Capitolo 1

Approssimazione di autovalori e autovettori

1.1 Richiami di algebra lineare

In questo capitolo descriveremo il metodo delle potenze per il calcolo dell'autovalore di massimo modulo di una matrice reale e una sua importante applicazione, cioè il cosiddetto algoritmo per il PageRank utilizzato dal motore di ricerca Google per assegnare un ranking alle pagine web. Prima di affrontare questo problema richiamiamo alcuni concetti e definizioni di algebra lineare.
D'ora in poi saranno considerati vettori appartenenti all'insieme $\mathbb{R}^n$.

Definizione 1.1.1 *Si definisce Prodotto scalare ogni funzione*

$$(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

che verifica le seguenti proprietà:

1. $(\mathbf{x},\mathbf{x}) \geq 0$ *per ogni* $\mathbf{x} \in \mathbb{R}^n$ *e* $(\mathbf{x},\mathbf{x}) = 0$ *se e solo se* $\mathbf{x} = 0$*;*
2. $(\mathbf{x},\mathbf{y}) = (\mathbf{y},\mathbf{x})$*;*
3. $(\alpha\mathbf{x} + \beta\mathbf{y},\mathbf{z}) = \alpha(\mathbf{x},\mathbf{z}) + \beta(\mathbf{y},\mathbf{z})$ *per ogni* $\alpha,\beta \in \mathbb{R}$ *e per ogni* $\mathbf{x},\mathbf{y},\mathbf{z} \in \mathbb{R}^n$.

Il classico prodotto scalare su $\mathbb{R}^n$ è definito come

$$(\mathbf{x},\mathbf{y}) = \mathbf{y}^T\mathbf{x} = \sum_{i=1}^{n} x_i y_i.$$

Definizione 1.1.2 *La funzione* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_+$ *si dice norma vettoriale se per ogni* $\mathbf{x} \in \mathbb{R}^n$ *soddisfa le seguenti proprietà:*

1. $\|\mathbf{x}\| \geq 0$ *e* $\|\mathbf{x}\| = 0$ *se e solo se* $\mathbf{x} = 0$*;*
2. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ *per ogni* $\alpha \in \mathbb{R}$*;*
3. $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ *per ogni* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ *(Disuguaglianza triangolare).*

Poichè il prodotto scalare tra un vettore e se stesso è un valore nonnegativo allora ogni prodotto scalare induce una norma, ed in particolare il prodotto scalare su $\mathbb{R}^n$ definisce la seguente norma:

$$\|\mathbf{x}\|_2 = \sqrt{(\mathbf{x}, \mathbf{x})} = \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

che viene appunto norma 2 o norma euclidea (ed indica la lunghezza del vettore stesso). Altre norme molto utilizzate sono le seguenti:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^{n} |x_j| \qquad \text{norma 1}$$

$$\|\mathbf{x}\|_\infty = \max_{1\leq j\leq n} |x_j| \qquad \text{norma infinito.}$$

Un importante legame tra prodotto scalare e norma è la cosiddetta disuguaglianza di Schwarz:

$$|(\mathbf{x}, \mathbf{y})| \leq \|\mathbf{x}\|\|\mathbf{y}\|. \tag{1.1}$$

Dalla disuguaglianza di Schwarz ed in particolare dal fatto che

$$\frac{(\mathbf{x}, \mathbf{y})}{\|\mathbf{x}\|\|\mathbf{y}\|} \leq 1$$

segue la definizione di angolo tra vettori. Infatti si definisce l'angolo θ tra i vettori $\mathbf{x}$ e $\mathbf{y}$

$$\theta = \arccos \frac{(\mathbf{x}, \mathbf{y})}{\|\mathbf{x}\|\|\mathbf{y}\|}.$$

Se il prodotto scalare tra due vettori è uguale a zero allora i due vettori si dicono ortogonali. Se un vettore $\mathbf{x}$ ha norma diversa da zero allora il vettore $\mathbf{x}/\|\mathbf{x}\|$ ha norma unitaria. Se $\mathbf{x}$ è un vettore di lunghezza unitaria allora il prodotto $\mathbf{x}^T\mathbf{y}$ fornisce la proiezione sulla semiretta su cui giace il vettore $\mathbf{x}$. In maniera simile è possibile definire anche norme su matrici quadrate.

Definizione 1.1.3 *Una funzione* $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ *che per ogni matrice* $A \in \mathbb{R}^{n\times n}$ *soddisfa le seguenti proprietà:*

1. $\|A\| \geq 0$ *e* $\|A\| = 0$ *se e solo se* $A = 0$*;*
2. $\|\alpha A\| = |\alpha|\|A\|$ *per ogni* $\alpha \in \mathbb{R}$*;*
3. $\|A + B\| \leq \|A\| + \|B\|$ *per ogni* $A, B \in \mathbb{R}^{n\times n}$*;*
4. $\|A \cdot B\| \leq \|A\| \cdot \|B\|$ *per ogni* $A, B \in \mathbb{R}^{n\times n}$*;*

si dice norma matriciale.

Definizione 1.1.4 *Si dice che una norma di matrice è compatibile con una norma di vettore se per ogni matrice* A *e per ogni vettore* $\mathbf{x}$ *risulta*

$$\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|.$$

Un modo per definire le norme di matrici compatibili con norme di vettori è il seguente. Sia $\mathbf{x} \neq 0$ con norma $\|\mathbf{x}\|$. Considerata la norma del vettore $A\mathbf{x}$, $\|A\mathbf{x}\|$, definiamo come norma di A il numero $\|A\|$ dato da

$$\|A\| = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\| \tag{1.2}$$

detta norma indotta dalla norma vettoriale $\|\mathbf{x}\|$.
Le principali norme matriciali indotte sono date le seguenti:

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|$$

$$\|A\|_2 = \sqrt{\rho(A^T A)}$$

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$$

dove $\rho(A^T A)$ indica il raggio spettrale di $A^T A$, ovvero la radice quadrata del massimo modulo di un autovalore della matrice $A^T A$. Non risulta essere una norma indotta la cosiddetta norma di Frobenius:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}.$$

Si può facilmente riconoscere che non si tratta di norma indotta in quanto la norma di Frobenius della matrice identità I di ordine n è uguale a $\sqrt{n}$, e non a 1 come dovrebbe risultare dalla relazione (1.2).

Definizione 1.1.5 *Sia $A \in \mathbb{C}^{m\times n}$ si definisce rango di A il numero massimo di righe (o di colonne) della matrice linearmente indipendenti.*

Definizione 1.1.6 *Sia $A \in \mathbb{C}^{n\times n}$ si dice che $\lambda \in \mathbb{C}$ è un autovalore di A se e solo se esiste un vettore $\mathbf{x} \in \mathbb{C}^n$, $\mathbf{x} \neq 0$, tale che:*

$$A\mathbf{x} = \lambda\mathbf{x}. \tag{1.3}$$

Se $\lambda \in \mathbb{C}$ è un autovalore di A, ogni vettore non nullo $\mathbf{x}$ che soddisfa la relazione (1.3) si dice autovettore associato a λ.
La relazione (1.3) può essere scritta come

$$(A - \lambda I)\mathbf{x} = 0, \tag{1.4}$$

dunque il vettore $\mathbf{x}$ deve essere una soluzione non banale del sistema omogeneo (1.4). Affinchè quest'ultimo ammetta soluzioni non banali deve essere

$$\det(A - \lambda I) = 0. \tag{1.5}$$

Sviluppando il primo membro in (1.5) con la regola di Laplace è facile vedere che rappresenta un polinomio di grado n in λ,

$$p_n(\lambda) = \det(A - \lambda I),$$

detto polinomio caratteristico di A. Gli autovalori di A sono le n radici dell'equazione caratteristica

$$\det(A - \lambda I) \equiv p_n(\lambda) = 0.$$

Se la matrice A ha elementi reali allora i coefficienti del polinomio caratteristico sono reali, tuttavia gli autovalori possono anche essere numeri complessi.

1.2 Il Metodo delle Potenze

In questo paragrafo sarà descritto un metodo per il calcolo di un particolare autovalore di una matrice reale. Sia A una matrice di ordine n ad

elementi reali, tale che $\lambda_1, \lambda_2, \ldots, \lambda_n$ siano i suoi autovalori e $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ i corrispondenti autovettori. Quindi

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad i = 1, \ldots, n. \tag{1.6}$$

Ricordiamo inoltre che, moltiplicando l'equazione (1.6) per A risulta

$$A^2\mathbf{v}_i = \lambda_i A\mathbf{v}_i = \lambda_i^2 \mathbf{v}_i,$$

e, generalizzando, risulta

$$A^k\mathbf{v}_i = \lambda_i^k \mathbf{v}_i,$$

ovvero gli autovalori delle potenze di A sono le potenze degli autovalori di A. Assumiamo le seguenti ipotesi semplificative:

1. $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$
2. $\{\mathbf{v}_i\}_{i=1}^n$ linearmente indipendenti.

Sotto la prima ipotesi λ_1 è detto autovalore dominante. Scopo del metodo delle potenze è quello di determinare λ_1 e il corrispondente autovettore. Una prima osservazione è che, sotto queste ipotesi, λ_1 deve essere necessariamente un numero reale. Infatti se la matrice A è reale allora anche il polinomio caratteristico ha coefficienti reali. Se λ_1 fosse un numero complesso allora anche il suo coniugato, $\overline{\lambda}_1$ sarebbe autovalore di A. In questo caso sarebbe violata la prima condizione, in quanto λ_1 e $\overline{\lambda}_1$ hanno il medesimo modulo.
Supponiamo ora che sia $\mathbf{x}^{(0)} \in \mathbb{R}^n$ un vettore arbitrario. Definiamo il processo iterativo

$$\mathbf{x}^{(k+1)} = A\mathbf{x}^{(k)} \qquad k = 0, 1, 2, \ldots \tag{1.7}$$

Detto $\mathbf{e}_j$ il j-esimo versore fondamentale di $\mathbb{R}^n$ si definisce il $(k+1)$-esimo elemento della successione $\lambda_1^{(k+1)}$ come il rapporto tra le j-esime componenti dei vettori $\mathbf{x}^{(k+1)}$ e $\mathbf{x}^{(k)}$, cioè

$$\lambda_1^{(k+1)} = \frac{\mathbf{e}_j^T \mathbf{x}^{(k+1)}}{\mathbf{e}_j^T \mathbf{x}^{(k)}} \tag{1.8}$$

dove $\mathbf{e}_j$ è il $j-$esimo versore della base canonica di $\mathbb{R}^n$ (il vettore che ha tutte le componenti uguali a zero tranne quella di indice j uguale a 1). Si dimostrerà che tale successione converge proprio all'autovalore dominante:

$$\lim_{k\to\infty} \lambda_1^{(k)} = \lambda_1.$$

Osserviamo che ogni vettore appartenente alla successione può essere espresso come

$$\mathbf{x}^{(k)} = A^k \mathbf{x}^{(0)} \qquad k = 0, 1, 2, \ldots \tag{1.9}$$

ovvero come prodotto tra una potenza della matrice A ed il vettore iniziale (da qui il nome di Metodo delle potenze per indicare l'algoritmo in questione). Poichè i vettori $\{\mathbf{v}_i\}_{i=1}^n$ sono linearmente indipendenti formano una base dello spazio $\mathbb{R}^n$ quindi $\mathbf{x}^{(0)}$ può essere espresso come combinazione lineare di tali vettori:

$$\mathbf{x}^{(0)} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i, \tag{1.10}$$

ipotizzando, per semplicità, che sia $\alpha_1 \neq 0$. Sostituendo (1.10) in (1.9), abbiamo

$$\mathbf{x}^{(k)} = A^k \sum_{i=1}^{n} \alpha_i \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i A^k \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k \mathbf{v}_i.$$

Poichè $\alpha_1 \neq 0$ allora si può scrivere

$$\mathbf{x}^{(k)} = \lambda_1^k \left(\alpha_1 \mathbf{v}_1 + \sum_{i=2}^{n} \alpha_i \frac{\lambda_i^k}{\lambda_1^k} \mathbf{v}_i \right) = \lambda_1^k (\alpha_1 \mathbf{v}_1 + \varepsilon^{(k)}) \tag{1.11}$$

avendo posto

$$\varepsilon^{(k)} = \sum_{i=2}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i.$$

Scrivendo la (1.11) per $k+1$ segue che

$$\mathbf{x}^{(k+1)} = \lambda_1^{k+1} (\alpha_1 \mathbf{v}_1 + \varepsilon^{(k+1)}). \tag{1.12}$$

Utilizzando le rappresentazioni (1.11) e (1.12) per i vettori $\mathbf{x}^{(k)}$ e $\mathbf{x}^{(k+1)}$ possiamo riscrivere l'espressione di $\lambda_1^{(k+1)}$:

$$\lambda_1^{(k+1)} = \frac{\lambda_1^{k+1} (\alpha_1 \mathbf{e}_j^T \mathbf{v}_1 + \mathbf{e}_j^T \varepsilon^{(k+1)})}{\lambda_1^k (\alpha_1 \mathbf{e}_j^T \mathbf{v}_1 + \mathbf{e}_j^T \varepsilon^{(k)})} = \frac{\lambda_1 (\alpha_1 \mathbf{e}_j^T \mathbf{v}_1 + \mathbf{e}_j^T \varepsilon^{(k+1)})}{\alpha_1 \mathbf{e}_j^T \mathbf{v}_1 + \mathbf{e}_j^T \varepsilon^{(k)}}.$$

Poichè

$$|\mathbf{e}_j^T \varepsilon^{(k)}| \overset{k \to \infty}{\longrightarrow} 0$$

in quanto $|\lambda_i / \lambda_1| < 1$ per ogni i, segue

$$\lim_{k \to \infty} \frac{\mathbf{e}_j^T \mathbf{x}^{(k+1)}}{\mathbf{e}_j^T \mathbf{x}^{(k)}} = \lambda_1 \, \frac{\alpha_1 \mathbf{e}_j^T \mathbf{v}_1}{\alpha_1 \mathbf{e}_j^T \mathbf{v}_1} = \lambda_1.$$

Inoltre, dalla (1.11), sempre poichè $|\lambda_i/\lambda_1| < 1$ per ogni i, segue

$$\lim_{k\to\infty} \frac{1}{\lambda_1^k}\, \mathbf{x}^{(k)} = \alpha_1 \mathbf{v}_1.$$

Quest'ultima relazione ci informa che per k sufficientemente grande, $\mathbf{x}^{(k)}$ tende all'autovettore $\mathbf{v}_1$.
Si deve considerare che, se $|\lambda_1| > 1$, si ha

$$\mathbf{x}^{(k)} \simeq |\lambda_1|^k \alpha_1 \mathbf{v}_1 \longrightarrow \infty$$

pertanto quando l'algoritmo viene implementato così come descritto possono nascere facilmente problemi di overflow (o analogamente problemi di underflow se risulta $|\lambda_1| < 1$).
È possibile evitare queste circostanze osservando che in realtà siamo interessati al rapporto $\mathbf{e}_j^T \mathbf{x}^{(k+1)} / \mathbf{e}_j^T \mathbf{x}^{(k)}$ e dunque è possibile normalizzare la successione $\mathbf{x}^{(k)}$ lasciando inalterata tale quantità. È possibile allora modificare l'algoritmo descritto nel seguente modo:
1. Si sceglie un vettore $\mathbf{x}^{(0)}$ tale che

$$\|\mathbf{x}^{(0)}\|_\infty = 1;$$

2. Si calcola il vettore

$$\mathbf{y}^{(k+1)} = A\mathbf{x}^{(k)};$$

3. Si pone

$$\lambda_1^{(k+1)} = \frac{e_{i_k}^T \mathbf{y}^{(k+1)}}{e_{i_k}^T \mathbf{x}^{(k)}}$$

4. Si calcola il vettore

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k+1)}}{\|\mathbf{y}^{(k+1)}\|_\infty}$$

Il processo iterativo viene fermato quando due elementi della successione sono sufficentemente vicini, cioè risulta

$$|\lambda_1^{(k+1)} - \lambda_1^{(k)}| \leq \varepsilon$$

dove ε è la precisione fissata.
L'algoritmo codificato in MatLab è il seguente (A è una matrice di ordine n mentre `tol` è la precisione fissata):

```
function [lambda1,k] = potenze(A,tol)
%
% [lambda1,k] = potenze(A)
%
% Parametri di input
% A = matrice quadrata di ordine n
% tol = tolleranza voluta
%
% Parametri di output
% lambda1 = approssimazione dell'autocalore dominante
%   Posto uguale a inf se il metodo non converge
%   dopo 500 iterazioni
% Num_it = numero di iterazioni
%   Posto uguale a inf se il metodo non converge
%
n = size(A,1);
%
% Se la tolleranza non e' un parametro specificato
% in input allora tol = 1e-8 per default
%
if nargin == 1
    tol = 1e-8;
end
%
% Si fissa a 500 il numero massimo di iterazioni
%
Max_it = 500;
error = 2*tol;
x = ones(n,1);
lambda0 = 0;
Num_it = 0;
while error > tol
    y = A*x;
    [norma,k] = max(abs(y));
    lambda1 = y(k)/x(k);
    x = y/norma;
    error = abs(lambda1-lambda0);
    lambda0 = lambda1;
```

```
    Num_it = Num_it+1;
    if Num_it > Max_it
        Num_it = inf;
        lambda1 = inf;
        return
    end
end
```

Nell'algoritmo possiamo scegliere ad ogni passo l'indice del vettore i_k tale che

$$|e_{i_k}^T \mathbf{x}^{(k)}| = \|\mathbf{x}^{(k)}\|_\infty = 1.$$

In questo caso i_k può cambiare ad ogni passo, sarebbe improprio applicare la relazione (1.8), tuttavia è possibile dimostrare che la convergenza della successione $\lambda_1^{(k)}$ è garantita.
Se α_1, componente di $\mathbf{x}^{(0)}$ su $\mathbf{v}_1$, è nulla e

$$|\lambda_2| > |\lambda_3|$$

allora il processo (1.8) dovrebbe, in teoria, convergere a λ_2. Tuttavia a causa degli errori di arrotondamento, accade sempre che $\alpha_1 \neq 0$ e dunque il metodo converge di fatto a λ_1.
Se

$$|\lambda_2| \simeq |\lambda_1|$$

allora, avendo osservato che la convergenza della successione $\{\lambda_1^{(k)}\}$ dipende dal rapporto

$$\frac{|\lambda_2|}{|\lambda_1|},$$

la velocità di convergenza risulterà particolarmente lenta.
Proviamo ora che $\lambda_1^{(k)}$ calcolato al punto 3. è lo stesso definito nell'algoritmo non normalizzato. Per comodità, definiamo $\mathbf{x}^{(k)}$ del punto 4. come $\mathbf{x}_{new}^{(k)}$, mentre quello prodotto dall'algoritmo non normalizzato lo chiameremo $\mathbf{x}_{old}^{(k)}$.

