sigma_i} v_i = V s.$$

Nell'ultima parte si mostra che effettivamente la norma 2 del residuo cioè $\|Ax^* - b\|$ coincide con

$$\gamma^2 = \sum_{i=r+1}^{m} |u_i^H b|^2.$$

In merito osserviamo che il rango della matrice è $r = 2$ ed $m = 3$ e quindi

$$\gamma^2 = \sum_{i=r+1}^{m} |u_i^H b|^2 = |u_i^H b|^2$$

da cui

$$\gamma^2 = |u_3^H b|^2 \Rightarrow \gamma = |u_3^H b|.$$

3. Può restare il dubbio di come si faccia a sapere per via numerica che il rango di A è 2. Questo corrisponde al numero di coefficienti diagonali di S non nulli e quindi da

```
>> A=[1 1; 1 -1; 1 3];
>> [U,S,V]=svd(A);
>> diag(S)
ans =
    3.4641
    1.4142
>> t=find(abs(diag(S))>0);
>> rango=length(t)
rango =
     2
>>
```

deduciamo che il rango è 2.

6 Un esempio: compressione immagini.

Consideriamo i siti web [6], [2], [4]. L'argomento esposto è la compressione di immagini via SVD. In particolare si accenna ad una interessante implementazione in Matlab (non funziona in Octave!) relativa alla compressione di immagini. Entriamo nei dettagli. Sia $A = U\Sigma V^T$ la fattorizzazione SVD della matrice A. Se σ_i sono gli elementi diagonali della matrice diagonale e rettangolare Σ, u_k la k-sima colonna di U, v_k la k-sima colonna di V, allora

$$A = \sum_{k=1}^{n} \sigma_k u_k v_k^T.$$

Infatti, se $e_k \in \mathbb{R}^m$ è il vettore colonna avente tutte componenti nulle eccetto la k sima che è uguale a 1, indicato con $V_{k,\cdot}^H \in \mathbb{C}^n$ la k-sima riga di V^H e con $U_{\cdot,k}$ la k-sima colonna di U abbiamo dopo qualche conto (provare a farli, anche a modo proprio!) che

$$\Sigma V^H = \sum_{k=1}^{n} \sigma_k e_k * V_{k,\cdot}^H$$

da cui evidenziando il prodotto tra matrici $*$, ricordando la sua proprietà associativa e ricordando la linearità delle operazioni

$$\begin{aligned} U * \Sigma * V^H &= U * (\Sigma * V^H) \\ &= U * \left(\sum_{k=1}^{n} \sigma_k e_k * V_{k,\cdot}^H \right) \\ &= \sum_{k=1}^{n} \sigma_k U * e_k * V_{k,\cdot}^H \\ &= \sum_{k=1}^{n} \sigma_k (U * e_k) * V_{k,\cdot}^H \\ &= \sum_{k=1}^{n} \sigma_k U_{\cdot,k} * V_{k,\cdot}^H \end{aligned} \tag{6}$$

per cui essendo u_k la k-sima colonna di U, v_k la k-sima colonna di V abbiamo $u_k = U_{\cdot,k}$ e $v_k^H = V_{k,\cdot}^H$, nonchè

$$U * \Sigma * V^H = \sum_{k=1}^{n} \sigma_k u_k * v_k^H.$$

Osserviamo che le matrici $C_k := u_k * v_k^H$ hanno rango 1 in quanto la loro immagine è uno spazio di dimensione 1, visto che

$$C_k * y = u_k * v_k^H * y = (v, y) u_k$$

dove al solito $(\cdot, \cdot)$ è il prodotto scalare in $\mathbb{C}^n$.

Se l'indice k viene ordinato cosicchè i valori singolari siano in ordine decrescente, cioè

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$$

e tronchiamo la somma dopo r termini otteniamo una approssimazione di rango r della matrice A. Se in particolare la matrice A è una matrice ottenuta codificando i bit di un'immagine, si capisce che si può comprimere la foto utilizzando per $r < n$ invece di A la matrice

$$A_r = \sum_{k=1}^{r} \sigma_k u_k v_k^H.$$

Vediamone un implementazione in Matlab. Scarichiamo dal sito [4] il file `compression.zip` e una volta decompresso, utilizziamo la routine `imcompr.m` per comprimere un'immagine di tipo bitmap, come ad esempio il gatto salvato in `gatto.bmp`. Osserviamo che il gatto è codificato tramite una matrice di interi di dimensione 416×325.