Allora si ha, per $k = 0$:

$$\begin{aligned}
\mathbf{y}^{(1)} &= A\mathbf{x}^{(0)} \\
\mathbf{y}^{(2)} &= A\mathbf{x}^{(1)} = A\frac{\mathbf{y}^{(1)}}{\|\mathbf{y}^{(1)}\|_\infty} = \frac{A^2\mathbf{x}^{(0)}}{\|\mathbf{y}^{(1)}\|_\infty} \\
\mathbf{y}^{(3)} &= A\mathbf{x}^{(2)} = A\frac{\mathbf{y}^{(2)}}{\|\mathbf{y}^{(2)}\|_\infty} = \frac{A^3\mathbf{x}^{(0)}}{\|\mathbf{y}^{(1)}\|_\infty\|\mathbf{y}^{(2)}\|_\infty}.
\end{aligned}$$

In generale

$$\mathbf{y}^{(k+1)} = \frac{A^{k+1}\mathbf{x}^{(0)}}{\prod_{j=1}^{k}\|\mathbf{y}^{(j)}\|_\infty} = \frac{\mathbf{x}_{old}^{(k+1)}}{\prod_{j=1}^{k}\|\mathbf{y}^{(j)}\|_\infty}.$$

Perciò:

$$\begin{aligned}
\frac{(\mathbf{y}^{(k+1)})_{i_0}}{(\mathbf{x}_{new}^{(k)})_{i_0}} &= \frac{(\mathbf{x}_{old}^{(k+1)})_{i_0}}{\prod_{j=1}^{k}\|\mathbf{y}^{(j)}\|_\infty}\frac{1}{(\mathbf{x}_{new}^{(k)})_{i_0}} = \frac{(\mathbf{x}_{old}^{(k+1)})_{i_0}}{\prod_{j=1}^{k}\|\mathbf{y}^{(j)}\|_\infty}\frac{\|\mathbf{y}^{(k)}\|_\infty}{(\mathbf{y}^{(k)})_{i_0}} \\
&= \frac{(\mathbf{x}_{old}^{(k+1)})_{i_0}}{\prod_{j=1}^{k}\|\mathbf{y}^{(j)}\|_\infty}\frac{\|\mathbf{y}^{(k)}\|_\infty\prod_{j=1}^{k-1}\|\mathbf{y}^{(j)}\|_\infty}{(\mathbf{x}_{old}^{(k)})_{i_0}} = \frac{(\mathbf{x}_{old}^{(k+1)})_{i_0}}{(\mathbf{x}_{old}^{(k)})_{i_0}}.
\end{aligned}$$

Nell'algoritmo la convergenza è garantita se $(\mathbf{x}^{(k)})_{i_0} \neq 0$ scegliendo i_0 tale che

$$(\mathbf{x}^{(k)})_{i_0} = \|\mathbf{x}^{(k)}\|_\infty = 1.$$

Poichè in questo caso i_0 può cambiare ad ogni passo, sarebbe improprio applicare la relazione (1.8). È tuttavia possibile provare che la convergenza della successione

$$\frac{(\mathbf{y}^{(k+1)})_{i_0}}{(\mathbf{x}^{(k)})_{i_0}}$$

è garantita.

1.2.1 Il Metodo delle Potenze Inverse

Quando la matrice A ha un solo autovalore (reale) λ_n di modulo minimo, eventualmente con molteplicità algebrica ≥ 1, allora è ancora possibile utilizzare il metodo delle potenze per definire una successione $\{\lambda_n^{(k)}\}$ convergente

a λ_n. Basta infatti osservare che se $\lambda \neq 0$ è un autovalore di A allora $\frac{1}{\lambda}$ è autovalore di A^{-1}. La dimostrazione è semplicissima:

$$A\mathbf{x} = \lambda\mathbf{x} \;\Rightarrow\; \mathbf{x} = A^{-1}\lambda\mathbf{x} = \lambda A^{-1}\mathbf{x} \;\Rightarrow\; \frac{1}{\lambda}\mathbf{x} = A^{-1}\mathbf{x}.$$

Pertanto la determinazione dell'autovalore di modulo minimo di A si può effettuare calcolando l'autovalore di massimo modulo di A^{-1}.

1. Calcolare la fattorizzazione $A = LU$
2. $\mathbf{x}^{(0)} = (1, 1, \ldots, 1)^T$
3. for $k = 0, 1, 2, \ldots, K_{\max} - 1$
4. Risolvere $L\tilde{\mathbf{y}}^{(k+1)} = \mathbf{x}^{(k)}$
5. Risolvere $U\mathbf{y}^{(k+1)} = \tilde{\mathbf{y}}^{(k+1)}$
6. $\lambda_1^{(k+1)} = (\mathbf{y}^{(k+1)})_{i_0}/(\mathbf{x}^{(k)})_{i_0}$
7. $\mathbf{x}^{(k+1)} = \mathbf{y}^{(k+1)}/\|\mathbf{y}^{(k+1)}\|_\infty$
8. end

La Variante di Wielandt

Il metodo delle potenze può essere generalizzato per il calcolo di un particolare autovalore e del corrispondente autovettore non appena si conosca una buona approssimazione dell'autovalore stesso. Il metodo così modificato prende il nome di Metodo di Wielandt. Sia λ_c un'approssimazione dell'autovalore λ della matrice A, a cui corrisponde l'autovettore $\mathbf{x}$. Evidentemente

$$(A - \lambda_c I)\mathbf{x} = (\lambda - \lambda_c)\mathbf{x}$$

e dunque $\lambda - \lambda_c$ è autovalore della matrice $A - \lambda_c I$ con corrispondente autovettore $\mathbf{x}$; conseguentemente $(\lambda - \lambda_c)^{-1}$ è autovalore della matrice $(A - \lambda_c I)^{-1}$ e $\mathbf{x}$ è il corrispondente autovettore. Ne discende che se λ_c è una buona approssimazione a λ allora $\frac{1}{\lambda - \lambda_c}$ è l'autovalore di massimo modulo per $(A - \lambda_c I)^{-1}$. Il calcolodi

$$\mu = \frac{1}{\lambda - \lambda_c}$$

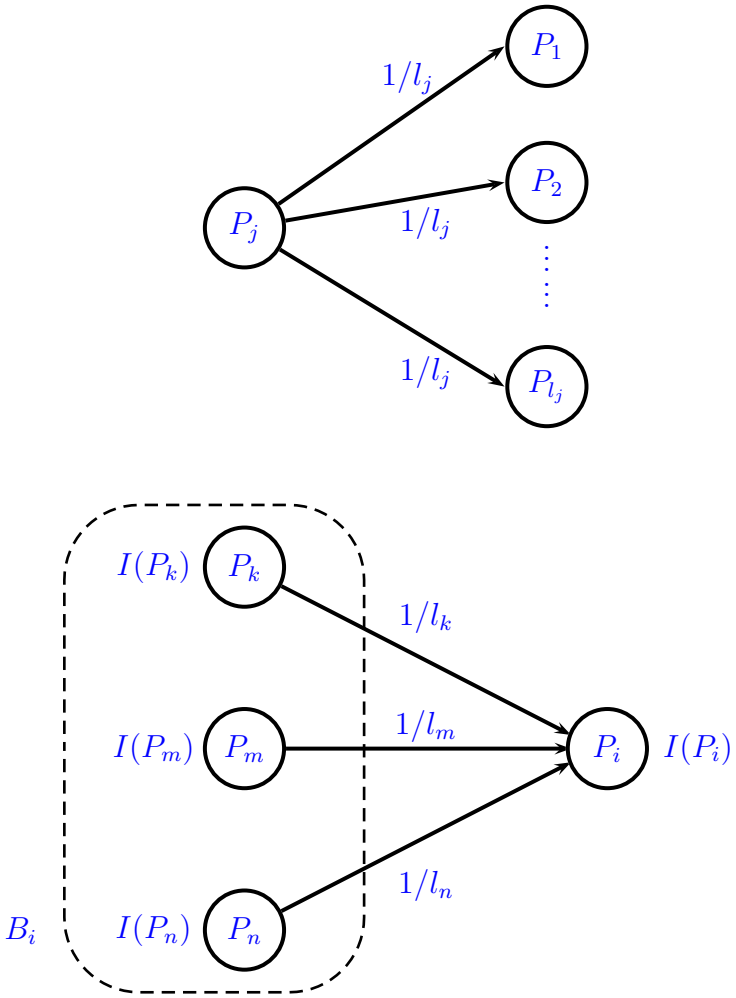
e quindi di

$$\lambda = \lambda_c + \frac{1}{\mu}$$

viene effettuato applicando il metodo delle potenze alla matrice $(A - \lambda_c I)^{-1}$.

1.2.2 L'algoritmo di PageRank di Google

Uno dei motivi per cui il metodo delle potenze è diventato particolarmente popolare è legato al fatto che costituisce il cuore di Google, il motore di ricerca più utilizzato nel web, ed in particolare al cosiddetto PageRank. Descriveremo brevemente in questo paragrafo le caratteristiche principali del PageRank ed il suo legame con il metodo delle potenze. Come è noto i motori di ricerca svolgono un'intensa, benchè invisibile, attività di monitoraggio delle pagine web, provvedendo ad indicizzare le parole più importanti che si trovano su un determinato documento e immagazzinando opportunamente tale informazione. Una volta che nel motore di ricerca viene inserita una determinata frase esso provvede a trovare tutte le pagine che riportano tale frase. Poichè il numero di tali pagine risulta essere particolarmente elevato è necessario che il motore di ricerca fornisca una misura dell'importanza di tali pagine in modo tale che queste siano visualizzate all'utente in base al valore di tale parametro. Lo scopo principale dell'algoritmo di PageRank è proprio quello di misurare il valore di tale parametro.
È chiaro che ogni pagina web contiene una serie di link ad altre pagine web, quindi l'idea alla base di PageRank è che l'importanza di una pagina dipende:
1) dalle pagine che hanno un link a tale pagina;
2) dall'importanza delle suddette pagine.
Il valore di PageRank viene valutato settimanalmente applicando l'algoritmo che ci apprestiamo a descrivere (aggiornamenti più frequenti sono impossibili a causa dell'enorme quantità di pagine pubblicate sul web). Consideriamo quindi una pagina web, che indichiamo con P_j, e che supponiamo abbia un numero di link pari ad l_j. Se uno di questi link è diretto verso la pagina P_i allora si può affermare che P_i ha un'importanza rispetto a P_j pari a $1/l_j$.



L'importanza di una pagina P_i, che indichiamo con $I(P_i)$, può essere calcolato come la somma di tutti i contributi delle pagine che hanno un link verso P_i. Se indichiamo con B_i l'insieme delle pagine che hanno un link alla pagina P_i allora possiamo definire la seguente formula:

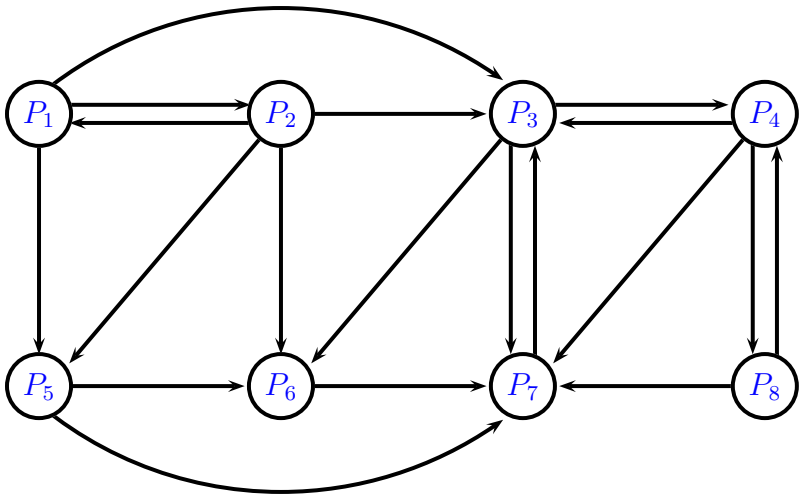
$$I(P_i) = \sum_{P_j \in B_i} \frac{I(P_j)}{l_j}. \tag{1.13}$$

Chiaramente l'importanza di una pagina dipende dall'importanza delle pagine che consentono di arrivarci. Per vedere quale relazione lega tali quantità con il metodo delle potenze dobbiamo riformulare la definizione appena vista

utilizzando una matrice H, detta Matrice di Hyperlink (Hyperlink Matrix), con elementi:

$$h_{ij} = \begin{cases} 1/l_j & \text{se } P_j \in B_i, \\ 0 & \text{altrimenti.} \end{cases}$$

La matrice H ha tutti elementi nonnegativi ed inoltre la somma degli elementi che si trovano sulla stessa colonna è uguale a uno. Vediamo un esempio, considerando otto pagine web connesse nel seguente modo:



La Hyperlink matrix è la seguente:

$$H = \begin{bmatrix} 0 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/4 & 0 & 1/3 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/2 \\ 1/3 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 1/3 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/2 & 1 & 0 & 1/2 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Per esempio risulta

$$I(P_1) = I(P_2)/4$$

oppure

$$I(P_3) = I(P_1)/3 + I(P_2)/4 + I(P_4)/3 + I(P_7).$$

In base alla relazione (1.13), detto $\mathbf{x}$ il vettore

$$\mathbf{x} = [I(P_1), I(P_2), \ldots, I(P_8)]^T$$

è verificata la seguente uguaglianza

$$\mathbf{x} = H\mathbf{x}$$

e quindi $\mathbf{x}$ è un autovettore della matrice di Hyperlink associato all'autovalore 1. Per calcolare il vettore $\mathbf{x}$ si può applicare il metodo delle potenze, in quanto è possibile verificare che 1 è proprio l'autovalore dominante di H, definendo il processo iterativo

$$\mathbf{x}^{(k+1)} = H\mathbf{x}^{(k)}.$$

Nella pratica, per ottenere la convergenza certa del metodo, si applica il metodo delle potenze non alla matrice di Hyperlink, ma alla cosiddetta Matrice di Google:

$$G = \alpha H + (1 - \alpha)\frac{U}{n}$$

dove n è il numero delle pagine web, U è una matrice quadrata di ordine n i cui elementi sono tutti uguali a uno ed α è un numero casuale compreso tra 0 e 1.

Capitolo 2

Approssimazione ai minimi quadrati

2.1 Introduzione

Uno dei problemi più frequenti che si affronta nel campo ingegneristico è il seguente. Supponiamo di dover effettuare una misura di una grandezza incognita utilizzando uno strumento di misura e facendo variare una determinata grandezza (oppure al variare del tempo). Se è nota la legge che lega le due grandezze in gioco (per esempio esse potrebbero essere l'intensità di corrente e la differenza di potenziale di un circuito, oppure la pressione e la temperatura di un materiale durante un determinato esperimento) si può notare che raramente le misure da noi effettuate obbediscono a tale legge matematica. Questo perchè le misure effettuate con strumenti sono sempre soggette ad errori dovuti alle più varie cause (errore di parallasse dovuto ad un'errata posizione nella lettura dello strumento oppure a problemi interni allo strumento come l'effetto di campi elettromagnetici oppure ad una cattiva taratura del fondo scala). Sostanzialmente il problema è simile a quello dell'interpolazione cioè bisogna trovare una funzione $\Psi(x)$ che però approssimi le coppie di dati (x_i, y_i) ma che non passi per questi poichè non sono dati precisi:

$$\Psi(x_i) \simeq y_i.$$

Il problema dell'approssimazione polinomiale prevede che la funzione $\Psi(x)$ sia un polinomio di grado opportuno (generalmente inferiore di molto rispetto al numero di dati disponibili e questa è un'ulteriore differenza rispetto

al problema dell'interpolazione). Un ulteriore grado di libertà sta nel fatto che si deve decidere come tradurre matematicamente il concetto di approssimazione, cioè come misurare la differenza tra i valori noti y_i e quelli che li approssimano, cioè $\Psi(x_i)$. Vedremo che una delle tecniche più usate nel campo dell'approssimazione è la cosiddetta approssimazione polinomiale ai minimi quadrati. Prima di vedere di cosa si tratta introduciamo un importante algoritmo di algebra lineare che è la fattorizzazione QR di matrici rettangolari reali e le proprietà principali delle matrici ortogonali.

2.2 Matrici Ortogonali

Definizione 2.2.1 *Una matrice $Q \in \mathbb{R}^{n\times n}$ si dice ortogonale se:*

$$QQ^T = Q^TQ = I_n$$

dove Q^T è la matrice trasposta ed I_n è la matrice identità di ordine n.

Dalla definizione di matrice ortogonale segue come conseguenza immediata che l'inversa coincide con la matrice trasposta:

$$Q^{-1} = Q^T.$$

Se riscriviamo la matrice Q per colonne

$$Q = [\mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3\ \ldots\ \mathbf{q}_n]$$

cioè in modo tale che $\mathbf{q}_i$ sia la $i-$esima colonna di Q, allora poichè $Q^TQ = I_n$ risulta

$$\begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \ldots & \mathbf{q}_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1^T\mathbf{q}_1 & \mathbf{q}_1^T\mathbf{q}_2 & \ldots & \mathbf{q}_1^T\mathbf{q}_n \\ \mathbf{q}_2^T\mathbf{q}_1 & \mathbf{q}_2^T\mathbf{q}_2 & \ldots & \mathbf{q}_2^T\mathbf{q}_n \\ \vdots & \vdots & & \vdots \\ \mathbf{q}_n^T\mathbf{q}_1 & \mathbf{q}_n^T\mathbf{q}_2 & \ldots & \mathbf{q}_n^T\mathbf{q}_n \end{bmatrix} = I_n$$

Quindi deve essere necessariamente

$$\mathbf{q}_i^T\mathbf{q}_j = (\mathbf{q}_j, \mathbf{q}_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Quindi una matrice ortogonale è caratterizzata dalla proprietà che tutti i suoi vettori colonna sono a due a due ortogonali ed ogni colonna ha norma euclidea uguale a 1. Una conseguenza di questa proprietà è che tutti gli elementi di una matrice ortogonale sono, in modulo, minori di 1. Infatti

$$\mathbf{q}_i^T \mathbf{q}_i = \sum_{j=1}^{n} q_{ji}^2 = 1.$$

Il discorso potrebbe essere ripetuto anche per le righe che risultano essere a due a due ortogonali e aventi norma euclidea uguale a 1.
Se Q_1 e Q_2 sono matrici ortogonali di ordine n allora anche la matrice

$$Q = Q_1 Q_2$$

è ortogonale. Infatti

$$QQ^T = Q_1 Q_2 (Q_1 Q_2)^T = Q_1 Q_2 Q_2^T Q_1^T = Q_1 Q_1^T = I_n.$$

Si dice che l'insieme delle matrici è chiuso rispetto all'operazione di moltiplicazione, cioè forma un Gruppo. Se Q_1 è una matrice ortogonale di ordine n e Q_2 è una matrice ortogonale di ordine m allora anche le seguenti matrici di ordine $n+m$ sono ortogonali.

$$\begin{bmatrix} O & Q_1 \\ Q_2 & O \end{bmatrix} \qquad \begin{bmatrix} Q_1 & O \\ O & Q_2 \end{bmatrix}.$$

Se Q è una matrice ortogonale di ordine n ed $\mathbf{x} \in \mathbb{R}^n$ allora

$$\|Q\mathbf{x}\|_2 = \|\mathbf{x}\|_2.$$

Infatti

$$\|Q\mathbf{x}\|_2^2 = (Q\mathbf{x}, Q\mathbf{x}) = (Q\mathbf{x})^T Q\mathbf{x} = \mathbf{x}^T Q^T Q\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|_2^2.$$

Tra le matrici ortogonali quelle che hanno una maggiore importanza nelle applicazioni sono le Matrici di Householder e le Matrici di Givens.

Le Matrici di Householder

Una matrice di Householder P di ordine n è definita nel seguente modo

$$P = I - \beta \mathbf{u}\mathbf{u}^T, \qquad \beta \in \mathbb{R}, \mathbf{u} \in \mathbb{R}^n.$$

Appare evidente che una matrice di questo tipo è simmetrica ($P^T = P$), e imponiamo ora che sia anche ortogonale, cioè che

$$PP^T = (I - \beta \mathbf{u}\mathbf{u}^T)(I - \beta \mathbf{u}\mathbf{u}^T) = I$$

e quindi

$$\begin{aligned} PP^T &= I - 2\beta \mathbf{u}\mathbf{u}^T + \beta^2 \mathbf{u}(\mathbf{u}^T\mathbf{u})\mathbf{u}^T = I - 2\beta \mathbf{u}\mathbf{u}^T + \beta^2 \|\mathbf{u}\|_2^2 \mathbf{u}\mathbf{u}^T \\ &= I + (\beta^2 \|\mathbf{u}\|_2^2 - 2\beta)\mathbf{u}\mathbf{u}^T = I + \beta(\beta\|\mathbf{u}\|_2^2 - 2)\mathbf{u}\mathbf{u}^T. \end{aligned}$$

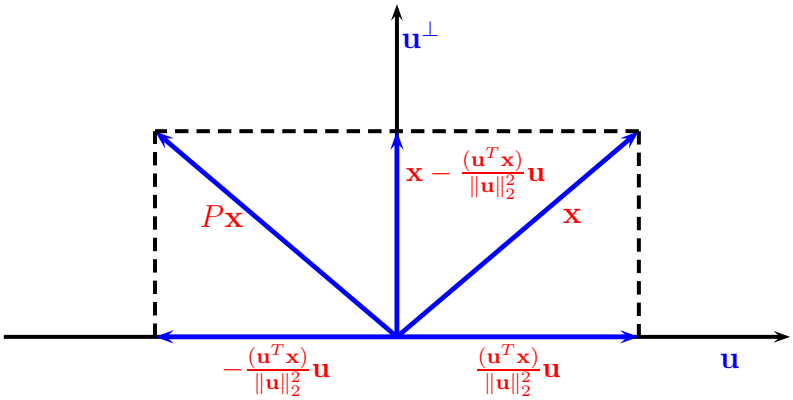
I valori di β che rendono P ortogonale sono due ma la soluzione $\beta = 0$ rende P coincidente con la matrice identità, e quindi di scarso interesse pratico. Il valore accettabile per β è

$$\beta = \frac{2}{\|\mathbf{u}\|_2^2}. \tag{2.1}$$

Le matrici di Householder sono dette anche matrici di riflessione poichè il vettore $P\mathbf{x}$ risulta essere il vettore riflesso di $\mathbf{x}$ rispetto allo spazio vettoriale perpendicolare al vettore $\mathbf{u}$, infatti

$$P\mathbf{x} = \mathbf{x} - \frac{\mathbf{u}^T\mathbf{x}}{\|\mathbf{u}\|_2^2}\mathbf{u} - \frac{\mathbf{u}^T\mathbf{x}}{\|\mathbf{u}\|_2^2}\mathbf{u}$$

e, come rappresentando tali vettori nella seguente figura, risulta chiara la relazione tra $\mathbf{x}$ e $P\mathbf{x}$.



Le Matrici di Givens

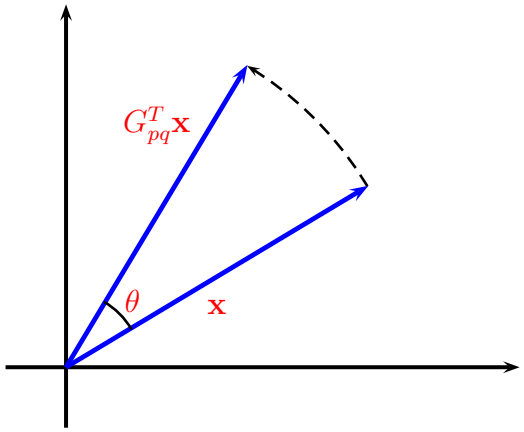
Fissati due numeri interi p, q compresi tra 1 ed n e tali che $p < q$ ed un angolo θ, poniamo

$$c = \cos\theta, \qquad s = \sin\theta$$

e definiamo la matrice di Givens G_{pq} nel seguente modo:

$$G_{pq} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ \hline & & c & & s & & \\ \hline & & & 1 & & & \\ & & & & \ddots & & \\ & & & & & 1 & \\ \hline & & -s & & c & & \\ \hline & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \text{riga } p \\ \\ \\ \\ \text{riga } q \\ \\ \\ \\ \end{matrix} \tag{2.2}$$

Solo le righe (e le colonne) di indice p e q differiscono dalla matrice identità. La verifica dell'ortogonalità di G_{pq} è piuttosto immediata.
Le matrici di Givens sono dette anche matrici di rotazione poichè il vettore $G_{pq}^T\mathbf{x}$ risulta essere il risultato della rotazione di $\mathbf{x}$ in senso antiorario di un angolo pari a θ.



2.3 Fattorizzazione QR

Assegnata una matrice $A \in \mathbb{R}^{m\times n}$, $m \geq n$, ci poniamo il problema di determinare una matrice $Q \in \mathbb{R}^{m\times m}$ ed una matrice $R \in \mathbb{R}^{m\times n}$ tali che:

$$A = QR \tag{2.3}$$

con Q matrice ortogonale, cioè $Q^TQ = I$, mentre R ha la seguente struttura

$$R = \begin{bmatrix} R_1 \\ \mathrm{O} \end{bmatrix},$$

dove $R_1 \in \mathbb{R}^{n\times n}$ è una matrice triangolare superiore. La fattorizzazione (2.3) prende il nome di fattorizzazione QR (oppure fattorizzazione di Householder) della matrice A.
Se $m = n$ e $\det A \neq 0$ allora si può risolvere il sistema $A\mathbf{x} = \mathbf{b}$ nel modo seguente:

$$A\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad QR\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad R\mathbf{x} = \mathbf{c} \qquad \text{con } \mathbf{c} = Q^T\mathbf{b}$$

e dunque per calcolare il vettore $\mathbf{x}$ basta risolvere un sistema triangolare superiore. Vedremo successivamente che tale fattorizzazione risolve brillantemente il problema della risoluzione del sistema $A\mathbf{x} = \mathbf{b}$, $A \in \mathbb{R}^{m\times n}$, $m > n$. Se la matrice A ha rango massimo allora la matrice triangolare superiore R_1 è invertibile.
Consideriamo preliminarmente il seguente problema: assegnato un vettore $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{x} \neq 0$, determinare una matrice di Householder P tale che

$$P\mathbf{x} = k\mathbf{e}_1 \qquad k \in \mathbb{R},$$

dove $\mathbf{e}_1$ è il primo versore fondamentale di $\mathbb{R}^m$. Quindi

$$P\mathbf{x} = (I - \beta\mathbf{u}\mathbf{u}^T)\mathbf{x} = \mathbf{x} - \beta\mathbf{u}\mathbf{u}^T\mathbf{x} = \mathbf{x} - \beta(\mathbf{u}^T\mathbf{x})\mathbf{u}.$$

Poichè il vettore $\mathbf{u}$, è arbitrario possiamo imporre

$$\beta\mathbf{u}^T\mathbf{x} = 1 \tag{2.4}$$

cosicchè

$$\mathbf{x} - \mathbf{u} = k\mathbf{e}_1$$

quindi deve essere necessariamente $u_i = x_i$, $i = 2, \ldots, m$, e $k = x_1 - u_1$. Determiniamo ora l'espressione dell'elemento u_1. Da (2.1) e (2.4) abbiamo:

$$\frac{2}{\|\mathbf{u}\|_2^2}\mathbf{u}^T\mathbf{x} = 1$$

$$\|\mathbf{u}\|_2^2 - 2\mathbf{u}^T\mathbf{x} = 0$$

$$u_1^2 + \sum_{i=2}^{m} x_i^2 - 2u_1x_1 - 2\sum_{i=2}^{m} x_i^2 = 0$$

$$u_1^2 - 2x_1u_1 - \sum_{i=2}^{m} x_i^2 = 0.$$

Risolvendo l'equazione di secondo grado si ottengono due soluzioni:

$$u_1 = x_1 \pm \sqrt{x_1^2 + \sum_{i=2}^{m} x_i^2} = x_1 \pm \|\mathbf{x}\|_2.$$

Per evitare la cancellazione di cifre significative (vedere paragrafo 2.3.1) si sceglie il segno in modo tale da evitare l'operazione di sottrazione, cioè

$$u_1 = x_1 + segn(x_1)\|\mathbf{x}\|_2.$$

La (2.1) può anche scriversi:

$$\begin{aligned}\beta &= \frac{2}{\|\mathbf{u}\|_2^2} = \frac{2}{u_1^2 + \sum_{i=2}^{m} x_i^2} = \frac{2}{(x_1 + segn(x_1)\|\mathbf{x}\|_2)^2 + \sum_{i=2}^{m} x_i^2} \\ &= \frac{2}{x_1^2 + \|\mathbf{x}\|_2^2 + 2|x_1|\|\mathbf{x}\|_2 + \sum_{i=2}^{m} x_i^2} \\ &= \frac{2}{2\|\mathbf{x}\|_2^2 + 2|x_1|\|\mathbf{x}\|_2} = \frac{1}{\|\mathbf{x}\|_2(|x_1| + \|\mathbf{x}\|_2)}.\end{aligned}$$

In definitiva scegliendo

$$\beta = \frac{1}{\|\mathbf{x}\|_2(|x_1| + \|\mathbf{x}\|_2)} \qquad \text{e} \qquad \mathbf{u} = \begin{bmatrix} x_1 + segn(x_1)\|\mathbf{x}\|_2 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

risulta

$$P\mathbf{x} = (I - \beta\mathbf{u}\mathbf{u}^T)\mathbf{x} = -segn(x_1)\|\mathbf{x}\|_2\mathbf{e}_1.$$

Sfruttiamo la proprieta appena trovata per calcolare la fattorizzazione QR della matrice rettangolare A. Per semplicità consideriamo il caso in cui $A \in \mathbb{R}^{7\times 5}$:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}.$$

Si conseidera come vettore $\mathbf{x}$ la prima colonna della matrice (evidenziata in rosso) e si trova la matrice di Householder P_1 che rende nulli tutti gli elementi tranne il primo. Moliplicando P_1 per A l'effetto è quello di aver reso nulli tutti gli elementi sottodiagonali della prima colonna di A:

$$P_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \\ 0 & \otimes & \times & \times & \times \end{bmatrix}.$$

A questo punto si considera come vettore $\mathbf{x}$ quello composto da tutti gli elementi evidenziati con $\otimes$ e si determina la matrice di Householder $\widehat{P}_2$ che azzera tutti tali elementi, tranne il primo. Poichè tale matrice ha ordine 6 allora per applicarla a P_1A si definisce la matrice

$$P_2 = \begin{bmatrix} 1 & \mathbf{o} \\ \mathbf{o}^T & \widehat{P}_2 \end{bmatrix}$$

cosicchè

$$P_2P_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \end{bmatrix}.$$

Se $A \in \mathbb{R}^{m\times n}$ allora al $k-$esimo passo si definisce la matrice

$$P_k = \begin{bmatrix} I_{k-1} & O \\ O^T & \widehat{P}_k \end{bmatrix}$$

con $\widehat{P}_k$ matrice di Householder di ordine $m-k+1$ e I_{k-1} matrice identità di ordine $k-1$. È facile osservare che le matrici P_k sono ortogonali per ogni k, in conseguenza di una proprietà vista in precedenza.
Se $m > n$ si devono applicare n matrici P_k cosicchè

$$R = P_nP_{n-1}\dots P_2P_1A$$

è triangolare superiore. La matrice A può essere scritta come

$$A = (P_nP_{n-1}\dots P_1)^{-1}\,R = \left(P_1^{-1}P_2^{-1}\dots P_n^{-1}\right)R = P_1P_2\dots P_nR,$$

poichè tutte le matrici P_k sono ortogonali e simmetriche (quindi $P_k^{-1} = P_k$). Posto

$$Q = P_1P_2\dots P_{n-1}P_n,$$

tale matrice è ortogonale in quanto prodotto di n matrici ortogonali, quindi A può essere fattorizzata come

$$A = QR.$$

Osserviamo che ad ogni passo la matrice P_k moltiplica la matrice al passo precedente che è già stata parzialmente triangolarizzata. Osserviamo che il numero di operazioni algebriche necessarie a calcolare tali colonne non è molto elevato. Infatti se P è una matrice di Householder ed y è un vettore diverso da $\mathbf{u}$ allora

$$P\mathbf{y} = \mathbf{y} - (\beta\mathbf{u}^T\mathbf{y})\mathbf{u}.$$

Per calcolare $\mathbf{u}^T\mathbf{y}$ sono necessarie $2m$ operazioni aritmetiche (β si suppone che sia già stato calcolato), poi sono necessari m prodotti per calcolare $(\beta\mathbf{u}^T\mathbf{y})\mathbf{u}$ e poi m differenze. In tutto sono richieste circa $4m$ operazioni mentre per calcolare il prodotto tra una matrice di ordine m ed un vettore di ordine m sono necessarie $2m^2$ operazioni aritmetiche. Ricordiamo che il costo computazionale di un algoritmo (cioè il numero di operazioni che richiede) è uno dei parametri da considerare quando si valuta l'efficienza di un algoritmo. L'algoritmo codificato in MatLab è il seguente (A è una matrice di $m \times n$):

```
function [Q,R] = myqr(A)
%
% [Q,R] = myqr(A)
%
% Calcola la fattorizzazione QR della matrice rettangolare
% A applicando le matrici di Householder
%
% Parametri di input
% A = matrice rettangolare di dimensione m x n
%
% Parametri di output
% Q = matrice ortogonale m x m
% R = matrice triangolare superiore m x n
%
[m,n] = size(A)
Q = eye(m)
for k=1:n
    u = A(k:m,k);
    norma = norm(u,2);
    u(1) = u(1)+sign(u(1))*norma;
    beta = 2/norm(u,2)^2;
    U = eye(m-k+1)-beta*u*u';
    P = [eye(k-1) zeros(k-1,m-k+1); zeros(m-k+1,k-1) U]
    A = P*A;
    Q = Q*P;
end
R = A;
return
```