Lanciando dalla shell di Matlab (non funziona in Octave!) il comando

```
>> [im] = imcompr ('gatto.bmp',20,'gatto20.bmp');
>> [im] = imcompr ('gatto.bmp',50,'gatto50.bmp');
>> [im] = imcompr ('gatto.bmp',100,'gatto100.bmp');
```

Il risultato sono 3 foto dello stesso gatto, con diversi fattori di compressione, in quanto abbiamo usato valori di r diversi (r=20, 50, 100).

Per ulteriori dettagli si confronti [3], p.180, esempio 6.9.
Nota. Siò notare che dopo la compressione, la dimensione del file compresso è maggiore di quello originale. La ragione è che il secondo è relativo ad una matrice avente componenti intere mentre quelle del primo sono reali. Nonostante questo problema, l'idea mostra una possibile strada per comprimere immagini. Inoltre non è chiaro perchè la compressione funzioni in pratica e quindi .

7 Un esempio: approssimazione polinomiale.

Sia f una funzione reale e continua e siano x_i punti a due a due distinti appartenenti al dominio di f. Si cerca il polinomio

$$p_{n-1}(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1},\ a_i \in \mathscr{R}$$

per cui sia minimo lo scarto quadratico

$$\sum_{i=1}^{m} [p_{n-1}(x_i) - f(x_i)]^2.$$

Tale polinomio p_{n-1} esiste unico ed è noto come *polinomio di miglior approssimazione* . Si prova che ció corrisponde a risolvere il problema sovradeterminato $Va = \bar{f}$ dove

$$V = \begin{pmatrix} 1 & x_1 & \ldots & x_1^{n-1} \\ 1 & x_2 & \ldots & x_2^{n-1} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & \ldots & x_m^{n-1} \end{pmatrix}$$

Figura 1: Compressione di una foto via SVD.

e $\bar{f} = (f(x_1),\dots,f(x_m))$, $x \in \mathscr{R}^m$, , $a = (a_i) \in \mathscr{R}^n$. Questa idea è alla base dell'approssimazione fornita dalla toolbox `polyfit`.
Vediamo i dettagli. Notiamo subito che

$$p_{n-1}(x_i) = a_0 + a_1 x_i + \ldots + a_{n-1} x_i^{n-1} = V_{i,\cdot}\, a$$

e quindi per definizione di $\bar{f}$

$$\sum_{i=1}^{m} [p_{n-1}(x_i) - f(x_i)]^2 = \sum_{i=1}^{m} [V_{i,\cdot}\, a - \bar{f}_i]^2.$$

Ricordiamo ora che la soluzione di un sistema sovradeterminato $Ax = b$ (con $A \in \mathscr{R}^{m\times n}$) è quella che minimizza $\|Ax - b\|_2$ e quindi pure

$$\|Ax - b\|_2^2 = \sum_i [(A * x)_i - b_i]^2.$$

Di conseguenza il vettore a avente quali coefficienti i termini a_i del *polinomio di miglior approssimazione* del problema di approssimazione è il vettore che risolve il sistema sovradeterminato $Va = \bar{f}$.

References

[1] D. Bini, M. Capovani, O. Menchi, *Metodi numerici per l'algebra lineare*, Zanichelli, 1988.

[2] Cleve's corner, Professor SVD
http://www.mathworks.com/company/newsletters/news_notes/oct06/clevescorner.html.

[3] A. Quarteroni ed F. Saleri, *Introduzione al Calcolo Scientifico*, Springer, 2006.

[4] L. Rosa, *Image compression*
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=4772&objectType=File.

[5] K. Sandberg, *The Singular Value Decomposition of an Image*
http://amath.colorado.edu/courses/4720/2000Spr/Labs/SVD/svd.html

[6] G.W. Steward, *On the Early History of the Singular Value Decomposition*
http://www.ima.umn.edu/preprints/April92/952.pdf.

[7] Wikipedia (Singular value decomposition)
http://en.wikipedia.org/wiki/Singular_value_decomposition.