2.3.1 La cancellazione di cifre significative

Ricordiamo che se x è un numero reale, la sua rappresentazione macchina, cioè il numero che approssima x nella memoria di un elaboratore, e che spesso viene indicata con $\mathrm{fl}(x)$, è un numero che è affetto da un certo errore (dipendente da x), tale che vale la seguente relazione:

$$\mathrm{fl}(x) = x(1 + \varepsilon_x)$$

dove ε_x rappresenta l'errore relativo ed è maggiorato, in modulo, dalla precisione macchina u (che dipende chiaramente dalla memoria dell'elaboratore e rappresenta il più piccolo numero reale rappresentabile). Chiaramente l'errore di rappresentazione di un numero reale può influire nella precisione dei risultati forniti dall'elaboratore poichè esso si propaga nei calcoli ed i calcoli stessi tendono ad introdurre ulteriori errori di rappresentazione. In questo paragrafo sarà affrontato un caso particolare legato all'operazione della differenza tra numeri reali, come effettuata all'interno dell'elaboratore. Supponiamo quindi di voler calcolare la differenza tra due numeri reali a e b. Siano $fl(a)$ e $fl(b)$ rispettivamente le loro rappresentazioni di macchina. Vogliamo vedere quale è l'errore che viene commesso dall'elaboratore quando calcola $a - b$.

$$\begin{aligned} fl(a) \ominus fl(b) &= [fl(a) - fl(b)](1 + \varepsilon_1) = [a(1 + \varepsilon_2) - b(1 + \varepsilon_3)](1 + \varepsilon_1) \\ &= (a + a\varepsilon_2 - b - b\varepsilon_3)(1 + \varepsilon_1) \\ &= (a - b) + (a - b)\varepsilon_1 + a\varepsilon_2 - b\varepsilon_3 + a\varepsilon_1\varepsilon_2 - b\varepsilon_1\varepsilon_3. \end{aligned}$$

Una maggiorazione per l'errore relativo è la seguente

$$\begin{aligned} \frac{|(fl(a) \ominus fl(b)) - (a - b)|}{|a - b|} \leq \; & |\varepsilon_1| + \frac{|a|}{|a - b|}(|\varepsilon_2| + |\varepsilon_1||\varepsilon_2|) + \\ & + \frac{|b|}{|a - b|}(|\varepsilon_3| + |\varepsilon_1||\varepsilon_3|). \end{aligned} \tag{2.5}$$

Se a e b hanno segno opposto risulta

$$\max(|a|, |b|) \leq |a - b|$$

e dalla (2.5) segue la maggiorazione

$$\frac{|(fl(a) \ominus fl(b)) - (a-b)|}{|a-b|} \leq 3u + O(u^2)$$

dove u è la precisione di macchina.
Se a e b hanno lo stesso segno allora l'errore relativo può essere molto grande quanto più a e b sono vicini. Questo fenomeno prende il nome di cancellazione di cifre significative. Questa analisi indica che, nell'effettuare operazioni aritmetiche sarebbe opportuno evitare la differenza tra due numeri reali aventi lo stesso segno e vicini tra loro.
Per esemplificare il fenomeno appena descritto consideriamo il problema di calcolare (per esempio in MatLab) le radici dell'equazione di secondo grado

$$p(x) = ax^2 + bx + c$$

applicando la consueta formula

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{2.6}$$

In alternativa si potrebbe calcolare la radice più grande in modulo

$$r_1 = \frac{-b - segno(b)\sqrt{b^2 - 4ac}}{2a} \tag{2.7}$$

e poi, sfruttando la proprietà che il prodotto tra le radici è pari a c/a, ottenere la seconda radice ponendo

$$r_2 = \frac{c}{ar_1}. \tag{2.8}$$

Considerando il polinomio

$$p(x) = x^2 - (10^7 + 10^{-7})x + 1$$

che ammette come radici 10^7 e 10^{-7}, applicando le formule (2.6), si ottiene

$$x_1 = 10^7, \qquad x_2 = 9.9652e-008$$

mentre utilizzando le formule (2.7) e (2.8) i risultati sono esatti

$$r_1 = 10^7, \qquad r_2 = 10^{-7}.$$

Nel primo caso il calcolo della radice x_1 avviene effettuando la differenza tra due numeri (ovvero $-b$ e $\sqrt{b^2 - 4ac}$) che sono molto vicini tra loro e pertanto generano il suddetto fenomeno. Nel secondo caso non viene effettuata alcuna differenza e pertanto il risultato è corretto.

2.3.2 Calcolo della fattorizzazione QR con le matrici di Givens

Sia $\mathbf{x} \in \mathbb{R}^m$, ci poniamo il problema di determinare una matrice di Givens $G_{pq} \in \mathbb{R}^{m\times m}$, con p, q numeri interi tali che $1 \leq p < q \leq m$, e tale che il vettore $\mathbf{y}$ definito come

$$\mathbf{y} = G_{pq}\mathbf{x}$$

abbia la componente di indice q uguale a zero. Ovviamente supponiamo che $x_q \neq 0$ altrimenti potremmo prendere G_{pq} uguale alla matrice identità di ordine m. Calcolando il prodotto $G_{pq}\mathbf{x}$ si ottiene

$$\begin{bmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & c & & & s & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & -s & & & & c & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{p-1} \\ x_p \\ x_{p+1} \\ \vdots \\ x_{q-1} \\ x_q \\ x_{q+1} \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_{p-1} \\ cx_p + sx_q \\ x_{p+1} \\ \vdots \\ x_{q-1} \\ -sx_p + cx_q \\ x_{q+1} \\ \vdots \\ x_m \end{bmatrix}.$$

Uguagliando le componenti al vettore $\mathbf{y}$ risulta:

$$\begin{aligned} y_i &= x_i, && i \neq p, q \\ y_p &= cx_p + sx_q \\ y_q &= -sx_p + cx_q. \end{aligned}$$

Il problema da risolvere è quello di trovare i valori di c, s $(c^2 + s^2 = 1)$ tali che

$$\begin{cases} cx_p + sx_q = y_p \\ -sx_p + cx_q = y_q = 0. \end{cases}$$

Prima di risolvere il sistema troviamo il valore di y_p. Ricordiamo che poichè la matrice G_{pq} è ortogonale allora i vettori $\mathbf{x}$ e $\mathbf{y}$ hanno la stessa norma 2:

$$\|\mathbf{y}\|_2^2 = \|G_{pq}\mathbf{x}\|_2^2 = \|\mathbf{x}\|_2^2.$$

Quindi

$$\sum_{i=1}^{m} y_i^2 = \sum_{i=1, i\neq p,q}^{m} y_i^2 + y_p^2 + y_q^2 = \sum_{i=1}^{m} x_i^2,$$

e poichè $x_i = y_i$ se $i \neq p, q$, e $y_q = 0$ allora

$$\sum_{i=1, i\neq p,q}^{m} x_i^2 + y_p^2 = \sum_{i=1, i\neq p,q}^{m} x_i^2 + x_p^2 + x_q^2.$$

Da cui si ottiene

$$y_p^2 = x_p^2 + x_q^2 \qquad \Rightarrow \qquad y_p = \sqrt{x_p^2 + x_q^2}$$

Il sistema da risolvere diviene

$$\begin{cases} cx_p + sx_q = \sqrt{x_p^2 + x_q^2} \\ \\ cx_q - sx_p = 0. \end{cases}$$

Applicando la regola di Cramer si trova

$$c = \frac{\begin{vmatrix} \sqrt{x_p^2 + x_q^2} & x_q \\ \\ 0 & -x_p \end{vmatrix}}{\begin{vmatrix} x_p & x_q \\ \\ x_q & -x_p \end{vmatrix}} = \frac{-x_p\sqrt{x_p^2 + x_q^2}}{-x_p^2 - x_q^2} = \frac{x_p\sqrt{x_p^2 + x_q^2}}{x_p^2 + x_q^2} = \frac{x_p}{\sqrt{x_p^2 + x_q^2}}.$$

$$s = \frac{\begin{vmatrix} x_p & \sqrt{x_p^2 + x_q^2} \\ \\ x_q & 0 \end{vmatrix}}{\begin{vmatrix} x_p & x_q \\ \\ x_q & -x_p \end{vmatrix}} = \frac{-x_q\sqrt{x_p^2 + x_q^2}}{-x_p^2 - x_q^2} = \frac{x_q\sqrt{x_p^2 + x_q^2}}{x_p^2 + x_q^2} = \frac{x_q}{\sqrt{x_p^2 + x_q^2}}.$$

Di seguito viene riportato il codice MatLab che, accettando in input il vettore $\mathbf{x}$ egli indici interi p e q, produce in output la matrice di Givens G_{pq} ch azzera la $q-$esima componente del vettore.

```
function G=givens(p,q,x)
%
% G = givens(p,q,x)
%
% Funzione che calcola la rotazione di Givens che annulla la
% componente q-esima del vettore x
%
% Parametri di input
% p,q = indici interi con p<q
% x = vettore colonna di n componenti
%
% Parametri di output
% G = matrice di Givens di ordine n che annulla la
%     q-esima componente del vettore x
%
n = length(x);
G = eye(n);
c = x(p)/(sqrt(x(p)^2+x(q)^2));
s = x(q)/(sqrt(x(p)^2+x(q)^2));
G(p,p) = c;
G(p,q) = s;
G(q,p) = -s;
G(q,q) = c;
return
```

Le matrici di Givens possono essere usate per calcolare la fattorizzazione QR di una matrice A rettagolare utilizzandole per azzerare gli elementi della parte triangolare inferiore. Se A è una matrice $m \times n$ allora possiamo definire la matrice di Givens G_{1m} che annulla l'elemento di A di posto $(m, 1)$, quindi definire la matrice $G_{1,m-1}$ che annulla l'elemento della nuova matrice di posto $(m-1, 1)$ e così via fino alla matrice $G_{1,2}$. Per gli elementi delle successive colonne si procede allo stesso modo, finchè la matrice non è completamente triangolare. La matrice triangolare (se $m \geq n$) R è uguale a:

$$R = G_{n,n+1} G_{n,n+2} \dots G_{1,m-1} G_{1m} A$$

quindi

$$A = \left(G_{n,n+1} G_{n,n+2} \dots G_{1,m-1} G_{1m}\right)^{-1} R = QR$$

con

$$\begin{aligned} Q &= (G_{n,n+1}G_{n,n+2}\dots G_{1,m-1}G_{1m})^{-1} \\ &= G_{1m}^T G_{1,m-1}^T \dots G_{n,n+2}^T G_{n,n+1}^T \end{aligned}$$

Come ultima osservazione si deve dire che, per calcolare la fattorizzazione QR, conviene utilizzare le matrici di Householder se la matrice A è piena, cioè molti suoi elementi sono diversi da zero, in quanto consente di annullare, in un singolo passo, tutti gli elementi appartenenti ad una singola porzione di colonna. Conviene utilizzare le matrici di Givens quando A è sparsa, cioè ha già una buona parte degli elementi uguali a zero, poichè consente di azzerare in modo selettivo tali elementi.

```
function [Q,R]=QRgivens(A)
%
% [Q,R] = QRgivens(A)
%
% Calcola la fattorizzazione QR della matrice rettangolare
% A applicando le matrici di Givens
%
% Parametri di input
% A = matrice rettangolare di dimensione m x n
%
% Parametri di output
% Q = matrice ortogonale m x m
% R = matrice triangolare superiore m x n
%
%
[m,n] = size(A)
Q = eye(m);
for j=1:n
    for i=m:-1:j+1
        if A(i,j) ~= 0
            G = givens(j,i,A(:,j));
            A = G*A;
            Q = Q*G';
        end
    end
end
```

```
R = A;
return
```

2.4 La Retta di Regressione

Come si è già accennato nell'introduzione di questo Capitolo quando i dati (x_i, y_i), $i = 0, \ldots, n$, sono rilevati con scarsa precisione, non ha molto senso cercare un polinomio di grado n (o, più in generale una funzione $\Psi(x)$) che interpoli i valori y_i nei nodi x_i. Può invece risultare più proficuo cercare un polinomio di grado $m < n$ che si avvicini il più possibile ai dati rilevati. I criteri che si possono scegliere per dare un senso alla frase si avvicini il più possibile possono essere diversi, per esempio si potrebbe cercare il polinomio $p_m(x)$ tale che

$$\max_i |p_m(x_i) - y_i| = \min_{p \in \Pi_m} \max_i |p(x_i) - y_i|$$

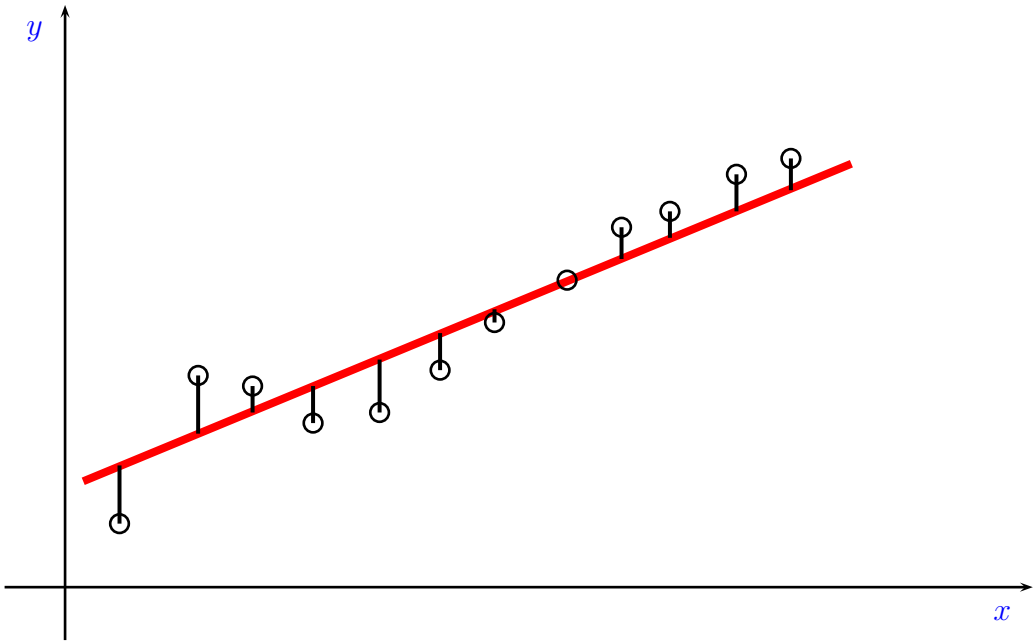
avendo indicato con Π_m l'insieme dei polinomi di grado al più m. Il tipo di approssimazione più utilizzato (e anche quello più semplice da risolvere) è quello polinomiale ai minimi quadrati. Un caso particolare è quello in cui si cerca una funzione lineare che approssima nel modo migliore i dati sperimentali. Quindi si pone

$$\Phi(x) = \alpha x + \beta, \qquad \alpha, \beta \in \mathbb{R} \tag{2.9}$$

e si cercano, tra tutte le possibili rette, i coefficienti α e β che globalmente minimizzano la differenza

$$\Phi(x_i) - y_i = \alpha x_i + \beta - y_i$$

La retta (2.9) che risolve tale problema viene detta Retta di regressione. Nella seguente figura sono evidenziate le quantità che devono essere globalmente minimizzate (i punti (x_i, y_i) sono evidenziati con il simbolo $\circ$).



Un modo per minimizzare globalmente le distanze della retta dalle approssimazioni è quello di trovare i valori α, β che minimizzano la funzione:

$$\Psi(\alpha, \beta) = \sum_{i=0}^{n} (\alpha x_i + \beta - y_i)^2 .$$

Per questo si parla di problema ai minimi quadrati (si minimizza una somma di quantità elevate al quadrato).
Per determinare tali valori calcoliamo le derivate parziali rispetto alle incognite:

$$\frac{\partial \Psi}{\partial \alpha} = 2 \sum_{i=0}^{n} x_i (\alpha x_i + \beta - y_i)$$

$$\frac{\partial \Psi}{\partial \beta} = 2 \sum_{i=0}^{n} (\alpha x_i + \beta - y_i)$$

$$\begin{cases} \dfrac{\partial \Psi}{\partial \alpha} = 2 \displaystyle\sum_{i=0}^{n} x_i (\alpha x_i + \beta - y_i) = 0 \\ \dfrac{\partial \Psi}{\partial \beta} = 2 \displaystyle\sum_{i=0}^{n} (\alpha x_i + \beta - y_i) = 0 \end{cases}$$

$$\begin{cases} \sum_{i=0}^{n} x_i \left(\alpha x_i + \beta - y_i\right) = 0 \\ \\ \sum_{i=0}^{n} \left(\alpha x_i + \beta - y_i\right) = 0 \end{cases}$$

$$\begin{cases} \alpha \sum_{i=0}^{n} x_i^2 + \beta \sum_{i=0}^{n} x_i - \sum_{i=0}^{n} x_i y_i = 0 \\ \\ \alpha \sum_{i=0}^{n} x_i + (n+1)\beta - \sum_{i=0}^{n} y_i = 0. \end{cases}$$

Poniamo per semplicità

$$S_{xx} = \sum_{i=0}^{n} x_i^2 \qquad S_x = \sum_{i=0}^{n} x_i$$

$$S_{xy} = \sum_{i=0}^{n} x_i y_i \qquad S_y = \sum_{i=0}^{n} y_i.$$

Il sistema diventa

$$\begin{cases} S_{xx}\alpha + S_x\beta = S_{xy} \\ \\ S_x\alpha + (n+1)\beta = S_y \end{cases}$$

la cui soluzione è

$$\alpha = \frac{(n+1)S_{xy} - S_x S_y}{(n+1)S_{xx} - S_x^2}$$

$$\beta = \frac{S_y S_{xx} - S_x S_{xy}}{(n+1)S_{xx} - S_x^2}.$$

La tecnica della retta di regressione può essere applicata anche nel caso in cui la relazione tra le ascisse x_i e le ordinate y_i sia di tipo esponenziale, ovvero si può ipotizzare che la funzione che meglio approssima i dati sperimentali sia

$$\Phi(x) = Be^{Ax}, \qquad A, B \in \mathbb{R}, B > 0.$$

Ponendo

$$Y = \log \Phi(x)$$

risulta

$$Y = \log(Be^{Ax}) = Ax + \log B$$

ovvero

$$Y = \alpha x + \beta, \qquad \alpha = A,\ \beta = \log B$$

quindi si può applicare la tecnica della retta di regressione ai dati $(x_i, \log y_i)$ (osserviamo che affinchè il modello abbia senso i valori y_i devono essere tutti strettamente positivi.

2.5 Approssimazione Polinomiale ai Minimi Quadrati

Esattamente nello stesso modo descritto nel precedente paragrafo si può cercare il polinomio di grado m che si avvicina ai dati sperimentali nel senso dei minimi quadrati, cioè si cerca il polinomio

$$p_m(x) = \sum_{j=0}^{m} a_j x^j,$$

di grado $m < n$, tale che:

$$\sum_{i=0}^{n} (p_m(x_i) - y_i)^2$$

sia minima. Cioè si vuole risolvere il problema di minimo:

$$\min_{a_0, a_1, \ldots, a_m} \sum_{i=0}^{n} \left(\sum_{j=0}^{m} a_j x_i^j - y_i \right)^2. \tag{2.10}$$

Inseriamo in un vettore i valori del polinomio $p_m(x)$ calcolati nelle ascisse x_i:

$$\begin{bmatrix} p_m(x_0) \\ p_m(x_1) \\ \vdots \\ p_m(x_n) \end{bmatrix} = \begin{bmatrix} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{m-1} x_0^{m-1} + a_m x_0^m \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{m-1} x_1^{m-1} + a_m x_1^m \\ \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_{m-1} x_n^{m-1} + a_m x_n^m \end{bmatrix}$$

Definendo la matrice $B \in \mathbb{R}^{(n+1)\times(m+1)}$ nel seguente modo:

$$B = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{m-1} & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^{m-1} & x_1^m \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{m-1} & x_n^m \end{bmatrix},$$

ed il vettore $\mathbf{a} = [a_0, a_1, \dots, a_m]^T$ si vede che risulta

$$\begin{bmatrix} p_m(x_0) \\ p_m(x_1) \\ \vdots \\ p_m(x_n) \end{bmatrix} = B\mathbf{a}$$

e quindi, definito il vettore $\mathbf{y} = [y_0, y_1, \dots, y_n]^T$ risulta

$$\begin{bmatrix} p_m(x_0) - y_0 \\ p_m(x_1) - y_1 \\ \vdots \\ p_m(x_n) - y_n \end{bmatrix} = B\mathbf{a} - \mathbf{y}.$$

Osserviamo che la quantità da minimizzare

$$\sum_{i=0}^{n} (p_m(x_i) - y_i)^2$$

corrisponde alla norma 2 al quadrato del vettore che ha componenti pari a $p_m(x_i) - y_i$ quindi il problema di minimo può essere riscritto in forma algebrica:

$$\min_{\mathbf{a}\in\mathbb{R}^{m+1}} \|B\mathbf{a} - \mathbf{y}\|_2^2. \tag{2.11}$$

Supponiamo ora che rango di B sia $m+1$, ricordando che, in questo caso, il rango di una matrice è il numero di colonne linearmente indipendenti. Tale ipotesi tuttavia non è restrittiva in questo caso poichè se i punti $x_0, x_1, \dots, x_n$ sono tutti distinti allora la matrice B ha sicuramente rango $m+1$ in quanto

ogni sottomatrice principale di ordine $m+1$ è una matrice di Vandermonde sicuramente non singolare. Consideriamo quindi la fattorizzazione QR della matrice B:

$$B = QR, \qquad Q \in \mathbb{R}^{(n+1)\times(n+1)}, R \in \mathbb{R}^{(n+1)\times(m+1)}$$

essendo Q una matrice ortogonale ed R con la seguente struttura:

$$R = \begin{bmatrix} R_1 \\ \mathrm{O} \end{bmatrix}$$

dove R_1 è una matrice triangolare superiore di ordine $m+1$. Sostituendo a B la sua fattorizzazione QR abbiamo:

$$\begin{aligned} \|B\mathbf{a} - \mathbf{y}\|_2^2 &= \|QR\mathbf{a} - \mathbf{y}\|_2^2 = \|Q(R\mathbf{a} - Q^T\mathbf{y})\|_2^2 \\ &= \|R\mathbf{a} - \mathbf{c}\|_2^2 \end{aligned}$$

avendo posto $\mathbf{c} = Q^T\mathbf{y}$.

$$\begin{aligned} \|B\mathbf{a} - \mathbf{y}\|_2^2 &= \left\| \begin{bmatrix} R_1 \\ \mathrm{O} \end{bmatrix} \mathbf{a} - \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \right\|_2^2 \\ &= \left\| \begin{bmatrix} R_1\mathbf{a} - \mathbf{c}_1 \\ -\mathbf{c}_2 \end{bmatrix} \right\|_2^2 = \|R_1\mathbf{a} - \mathbf{c}_1\|_2^2 + \|\mathbf{c}_2\|_2^2. \end{aligned}$$

Poichà $\mathbf{c}_2$ è un vettore che non dipende da $\mathbf{a}$ il problema di minimo si riduce a minimizzare $\|R_1\mathbf{a} - \mathbf{c}_1\|_2$ e poichè R_1 è una matrice non singolare se $\mathbf{a}$ è il vettore che risolve il sistema

$$R_1\mathbf{a} = \mathbf{c}_1$$

allora tale minimo viene raggiunto. Si noti che questo algoritmo fornisce anche la misura del minimo che è pari a $\|\mathbf{c}_2\|_2$.
Per risolvere il problema dell'approssimazione polinomiale ai minimi quadrati si devono effettuare i seguenti passi:
1. Calcolare la fattorizzazione $B = QR$;
2. Calcolare il vettore $\mathbf{c} = Q^T\mathbf{y}$;
3. Risolvere il sistema triangolare superiore $R_1\mathbf{a} = \mathbf{c}_1$.

In questa parte del paragrafo riportiamo la codifica MatLab dell'algoritmo che serve per risolvere il problema dell'approssimazione polinomiale ai minimi quadrati. La seguente routine riceve in input i due vettori contenenti

rispettivamente le ascisse e le ordinate dei dati del problema ed il grado m del polinomio approssimante che si vuole ottenere. In output vengono calcolati il vettore, di dimensione $m + 1$, dei coefficienti di tale polinomio in ordine descrescente (cioè `a(1)` è il coefficiente della potenza più elevata, cioè x^m, ed il valore del cosiddetto residuo, cioè il valore del minimo calcolato (si veda (2.11)).

```
function [a,r] = LeastSquares(x,y,m)
%
% Funzione per il calcolo dell'approssimazione
% polinomiale ai minimi quadrati
% [a,r] = LeastSquares(x,y,m)
%
% x = vettore colonna delle n+1 ascisse
% y = vettore colonna delle n+1 misure
% m = grado dell'approssimazione
%
% a = vettore dei coefficienti del polinomio soluzione
% r = residuo calcolato
%
n = length(x);
a = zeros(m+1,1);
B = ones(n,m+1);
%
% Assegnazione della matrice B
%
for j=2:m+1
    B(:,j) = B(:,j-1).*x;
end
%
% Calcolo della fattorizzazione QR di B
%
[Q,R] = myqr(B);
%
% Calcolo del vettore c
%
c = Q'*y;
%
```

```
% Calcolo del vettore c2
%
c2 = c(m+2:n);
%
% Calcolo del vettore a
%
a(m+1) = c(m+1)/R(m+1,m+1);
for i=m:-1:1
    a(i) = (c(i)-R(i,i+1:m+1)*a(i+1:m+1))/R(i,i);
end
a = flipud(a);
%
% Calcolo del residuo
%
r=norm(c2,2);
return
```

Esempio 2.5.1 *Supponiamo di voler calcolare i polinomi che approssima ai minimi quadrati i punti* $(-1,-1)$, $(0,1)$, $(1,-1)$, $(3,2)$ *e* $(5,6)$. *Nella Figure 2.1, 2.2 e 2.3 sono tracciati rispettivamente i polinomi di approssimazione ai minimi quadrati di grado rispettivamente 1, 2 e 3 mentre i piccoli cerchi* `'o'` *indicano i punti che intendiamo approssimare. È facile osservare come tali polinomi non passano, di solito, per i punti del problema.*

2.5.1 Il Metodo QR

Completiamo la trattazione del metodo QR descrivendo un'ulteriore applicazione di tale fattorizzazione. Il metodo QR infatti è forse il metodo più popolare (e uno dei più efficienti) per il calcolo di tutti gli autovalori di una matrice. Una trattazione completa e approfondita che ne permetta un'efficiente implementazione risulta abbastanza complessa pertanto il metodo sarà descritto brevemente evidenziandone solo gli aspetti salienti.
Gli autovalori di A si ottengono come autovalori della matrice A_∞ ottenuta come limite del processo iterativo:

$$\begin{array}{lll} A_1 & = A & \\ A_k & = Q_k R_k & i = 1, 2, \ldots \\ A_{k+1} & = R_k Q_k = Q_{k+1} R_{k+1} & \end{array}$$

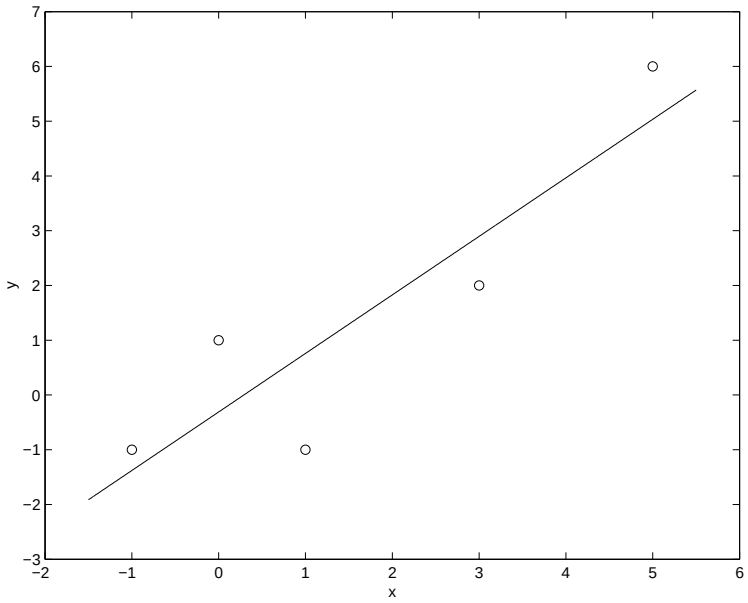


Figura 2.1: Approssimazione lineare ai minimi quadrati

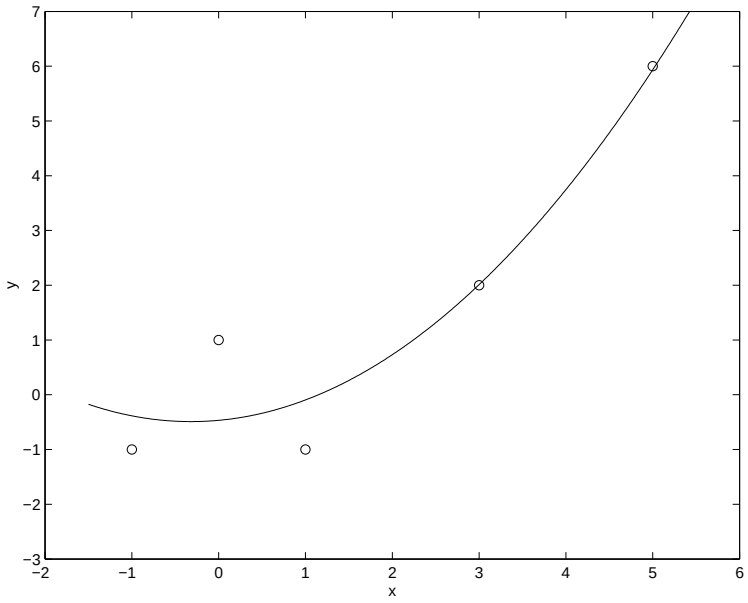


Figura 2.2: Approssimazione polinomiale di secondo grado ai minimi quadrati

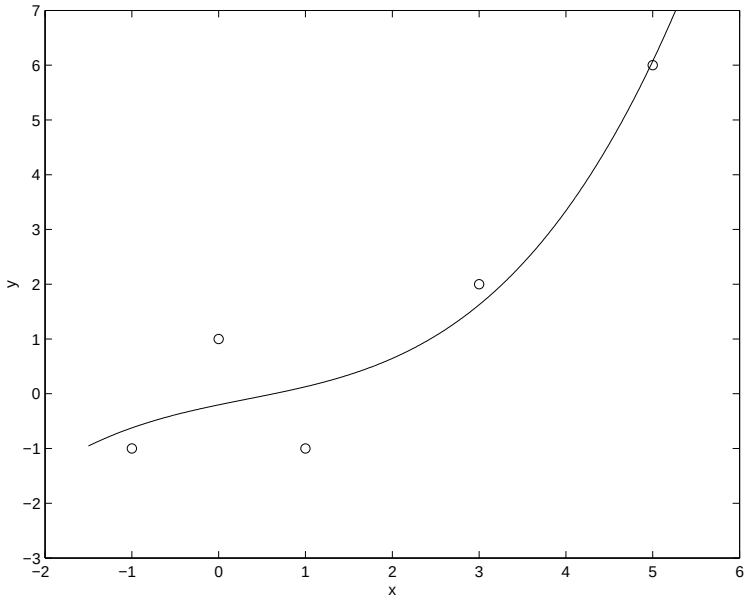


Figura 2.3: Approssimazione polinomiale di terzo grado ai minimi quadrati

dove $Q_k R_k$ è la fattorizzazione QR della matrice A_k. Si prova facilmente che le matrici $\{A_k\}$ sono tutte simili tra loro e dunque simili ad A. Infatti

$$A_{k+1} = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_k Q_k.$$

Poichè matrici simili hanno gli stessi autovalori è chiaro che gli autovalori di A_∞ sono gli stessi della matrice A.
In particolare si possono provare le seguenti proprietà:

1. Se A è simmetrica allora A_∞ è diagonale e dunque ha sulla sua diagonale gli autovalori di A;

2. Se A non è simmetrica ma ha tutti gli autovalori reali, allora A_∞ è una matrice triangolare superiore, che esibisce sulla diagonale principale gli autovalori di A;

3. Se A è generica ma reale allora A_∞ è una matrice triangolare superiore a blocchi, con blocchi diagonali di dimensione al più 2: gli autovalori di ogni blocco 2×2 corrispondono a coppie di autovalori di A complessi e coniugati.

Osservazione. Se la matrice A è densa (ovvero ha molti elementi diversi da zero) allorail metodo QR risulta troppo oneroso dal punto di vista computazionale se viene applicato direttamente ad A. Potendosi dimostrare che se la matrice A è in forma di Hessenberg superiore[1] o tridiagonale allora tutte le matrici A_i continuano a conservare questa struttura, risulta molto economico, prima di applicare il metodo QR, trasformare la matrice A in forma tridiagonale, se A è simmetrica, o di Hessenberg superiore, se A è generica.

[1]Ricordiamo che una matrice si dice in forma di Hessenberg superiore se $a_{ij} = 0$ per $i > j + 1$.

Capitolo 3

La Decomposizione ai Valori Singolari

3.1 Valori singolari

Definizione 3.1.1 *Se $A \in \mathbb{R}^{m\times n}$ allora esistono due matrici ortogonali U, V, con $U \in \mathbb{R}^{m\times m}$ e $V \in \mathbb{R}^{n\times n}$, ed una matrice diagonale $\Sigma \in \mathbb{R}^{m\times n}$, con elementi diagonali non negativi, tali che*

$$A = U\Sigma V^T. \tag{3.1}$$

Tale decomposizione viene appunto detta Decomposizione ai Valori Singolari (anche nota con l'acronimo inglese SVD=Singular Value Decomposition). Gli elementi diagonali della matrice Σ sono detti valori singolari. Se $m = n$ allora le tre matrici sono quadrate.
Invece se $m > n$ allora

$$\Sigma = \left[\begin{array}{ccccc} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & \sigma_n \\ \hline & & \mathrm{O} & & \end{array}\right],$$

mentre se $m < n$ allora risulta

$$\Sigma = \left[\begin{array}{ccccc|c} \sigma_1 & & & & & \\ & \sigma_2 & & & & \\ & & \ddots & & & \mathrm{O} \\ & & & \sigma_{m-1} & & \\ & & & & \sigma_m & \end{array}\right].$$

Si può osservare che il numero di valori singolari di una matrice $A \in \mathbb{R}^{m\times n}$ è pari a $p = \min\{m, n\}$. In ogni caso, per convenzione i valori singolari sono ordinati in modo tale che

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0.$$

Dalla relazione (3.1), moltiplicando i due membri per V, segue che

$$AV = U\Sigma, \tag{3.2}$$

mentre moltiplicandola per U^T risulta

$$U^T A = \Sigma V^T \quad \Rightarrow \quad A^T U = V\Sigma^T. \tag{3.3}$$

Riscriviamo le relazioni (3.2) e (3.3) considerando l'uguaglianza tra le colonne delle matrici, pertanto indichiamo con $\mathbf{u}_i$ e $\mathbf{v}_i$ le colonne di U e V:

$$U = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3\ \ldots\ \mathbf{u}_{m-1}\ \mathbf{u}_m]$$

$$V = [\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3\ \ldots\ \mathbf{v}_{n-1}\ \mathbf{v}_n].$$

I Caso: $m > n$ La relazione (3.2) risulta

$$\begin{aligned} AV &= [A\mathbf{v}_1\ A\mathbf{v}_2\ \ \ldots\ A\mathbf{v}_n] \\ U\Sigma &= [\sigma_1\mathbf{u}_1\ \sigma_2\mathbf{u}_2\ \ \ldots\ \sigma_n\mathbf{u}_n] \end{aligned} \tag{3.4}$$

pertanto

$$A\mathbf{v}_i = \sigma_i\mathbf{u}_i, \qquad i = 1, \ldots, n.$$

Per la relazione (3.3) invece si ottiene

$$\begin{aligned} A^T U &= \left[A^T\mathbf{u}_1\ A^T\mathbf{u}_2\ \ \ldots\ A^T\mathbf{u}_n\ A^T\mathbf{u}_{n+1}\ \ldots\ A^T\mathbf{u}_m\right] \\ V\Sigma^T &= [\sigma_1\mathbf{v}_1\ \sigma_2\mathbf{v}_2\ \ \ldots\ \sigma_n\mathbf{v}_n\ \mathbf{o}\ \ldots\ \mathbf{o}] \end{aligned} \tag{3.5}$$

da cui, uguagliando le colonne tra le matrici a primo e secondo membro

$$A^T\mathbf{u}_i = \sigma_i\mathbf{v}_i, \qquad i = 1,\dots,n$$

$$A^T\mathbf{u}_i = 0, \qquad i = n+1,\dots,m.$$

II Caso: $m < n$ La relazione (3.2) risulta

$$\begin{aligned} AV &= [A\mathbf{v}_1\ A\mathbf{v}_2\ \ \dots\ A\mathbf{v}_m\ A\mathbf{v}_{m+1}\ \dots\ A\mathbf{v}_n] \\ U\Sigma &= [\sigma_1\mathbf{u}_1\ \sigma_2\mathbf{u}_2\ \ \dots\ \sigma_m\mathbf{u}_m\ \mathbf{o}\ \dots\ \mathbf{o}] \end{aligned} \tag{3.6}$$

pertanto

$$A\mathbf{v}_i = \sigma_i\mathbf{u}_i, \qquad i = 1,\dots,m$$

$$A\mathbf{v}_i = 0, \qquad i = m+1,\dots,n.$$

Per la relazione (3.3) invece si ottiene

$$\begin{aligned} A^TU &= \left[A^T\mathbf{u}_1\ A^T\mathbf{u}_2\ \ \dots\ A^T\mathbf{u}_m\right] \\ V\Sigma^T &= [\sigma_1\mathbf{v}_1\ \sigma_2\mathbf{v}_2\ \ \dots\ \sigma_m\mathbf{v}_m] \end{aligned} \tag{3.7}$$

da cui, uguagliando le colonne tra le matrici a primo e secondo membro

$$A^T\mathbf{u}_i = \sigma_i\mathbf{v}_i, \qquad i = 1,\dots,m.$$

I vettori $\mathbf{u}_i$ sono detti vettori singolari sinistri, mentre i vettori $\mathbf{v}_i$ sono detti vettori singolari destri.
I valori singolari al quadrato sono gli autovalori non nulli della matrice A^TA (e anche di AA^T). Infatti supponendo $m > n$

$$A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T = V\Sigma_1V^T$$

dove

$$\Sigma_1 = \Sigma^T\Sigma = \begin{bmatrix} \sigma_1^2 & & & & \\ & \sigma_2^2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1}^2 & \\ & & & & \sigma_n^2 \end{bmatrix}.$$

Riscrivendo la relazione

$$A^TA = V\Sigma_1 V^T \qquad \Rightarrow \qquad (A^TA)V = V\Sigma_1$$

risulta

$$(A^TA)\mathbf{v}_i = \sigma_i^2\mathbf{v}_i, \qquad i = 1, \ldots, n,$$

quindi σ_i^2 è l'$i-$esimo autovalore di A^TA e $\mathbf{v}_i$ è il relativo autovettore.
Calcolando invece la matrice AA^T, si verifica che essa ha n autovalori che coincidono con il quadrato dei valori singolari di A, mentre i restanti $m-n$ sono nulli (infatti la matrice ha ordine m e quindi ammette necessariamente m autovalori). Anche in questo caso tuttavia i vettori singolari destri sono autovettori della matrice.
Ricordiamo ora che si definisce Rango di una matrice (rank) il numero massimo di righe (o colonne) di una matrice linearmente indipendenti. Se $r = \text{rank}(A)$ allora

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0, \qquad p = \min\{m, n\}.$$

La rappresentazione della SVD può avvenire in forma più compatta eliminando dalle matrici U, Σ e V le righe (o le colonne) che sono moltiplicate per elementi uguali a zero. In forma compatta una matrice $A \in \mathbb{R}^{m\times n}$ di rango $r < n < m$ viene rappresentata dalla decomposizione

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

in cui

$$U \in \mathbb{R}^{m\times r}, \qquad \Sigma \in \mathbb{R}^{r\times r}, \qquad V \in \mathbb{R}^{n\times r}.$$

Osserviamo che la matrice Σ può essere scritta in modo diverso. Sia E_i la matrice di dimensione $r \times r$ i cui elementi sono tutti uguali a zero tranne quello di posto (i, i) che e uguale a 1. Quindi

$$\Sigma = \sum_{i=1}^{r} \sigma_i E_i.$$

Per esempio se $r = 3$ allora

$$\Sigma = \sigma_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \sigma_2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \sigma_3 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Indicato con $\mathbf{e}_i$ l'$i-$esimo vettore della base canonica di $\mathbb{R}^r$ allora la matrice E_i può essere scritta come

$$E_i = \mathbf{e}_i \mathbf{e}_i^T.$$

Per esempio

$$E_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \mathbf{e}_2 \mathbf{e}_2^T.$$

Scrivendo Σ come

$$\Sigma = \sum_{i=1}^{r} \sigma_i \mathbf{e}_i \mathbf{e}_i^T$$

e sostituendo tale rappresentazione all'interno dell'espressione della SVD risulta:

$$A = U\Sigma V^T = U \sum_{i=1}^{r} \sigma_i \mathbf{e}_i \mathbf{e}_i^T V^T = \sum_{i=1}^{r} \sigma_i U \mathbf{e}_i \mathbf{e}_i^T V^T = \sum_{i=1}^{r} \sigma_i (U\mathbf{e}_i)(V\mathbf{e}_i)^T.$$

Il vettore $U\mathbf{e}_i$ rappresenta la $i-$esima colonna della matrice U, cioè l'$i-$esimo vettore singolare sinistro $\mathbf{u}_i$, mentre $V\mathbf{e}_i$ rappresenta la $i-$esima colonna della matrice V, cioè l'$i-$esimo vettore singolare destro $\mathbf{v}_i$, quindi

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

in cui ogni matrice $\mathbf{u}_i \mathbf{v}_i^T$ ha rango 1.
Osserviamo che per memorizzare tutti gli elementi della matrice A (indipendentemente dal rango) sono necessarie $m \times n$ locazioni di memoria, mentre per memorizzare la decomposizione ai valori singolari della matrice A di rango r ne servono $m \times r$ locazioni per memorizzare la matrice U, $n \times r$ locazioni per memorizzare la matrice V ed r locazioni per memorizzare i valori singolari. In tutto le locazioni necessarie sono:

$$mr + nr + r = (m + n + 1)r,$$

quindi memorizzare la matrice usando la sua SVD conviene se:

$$r(m + n + 1) < mn \qquad \Rightarrow \qquad r < \frac{mn}{m + n + 1}.$$

Per esempio se $m = 640$ e $n = 480$ allora

$$r \leq 274.$$

Sfruttando la positività dei valori singolari si può porre

$$\mathbf{w}_i = \sqrt{\sigma_i}\mathbf{u}_i, \qquad \mathbf{z}_i = \sqrt{\sigma_i}\mathbf{v}_i$$

cosicchè la matrice A viene scritta nel seguente modo

$$A = \sum_{i=1}^{r} \mathbf{w}_i \mathbf{z}_i^T.$$

3.1.1 Risoluzione del problema dei minimi quadrati con i valori singolari

Sia $A \in \mathbb{R}^{m\times n}$, di rango r, con $m > n \geq r$, allora si vuole risolvere il problema di minimo

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

essendo nota la SVD della matrice A:

$$A = U\Sigma V^T.$$

Sostituendo la decomposizione ai valori singolari al posto della matrice A

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\|_2^2 &= \|U\Sigma V^T\mathbf{x} - \mathbf{b}\|_2^2 = \|U(\Sigma V^T\mathbf{x} - U^T\mathbf{b})\|_2^2 \\ &= \|\Sigma V^T\mathbf{x} - U^T\mathbf{b}\|_2^2. \end{aligned}$$

Posto, per semplicità, $\mathbf{y} = V^T\mathbf{x}$ risulta

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 = \|\Sigma\mathbf{y} - U^T\mathbf{b}\|_2^2.$$

Per trovare un'espressione per il vettore $U^T\mathbf{b}$ possiamo sfruttare la decomposizione di U in colonne:

$$U^T\mathbf{b} = \begin{bmatrix} \mathbf{u}_1^T\mathbf{b} \\ \mathbf{u}_2^T\mathbf{b} \\ \vdots \\ \mathbf{u}_m^T\mathbf{b} \end{bmatrix}$$

ottenendo

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 = \sum_{i=1}^{r}(\sigma_i y_i - \mathbf{u}_i^T\mathbf{b})^2 + \sum_{i=r+1}^{m}(\mathbf{u}_i^T\mathbf{b})^2.$$

Il minimo di tale funzione viene raggiunto quando

$$\sigma_i y_i - \mathbf{u}_i^T\mathbf{b} = 0$$

da cui segue che

$$y_i = \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}, \qquad i = 1, \ldots, r.$$

Il vettore $\mathbf{y}^* \in \mathbb{R}^n$ che risolve il problema di minimo ha come componenti

$$y_i^* = \begin{cases} \dfrac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}, & i = 1, \ldots, r \\ \\ 0 & i = r+1, \ldots, n. \end{cases}$$

Poichè $\mathbf{x} = V\mathbf{y}$ allora il vettore $\mathbf{x}^* \in \mathbb{R}^n$ che risolve il problema di minimo definito inizialmente è

$$\mathbf{x}^* = V\mathbf{y}^* = \sum_{i=1}^{r} y_i^*\mathbf{v}_i = \sum_{i=1}^{r}\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i$$

mentre il valore minimo cercato è

$$\min_{\mathbf{x}\in\mathbb{R}^n}\|A\mathbf{x} - \mathbf{b}\|_2^2 = \sum_{i=r+1}^{m}\left(\mathbf{u}_i^T\mathbf{b}\right)^2.$$

3.1.2 Calcolo della SVD (Cenno)

La decomposizione ai valori singolari è calcolata tipicamente usando una procedura a due fasi. Nella prima la matrice viene ridotta in forma bidiagonale (ovvero gli unici elementi della matrice diversi da zero sono concentrati sulla diagonale principale e sulla prima sottodiagonale). Nella seconda fase viene calcolata la SVD della matrice bidiagonale B.
Il primo passo viene effettuato applicando le matrici di Householder in modo alternato alle colonne e alle righe (dimensionando opportunamente le matrici in base al numero di elementi da azzerare). Se le dimensioni della matrice

sono grandi allora si può rendere la matrice preliminarmente triangolare superiore per poi applicare i riflettori di Householder. La seconda fase viene effettuata applicando una variante del metodo QR per il calcolo degli autovalori della matrice B^TB. Un metodo alternativo prevede l'applicazione delle matrici di Givens (metodo di Jacobi), calcolando una successione di decomposizioni ai valori singolari di dimensione 2.

3.1.3 La Decomposizione ai Valori Singolari Troncata

Ricordiamo che se $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $r = \text{rank}(A)$, allora

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T.$$

La matrice A (di rango r) viene scritta come una somma di r matrici di rango 1. Ricordiamo che se $A \in \mathbb{R}^{m\times n}$ si definisce la norma 2 della matrice come il numero

$$\|A\|_2 = \sqrt{\rho(A^TA)}$$

dove $\rho(\cdot)$ indica il cosiddetto raggio spettrale della matrice, cioè il massimo modulo di un autovalore. Poichè gli autovalori della matrice A^TA coincidono con il quadrato dei valori singolari di A segue che

$$\|A\|_2 = \sqrt{\rho(A^TA)} = \sqrt{\sigma_1^2} = \sigma_1.$$

Fissato un valore k, intero compreso tra 1 ed r, $1 \leq k < r$, vogliamo trovare la matrice di rango k più vicina ad A, ovvero si vuole risolvere il seguente problema di minimo:

$$\min_{\text{rank}(X)=k} \|A - X\|_2.$$

La risoluzione a questo problema è legata ancora una volta alla SVD della matrice A ed è la cosiddetta SVD Troncata:

$$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

ovvero

$$\|A - A_k\|_2 = \min_{\text{rank}(X)=k} \|A - X\|_2.$$

Infatti poichè la matrice A_k ha k valori singolari diversi da zero, ha ovviamente rango k. Inoltre, calcolando la differenza tra le matrici A e A_k, risulta

$$A - A_k = \sum_{i=k+1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

e quindi i valori singolari della matrice $A - A_k$ sono

$$\sigma_{k+1} \geq \sigma_{k+2} \geq \cdots \geq \sigma_r.$$

La norma 2 di tale matrice è

$$\|A - A_k\|_2 = \sigma_{k+1}.$$

Se il più grande dei valori singolari esclusi dalla sommatoria è molto piccolo allora la matrice A_k è una buona approssimazione di A.

3.1.4 La Pseudoinversa di Moore-Penrose

Se la matrice A fosse quadrata e non singolare allora risolvere il sistema $A\mathbf{x} = \mathbf{b}$ trovando il vettore che minimizza la norma 2 $\|A\mathbf{x} - \mathbf{b}\|_2^2$ significa trovare, il vettore uguale al prodotto tra l'inversa di A (che esiste) ed il vettore dei termini noti $\mathbf{b}$:

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

La matrice inversa non esiste per matrici rettangolari, tuttavia il concetto può essere esteso introducendo la nozione di matrice pseudoinversa, indicata con A^+, tale che il vettore $\mathbf{x}^*$ trovato al termine del paragrafo precedente possa essere scritto come

$$\mathbf{x}^* = A^+\mathbf{b}.$$

Definizione 3.1.2 *Sia $A \in \mathbb{R}^{m\times n}$ una matrice di rango r. La matrice $A^+ \in \mathbb{R}^{n\times m}$ tale che*

$$A^+ = V\Sigma^+U^T$$

dove $\Sigma^+ \in \mathbb{R}^{n\times m}$ è la matrice che ha elementi σ_{ij}^+ nulli per $i \neq j$, mentre se $i = j$:

$$\sigma_{ii}^+ = \begin{cases} \dfrac{1}{\sigma_i}, & i = 1, \ldots, r \\ \\ 0 & i = r+1, \ldots, p = \min\{m, n\}. \end{cases}$$

è detta pseudoinversa di Moore-Penrose.

Valgono la seguenti proprietà:

1. La matrice A^+ è l'unica matrice che soddisfa le seguenti equazioni, dette di Moore-Penrose:
$$\begin{array}{ll} a) & AA^+A = A \\ b) & A^+AA^+ = A^+ \\ c) & (AA^+)^T = AA^+ \\ d) & (A^+A)^T = A^+A; \end{array}$$

2. Se il rango di A è massimo allora:
se $m \geq n$, $A^+ = (A^TA)^{-1}A^T$,
se $m \leq n$, $A^+ = A^T(AA^T)^{-1}$,
se $m = n$, $A^+ = A^{-1}$;

3. $$A^+ = \sum_{i=1}^{r} \frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^T;$$

4. Considerato il sistema rettangolare
$$A\mathbf{x} = \mathbf{b}$$
allora il vettore soluzione nel senso dei minimi quadrati è
$$\mathbf{x}^* = A^+\mathbf{b}.$$
Infatti
$$A^+\mathbf{b} = \sum_{i=1}^{r} \frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^T\mathbf{b} = \sum_{i=1}^{r} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i = \mathbf{x}^*.$$

Ricordiamo che se $A \in \mathbb{R}^{m\times n}$ allora si definisce Range di A, e si indica con $R(A)$, l'insieme:
$$R(A) = \{\mathbf{y} \in \mathbb{R}^m \mid \mathbf{y} = A\mathbf{x},\ \mathbf{x} \in \mathbb{R}^n\}.$$
Quindi se $\mathbf{y} \in R(A)$ allora esiste un vettore $\mathbf{x} \in \mathbb{R}^n$ tale che:
$$\mathbf{y} = A\mathbf{x} = \sum_{i=1}^{p} \sigma_i\mathbf{u}_i\mathbf{v}_i^T\mathbf{x} = \sum_{i=1}^{p} (\sigma_i\mathbf{v}_i^T\mathbf{x})\mathbf{u}_i,$$
quindi le colonne della matrice U rappresentano una base (ovviamente ortonormale) dell'insieme $R(A)$. Procedendo in modo analogo si può definire il Range della matrice A^T, $R(A^T)$ e verificare che le colonne della matrice V rappresentano una base per tale spazio.

3.2 Un'applicazione della SVD: La Compressione di Immagini Digitali

Il processo di digitalizzazione è un processo di approssimazione (campionamento): fisicamente all'immagine reale viene associata una griglia composta da un elevato numero di piccoli quadrati, detti pixel. A ciascuno di questi pixel viene attribuito un colore (di solito uguale alla media dei colori contenuti nella cella oppure uguale al colore dominante). Nelle foto in bianco e nero il valore di un pixel è un valore che corrisponde ad una tonalità di grigio. Teoricamente il valore può andare da 0 a ∞ perchè infinite sono le tonalità di grigio possibili. Non essendo possibile tuttavia rappresentare infiniti valori allora si pone un limite massimo e si considera un sottoinsieme finito di tali tonalità. Una rappresentazione conveniente è quella di associare ad ognitonalità di grigio un valore reale appartenente all'intervallo $[0, 1]$, considerando che il valore 0 viene associato al bianco e 1 al nero. A questo punto di sceglie un sottoinsieme discreto di tali valori: una quantificazione consueta è quella che prevede 256 livelli di grigio, così ogni livello di grigio può essere rappresentato utilizzando giusto 8 bit ($256 = 2^8$). Le immagini a colori sono rappresentate in modo più complesso perchè l'informazione è multidimensionale. Per rappresentare un colore si usa un determinato spazio definito scegliendo un insieme di colori base. Lo spazio di rappresentazione usato più comunemente è quello RGB (Red-Green-Blue cioè Rosso Verde Blu). In tale spazio ogni pixel è rappresentato da un vettore tridimensionale in cui ogni componente rappresenta l'intensità di uno dei colori di base:

$$Colore = x_R R + x_G G + x_B B.$$

I valori x_R, x_G, x_B variano nell'intervallo di numeri interi $[0, 255]$ oppure assumono valori normalizzati appartenenti all'intervallo reale $[0, 1]$. Simile allo spazio RGB è lo spazio CMYK che si basa sulla scomposizione dei colori nei 4 colori primari Ciano, Magenta, Giallo e Nero. In questo caso il colore viene scomposto in una quadrupla di valori:

$$Colore = x_C C + x_M M + x_Y Y + x_N N.$$

Ci sono altre rappresentazioni molto più complesse come quella YIQ, utilizzata dallo standard televisivo stantunitense NTSC (National Television Standard Committee). In questo caso il valore Y rappresenta una tonalità del grigio mentre I e Q sono i cosiddetti segnali di crominanza e rappresentano

le coordinate del colore in una particolare rappresentazione. La conversione da RGB a YIQ avviene attraverso la relazione matriciale:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

Una volta stabilito lo spazio dei colori ogni pixel è memorizzato attraverso un vettore definito in tale spazio.
Se l'immagine ha $m \times n$ pixel e lo spazio scelto è quello RGB allora per rappresentarla è necessario memorizzare $3mn$ elementi. Nella seguente tabella sono state riportate le quantità di memoria necessarie per memorizzare immagini digitali di dimensioni variabili nei due spazi usati più comunemente:

Dimensione	Spazio RGB	Spazio CMYK	Memoria (RGB) in Mb
640×480	921600 byte	1228800 byte	0.9 Mb
800×600	1440000 byte	1920000 byte	1.1 Mb
1024×768	2359296 byte	3145728 byte	2.25 Mb

La dimensioni dei file sono piuttosto elevate quindi se un'immagine digitale deve essere trasmessa usando un opportuno canale il costo di trasmissione sia in termini di tempo che di prestazioni richieste è piuttosto elevato. Questa è la motivazione che ha spinto a ricercare meccanismi possibimente affidabili e poco costosi per comprimere l'informazione contenuta in un'immagine digitale. La tecnica di compressione delle immagini si basa sul fatto che queste non sono costituite da un insieme casuale di punti colorati ma esiste una determinata struttura. Questa struttura può essere sfruttata per rappresentare l'immagine utilizzando un numero minore di elementi. Esistono due tipi ci compressione:
1. Compressione lossless (Senza perdita di informazione);
2. Compressione lossy (Con perdita di informazione).
Per misurare la qualità di un algoritmo di compressione si utilizza il cosiddetto indice di compressione

$$C = \frac{n_I}{n_{I_c}}$$

dove n_I è il numero di unità di memoria necessarie a memorizzare l'immagine originale, mentre n_{I_c} è il numero di unità di memoria necessarie a memorizzare l'immagine compressa.

Un valore $C = 10$ significa che per memorizzare l'immagine compressa serve il 10% di quella necessaria a memorizzare l'immagine originale. Nella compressione lossless l'immagine ricostruita è identica, pixel per pixel, all'immagine originale. Le routine di compressione vengono applicate all'insieme dei dati cercando di memorizzare le stesse informazioni ma con un numero inferiore di informazioni (esempio: zone di colore uniforme). Le due tecniche più utilizzate per la compressione di immagini digitali sono:
1. GIF (Graphic Interchange Format): L'immagine viene convertita in una lunga stringa di dati (concatendo delle righe) a cui è applicato l'algoritmo di compressione LZW;
2. JPEG (Joint Experts Photographic Group): Alla stringa di dati viene applicata prima la Trasformata Discreta Coseno e poi la cosiddetta codifica di Huffman.
Il vantaggio principale di tale tecnica è la ricostruzione fedele dell'immagine di partenza mentre lo svantaggio sta nel fatto che l'indice di compressione potrebbe essere un numero non molto elevato (prossimo a 1) e quindi il processo di compressione risulta poco efficace. Nella compressione lossy l'immagine ricostruita non è perfettamente uguale a quella di partenza. La motivazione di questa scelta dipende dal fatto che il processo attraverso il quale l'occhio umano percepisce un'immagine reale è esattamente lo stesso su cui si basa la digitalizzazione di un'immagine e pertanto l'occhio umano non è in grado di percepire perfettamente la variazione del singolo pixel. Questo tipo di compressione funziona bene solitamente con immagini prese dalla vita reale (foto non artistiche). L'immagine ricostruita è simile a quella originale nel senso che l'occhio umano percepisce l'immagine nel suo insieme e non nei dettagli. L'uso della SVD nella compressione di immagini digitali avviene nel seguente modo: se A è la matrice $m \times n$ dei pixel dell'immagine digitale si può ragionevolmente pensare che ci siano diverse righe (o colonne) che variano poco (cioè tali da poter essere considerate come linearmente dipendenti), quindi il rango di A può essere un valore molto più piccolo del $\min\{m, n\}$. All'immagine definita dalla A viene sostituita l'immagine definita da A_k.
Se indichiamo con $\widehat{A}$ la matrice dell'immagine approssimata allora la qualità dell'immagine percepita dall'occhio umano è ovviamente una misura molto

soggettiva. Definiamo la seguente quantità:

$$RSME = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - \widehat{a}_{ij})^2}{mn}}$$

Il valore RSME (Root Square Mean Error-ma spesso riportato anche come RMSE) è una misura (media) oggettiva di quanto l'immagine approssimata differisce da quella originale. Il seguente valore

$$PSNR = 10\log_{10}\left(\frac{1}{RSME}\right)$$

misura invece l'errore percepito dall'occhio. Poichè il calcolo della SVD di una matrice di grandi dimensioni ha un alto costo computazionale (pari a circa m^2n) allora può essere utile suddividere l'immagine in un certo numero di parti (anche di dimensioni diseguali) e manipolarle in modo indipendente. Questa tecnica consente di utilizzare un valore del rango non costante per le parti dell'immagine. Per esempio una parte di immagine che rappresenta il cielo (o il mare) può essere rappresentata con una matrice di rango 1 mentre per la parte in cui sono presenti elementi dai colori e forme diversi si può utilizzare un rango superiore.
Nelle pagine seguenti sono presentati alcuni esempi di applicazione della SVD alla compressione di un'immagine in bianco e nero 512×512 di rango massimo, utilizzando approssimazioni troncate di rango crescente.

Figura 3.1: Immagine originale

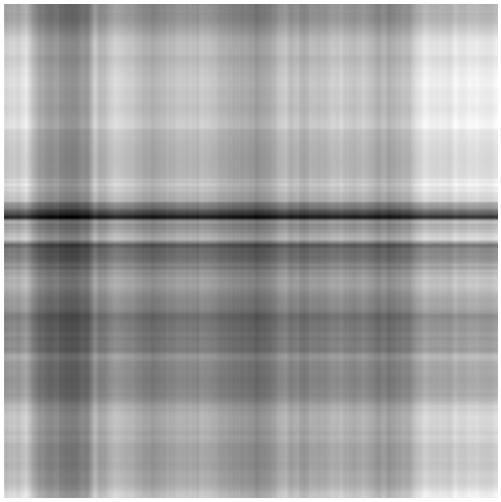


Figura 3.2: Approssimazione di rango 1

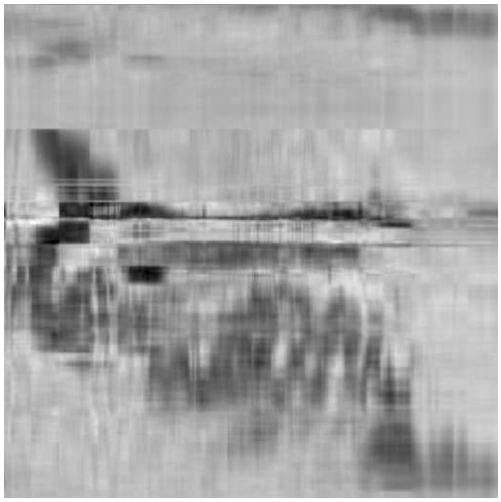


Figura 3.3: Approssimazione di rango 10

Figura 3.4: Approssimazione di rango 25

Figura 3.5: Approssimazione di rango 50

Figura 3.6: Approssimazione di rango 100

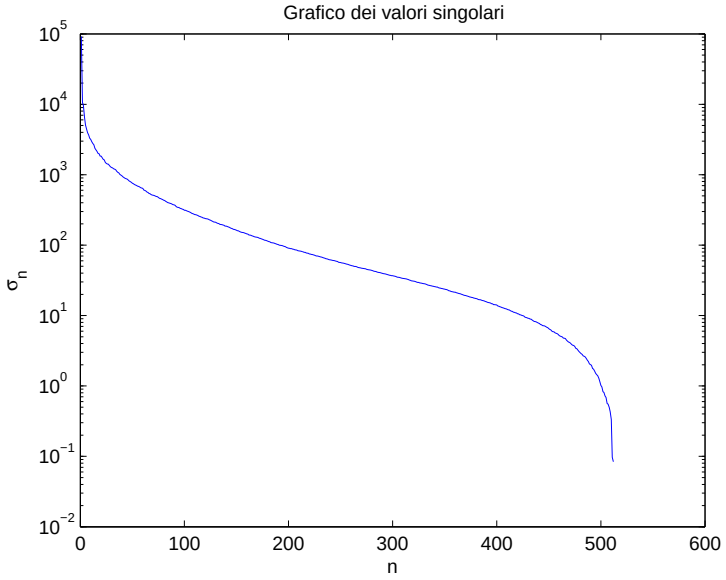


Figura 3.7: Valori singolari dell'immagine digitale

Capitolo 4

Sistemi di Equazioni non Lineari

4.1 Introduzione

Supponiamo che sia Ω un sottoinsieme di $\mathbb{R}^n$ e che siano assegnate le n funzioni

$$f_i : \Omega \to \mathbb{R}, \qquad i = 1, \ldots, n.$$

Ogni vettore $\mathbf{x} \in \mathbb{R}^n$, soluzione del sistema non lineare di n equazioni in n incognite

$$\begin{aligned} &f_1(x_1, x_2, \ldots, x_n) = 0 \\ &f_2(x_1, x_2, \ldots, x_n) = 0 \\ &\vdots \\ &f_n(x_1, x_2, \ldots, x_n) = 0 \end{aligned} \tag{4.1}$$

prende il nome di radice dell'equazione

$$F(\mathbf{x}) = 0 \tag{4.2}$$

oppure di zero della funzione vettoriale

$$F : \Omega \to \mathbb{R}^n$$

dove il vettore $F(\mathbf{x})$ è definito da:

$$F(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix}.$$

Ovviamente se $n = 1$ la teoria è quella degli zeri di funzioni scalari. Anche il caso in cui $F(\mathbf{x})$ sia una funzione lineare (cioè $F(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, con A matrice quadrata non singolare di ordine n e $\mathbf{b}$ vettore colonna ad n componenti) è già stato considerato. Tutti i metodi per la risoluzione del sistema non lineare $F(\mathbf{x}) = 0$ partono dalle seguenti due ipotesi:

1. la funzione $F(\mathbf{x})$ è calcolabile in ogni punto del dominio Ω;
2. la funzione $F(\mathbf{x})$ è continua in un opportuno intorno della radice.

Come nel caso scalare l'equazione $F(\mathbf{x}) = 0$ viene trasformata in un problema del tipo

$$\mathbf{x} = \Phi(\mathbf{x}) \tag{4.3}$$

ovvero

$$x_i = \Phi_i(x_1, x_2, \ldots, x_n), \qquad i = 1, 2, \ldots, n$$

con $\Phi(\mathbf{x})$ funzione definita in Ω e scelta in modo tale che le proprietà richiesta ad $F(\mathbf{x})$ si trasferiscano su Φ, cioè anch'essa deve essere continua in un opportuno intorno della radice e calcolabile nell'insieme di definizione. Il motivo di tali richieste è che la funzione $\Phi(\mathbf{x})$ viene utilizzata per definire una successione di vettori nel seguente modo. Sia $\mathbf{x}^{(0)}$ un vettore iniziale appartenente a Ω e definiamo la seguente successione

$$\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)}), \qquad k = 0, 1, 2, 3, \ldots \tag{4.4}$$

ovvero

$$x_i^{(k+1)} = \Phi_i(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}), \qquad i = 1, 2, \ldots, n.$$

La funzione $\Phi(\mathbf{x})$ prende il nome di funzione iteratrice dell'equazione non lineare $F(\mathbf{x}) = 0$. Ricordiamo che un vettore $\mathbf{x}^*$ che soddisfa la (4.3) viene detto punto fisso di $\Phi(\mathbf{x})$ (oppure punto unito). La successione dei vettori $\mathbf{x}^{(k)}$ definisce il metodo delle approssimazioni successive per il calcolo appunto di tale punto fisso. Quello che si richiede a tale successione è che essa converga al vettore $\mathbf{x}^*$, soluzione del sistema non lineare. In questo caso per convergenza si intende che

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

cioè, in termini di componenti,

$$\lim_{k\to\infty} \mathbf{x}_i^{(k)} = \mathbf{x}_i^*.$$

Per la convergenza del metodo delle approssimazioni successive vale quindi il seguente teorema.

Teorema 4.1.1 *Se la funzione* $\Phi(\mathbf{x})$ *è differenziabile con continuità in un intorno del punto fisso* $\mathbf{x}^*$*, e risulta*

$$\rho(J_\Phi(\mathbf{x}^*)) < 1$$

allora, scelto $\mathbf{x}^{(0)}$ *appartenente a tale intorno, la successione costruita con il metodo delle approssimazioni successive è convergente a* $\mathbf{x}^*$*.*

Chiaramente il risultato appena enunciato ha un'importanza teorica in quanto generalmente è molto complesso (o non è possibile) conoscere gli autovalori della matrice Jacobiana nella soluzione del sistema non lineare.

4.2 Il Metodo di Newton per Sistemi non Lineari

Se si conosce abbastanza bene l'approssimazione iniziale della soluzione del sistema di equazioni

$$F(\mathbf{x}) = 0$$

il metodo di Newton risulta molto efficace. Il Metodo di Newton per risolvere il sistema (4.2) può essere derivato in modo semplice come segue. Sia $\mathbf{x}^{(k)}$ una buona approssimazione a $\mathbf{x}^*$, soluzione di $F(\mathbf{x}) = 0$, possiamo allora scrivere lo sviluppo in serie della funzione F valutata nella soluzione del sistema non lineare prendendo come punto iniziale proprio il vettore $\mathbf{x}^{(k)}$:

$$0 = F(\mathbf{x}^*) = F(\mathbf{x}^{(k)}) + J_F(\xi_k)(\mathbf{x}^* - \mathbf{x}^{(k)})$$

dove

$$\xi_k = \theta\mathbf{x}^* + (1-\theta)\mathbf{x}^{(k)} \qquad 0 \leq \theta \leq 1$$

ovvero, suppondendo che lo Jacobiano sia invertibile,

$$\mathbf{x}^* - \mathbf{x}^{(k)} = -J_F^{-1}(\xi_k)F(\mathbf{x}^{(k)}) \Rightarrow \mathbf{x}^* = \mathbf{x}^{(k)} - J_F^{-1}(\xi_k)F(\mathbf{x}^{(k)}). \tag{4.5}$$

Se $\mathbf{x}^{(k)}$ è abbastanza vicino a $\mathbf{x}^*$ allora possiamo confondere $\mathbf{x}^{(k)}$ con ξ_k; in tal modo però (4.5) non fornirà esattamente $\mathbf{x}^*$ ma un valore ad esso vicino, diciamo $\mathbf{x}^{(k+1)}$. Abbiamo quindi lo schema iterativo:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_F^{-1}(\mathbf{x}^{(k)})F(\mathbf{x}^{(k)}). \tag{4.6}$$

Il discorso può essere ripetuto ora partendo da $\mathbf{x}^{(k+1)}$; dunque k può essere visto come un indice di iterazione. La (4.6) è appunto la formula che definisce il metodo di Newton.
Può essere interessante soffermarsi su alcuni dettagli di implementazione del metodo (4.6). Poniamo infatti

$$\mathbf{z}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$$

e osserviamo che, moltiplicando per la matrice $J_F(\mathbf{x}^{(k)})$ l'espressione del metodo di Newton diventa

$$J_F(\mathbf{x}^{(k)})\mathbf{z}^{(k)} = -F(\mathbf{x}^{(k)})$$

da cui, risolvendo il sistema lineare che ha $J_F(\mathbf{x}^{(k)})$ come matrice dei coefficienti e $-F(\mathbf{x}^{(k)})$ come vettore dei termini noti si può ricavare il vettore $z^{(k)}$ e ottenere il vettore al passo $k+1$:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}.$$

L'algoritmo, ad un generico passo k, può essere così riassunto:

1. Calcolare la matrice $J_F(\mathbf{x}^{(k)})$ e il vettore $-F(\mathbf{x}^{(k)})$;
2. Risolvere il sistema lineare $J_F(\mathbf{x}^{(k)})\mathbf{z}^{(k)} = -F(\mathbf{x}^{(k)})$;
3. Calcolare il vettore $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}$;
4. Valutare la convergenza: fissata una tolleranza ε, se risulta

   $$\|\mathbf{z}^{(k)}\| \leq \varepsilon$$

   allora $\mathbf{x}^{(k+1)}$ è una buona approssimazione della soluzione, altrimenti si ritorna al passo 1.

Occorre evidenziare subito come il punto 2. richieda un notevole costo computazionale. Infatti se il sistema in questione viene risolto utilizzando la fattorizzazione LU ad ogni passo si deve calcolare tale fattorizzazione e poi procedere alla risoluzione di due sistemi triangolari. Per diminuire tale costo si potrebbe procedere nel seguente modo. Anzichè calcolare la matrice al punto 1. ad ogni k si potrebbe calcolarla ogni m iterazioni, cosicchè si calcolerebbe la fattorizzazione LU alla $m-$esima iterazione e ad ogni passo per risolvere il sistema al punto 2. si risolverebbero solo i due sistemi triangolari. L'algoritmo potrebbe essere riassunto nel seguente modo:

1. Se $k = qm$ allora calcolare la matrice $A = J_F(\mathbf{x}^{(k)})$ e la sua fattorizzazione $A = LU$;
2. Calcolare il vettore $-F(\mathbf{x}^{(k)})$;
3. Risolvere il sistema triagolare inferiore $L\mathbf{w}^{(k)} = -F(\mathbf{x}^{(k)})$;
4. Risolvere il sistema triagolare superiore $U\mathbf{z}^{(k)} = \mathbf{w}^{(k)}$;
5. Calcolare il vettore $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}$;
6. Valutare la convergenza, come al punto 4. del metodo di Newton.

Questo metodo viene detto Metodo di Newton Modificato (o Congelato o Semplificato) e fa parte della classe dei metodi cosiddetti quasi-Newton in quanto derivati da tale metodo.
Un modo di evitare il calcolo dell'intera matrice $J_F(x)$ ad ogni passo consiste nel considerare l'$i-$esima equazione del sistema (4.1) come un'equazione nella sola incognita x_i ed applicando a ciascuna equazione del sistema il metodo di Newton-Raphson per equazioni. Supponendo che, in un opportuno dominio contenenete $\mathbf{x}^*$ sia

$$\frac{\partial f_i(x_1, \ldots, x_n)}{\partial x_i} \neq 0, \qquad i = 1, 2, \ldots, n$$

si ottiene, per $k = 0, 1, 2, \ldots$

$$x_i^{(k+1)} = x_i^{(k)} - \frac{f_i(x_1^{(k)}, \ldots, x_n^{(k)})}{\dfrac{\partial f_i(x_1^{(k)}, \ldots, x_n^{(k)})}{\partial x_i}}, \qquad i = 1, 2, \ldots, n, \tag{4.7}$$

che viene detto metodo di Jacobi-Newton.
Una variante del metodo (4.7) si ottiene introducendo la cosiddetta tecnica iterativa di Gauss-Seidel, della forma

$$x_i^{(k+1)} = x_i^{(k)} - \frac{f_i(x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)} x_i^{(k)}, \ldots, x_n^{(k)})}{\dfrac{\partial f_i(x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)} x_i^{(k)}, \ldots, x_n^{(k)})}{\partial x_i}}, \qquad i = 1, 2, \ldots, n, \tag{4.8}$$

dove si sono utilizzate, per il calcolo di $x_i^{(k+1)}$, le prime $i-1$ componenti del vettore al passo $k+1$ già note. Il metodo (4.8) è noto come metodo di Newton-Gauss-Seidel.

Capitolo 5

Il MATLAB

5.1 Introduzione al MATLAB

Il Matlab (acronimo delle parole inglesi *MATrix LABoratory*) è un software basato sulla manipolazione di matrici molto utilizzato nel campo della ricerca scientifica, non solo matematica, per la sua grande portabilità (infatti è disponibile sia per grandi workstation che per comuni personal computers), unita ad una notevole facilità d'uso e alle potenzialità di calcolo. Inoltre l'uso del Matlab è reso facile dalla presenza di un manuale dei comandi in linea, che può essere invocato tramite il comando `help`, e dalla presenza del comando `demo` che presenta numerosi e significativi esempi di applicazioni di tutte le funzioni Matlab. Nelle seguenti pagine faremo riferimento alle istruzioni presenti nelle ultime versioni del software.
Una volta lanciata l'esecuzione del programma compare il prompt del software

```
>>
```

il che significa che MatLab resta in attesa che venga effettuata una qualsiasi operazione (editare un programma, lanciare l'esecuzione di un file oppure eseguire un'istruzione digitata sulla riga di comando e così via).
Il comando `help`, come già detto, fornisce tutte le informazioni relative ad un particolare comando oppure una lista di tutti gli argomenti per i quali è presente un aiuto. La sintassi del comando è semplice:

```
>> help
```

oppure

```
>> help comando
```

Per esempio per sapere l'uso del comando `load`, che descriveremo in dettaglio nel seguito, è sufficiente scrivere

```
>> help load
```

Anche il comando `demo` ha una sintassi molto semplice:

```
>> demo
```

a questo punto compariranno sullo schermo alcuni menu e basterà scegliere, tramite il mouse, l'argomento del quale si vuole vedere una dimostrazione.
Il Matlab può essere considerato un interprete le cui istruzioni sono del tipo:

```
variabile = espressione
```

oppure

```
variabile
```

In quest'ultimo caso, quando cioè un'istruzione è costituita solo dal nome di una variabile viene interpretata come la visualizzazione del valore di tale variabile. Vediamo i seguenti esempi.

```
>> b=5;
>> b
ans =
   5
>>


>> b=5
b =
   5
>>
```

Nel primo caso il valore di output di `b` è stato attribuito alla variabile di comodo `ans` (abbreviazione per la parola inglese *answer*). Questo modo di procedere viene utilizzato anche quando si chiede di valutare un'espressione di tipo numerico senza l'ausilio di variabili.

```
>> 3+4
ans =
   7
>>
```

Ogni espressione introdotta viene interpretata e calcolata. Ogni istruzione può essere scritta anche su due righe purchè prima di andare a capo vengano scritti 3 punti ″...″. Più espressioni possono essere scritte sulla stessa riga purchè siano separate da una virgola o dal punto e virgola. Se una riga di un file Matlab inizia con % allora tale riga viene considerata come un commento. Il Matlab fa distinzione tra lettere minuscole e maiuscole, quindi se abbiamo definito una variabile `A` e facciamo riferimento a questa scrivendo `a` essa non viene riconosciuta.
Le frecce della tastiera consentono di richiamare e riutilizzare comandi scritti in precedenza; utilizzando infatti ripetutamente il tasto ↑ vengono visualizzate le linee di comando precedentemente scritte. Per tornare ad un'istruzione sorpassata basta premere il tasto ↓. Con i tasti ⟵ e ⟶ ci si sposta veso sinistra oppure verso destra sulla riga di comando su cui ci si trova.

5.2 Assegnazione di matrici

La prima cosa da imparare del Matlab è come manipolare le matrici che costituiscono la struttura fondamentale dei dati. Una matrice è una tabella di elementi caratterizzata da due dimensioni: il numero delle righe e quello delle colonne. I vettori sono matrici aventi una delle dimensioni uguali a 1. Infatti esistono due tipi di vettori: i *vettori riga* aventi dimensione $1 \times n$, e i *vettori colonna* aventi dimensione $n \times 1$. I dati scalari sono matrici di dimensione 1×1. Le matrici possono essere introdotte in diversi modi, per esempio possono essere assegnate esplicitamente, o caricate da file di dati esterni, o generate utilizzando funzioni predefinite.
Per esempio l'istruzione

```
>> A = [1 2 3; 4 5 6; 7 8 9];
```

assegna alla variabile `A` una matrice di tre righe e tre colonne. Gli elementi di una riga della matrice possono essere separate da virgole o dallo spazio, mentre le diverse righe sono separate da un punto e virgola. Se alla fine

dell'assegnazione viene messo il punto e virgola allora la matrice non viene visualizzata sullo schermo. In generale se vogliamo assegnare ad `A` una matrice ad m righe ed n colonne la sintassi è la seguente:

```
>> A = [riga 1; riga 2; ...; riga m];
```

Per assegnare ad una variabile `x` un vettore riga si ha

```
>> x = [3 -4 5];
```

gli elementi possono anche essere separati da una virgola

```
>> x = [3,-4,5];
```

Per assegnare invece ad una variabile un vettore colonna basta separare gli elementi con un punto e virgola:

```
>> y = [1;-3;6];
```

La stessa matrice `A` dell'esempio visto in precedenza può essere assegnata anche a blocchi:

```
>> A = [ 1 2 3; 4 5 6];
>> b = [ 7 8 9];
>> A = [ A; b];
```

mentre in modo analogo si può anche aggiungere una colonna:

```
>> A = [-1 2 3; 0 5 6; -5 4 3];
>> x = [-7; 0; 9];
>> A = [ A, x];
```

Descriviamo ora alcune funzioni predefinite che forniscono in output determinate matrici.

```
>> A=rand(m,n)
```

costruisce una matrice $m \times n$ di elementi casuali uniformemente distribuiti tra 0 e 1;

```
>> A=zeros(m,n)
```

costruisce una matrice $m \times n$ di elementi nulli;

```
>> A=ones(m,n)
```

costruisce una matrice $m \times n$ di elementi tutti uguali a 1;

```
>> A=eye(m,n)
```

costruisce una matrice $m \times n$ i cui elementi sono uguali a 1 sulla diagonale principale e 0 altrove. Per le funzioni appena viste se uno dei due parametri è omesso allora la matrice costruita viene considerata quadrata.
Il dimensionamento delle matrici è automatico. Per esempio se si pone

```
>> B = [1 2 3; 4 5 6];
```

e successivamente

```
>> B = [1 0; 0 7];
```

il programma riconosce che la matrice `B` ha cambiato le dimensioni da 2×3 a 2×2.
Per identificare l'elemento della matrice che si trova al posto `j` nella riga `i` si usa la notazione `A(i,j)`.
Per esempio `A(4,2)` indica l'elemento che si trova nella quarta riga e in colonna 2. La numerazione delle righe e delle colonne parte sempre da 1 (quindi il primo elemento della matrice è sempre `A(1,1)`).
Per fare riferimento ad un singolo elemento di un vettore (sia riga che colonna) è sufficiente utilizzare un solo indice (quindi la notazione `x(i)` indica l'`i`-esima componente del vettore `x`).
Se si fa riferimento a un elemento di una matrice di dimensione $m \times n$ che non esiste allora il Matlab segnala l'errore con il seguente messaggio:
`Index exceeds matrix dimension`.
Se `C` è una matrice non ancora inizializzata allora l'istruzione

```
>> C(3,2)= 1
```

fornisce come risposta

```
C =
    0    0
    0    0
    0    1
```

cioè il programma assume come dimensioni per `C` dei numeri sufficientemente grandi affinchè l'assegnazione abbia senso. Se ora si pone

```
>> C(1,3)= 2
```

si ha:

```
C =
   0   0   2
   0   0   0
   0   1   0
```

In Matlab gli indici devono essere strettamente positivi mentre se si richiede un elemento di indice negativo oppure uguale a zero si ha sullo schermo il seguente messaggio di errore:
`Index into matrix is negative or zero`

5.2.1 Sottomatrici e notazione :

Vediamo ora alcuni esempi che illustrano l'uso di : per vettori e matrici. Le istruzioni

```
>> x=[1:5];
```

e

```
>> x=1:5;
```

sono equivalenti all'assegnazione diretta del vettore `x`:

```
>> x=[1 2 3 4 5];
```

Ciò vale anche per vettori di elementi reali. Infatti l'istruzione

```
>> x=[0.2:0.2:1.2];
```

equivale a scrivere

```
>> x=[0.2 0.4 0.6 0.8 1.0 1.2];
```

Inoltre è possibile anche l'uso di incrementi negativi:

```
>> x=[5:-1:1];
```

è equivalente a

```
>> x=[5 4 3 2 1];
```

L'istruzione

```
>> x=x(n:-1:1);
```

inverte gli elementi del vettore `x` di dimensione n. La notazione : può essere anche applicata a matrici. Infatti se `A` è una matrice abbiamo:

```
>> y=A(1:4,3);
```

assegna al vettore colonna `y` i primi 4 elementi della terza colonna della matrice `A`;

```
>> y=A(4,2:5);
```

assegna al vettore riga `y` gli elementi della quarta riga di `A` compresi tra il secondo e il quinto;

```
>> y=A(:,3);
```

assegna al vettore colonna `y` la terza colonna di `A`;

```
>> y=A(2,:);
```

assegna al vettore riga `y` la seconda riga di `A`;

```
>> B=A(1:4,:);
```

assegna alla matrice `B` le prime 4 righe di `A`;

```
>> B=A(:,2:6);
```

assegna alla matrice `B` le colonne di `A` il cui indice è compreso tra 2 e 6;

```
>> B=A(:,[2 4]);
```

assegna alla matrice `B` la seconda e la quarta colonna di `A`;

```
>> A(:,[2 4 5])=B(:,1:3);
```

sostituisce alle colonne 2, 4 e 5 della matrice `A` le prime 3 colonne della matrice `B`.

5.3 Operazioni su matrici e vettori

In Matlab sono definite le seguenti operazioni su matrici e vettori:

+	addizione
−	sottrazione
*	moltiplicazione
^	elevazione a potenza
'	trasposto
/	divisione
( )	specificano l'ordine di valutazione delle espressioni

Ovviamente queste operazioni possono essere applicate anche a scalari. Se le dimensioni delle matrici coinvolte non sono compatibili allora viene segnalato un errore eccetto nel caso di operazione tra uno scalare e una matrice. Per esempio se A è una matrice di qualsiasi dimensione allora l'istruzione

```
>> C = A+2;
```

assegna alla matrice C gli elementi di A incrementati di 2.
Nel caso del prodotto tra matrici è necessario prestare molta attenzione alle dimensioni delle matrici. Infatti ricordiamo che se $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$ allora la matrice

$$C = A \cdot B, \qquad C \in \mathbb{R}^{m\times n}$$

si definisce nel seguente modo:

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, \qquad i = 1, \ldots, m \; j = 1, \ldots, n$$

ed è la matrice che viene calcolata scrivendo l'istruzione

```
>> C = A*B
```

In caso contrario se scriviamo

```
>> C = B*A
```

allora il programma segnala errore a meno che non sia $m = n$.
È importante notare che le operazioni $*, \hat{}$ e / operano elemento per elemento se sono precedute da un punto:

`C=A.*B` $\Rightarrow$ $c_{ij} = a_{ij}b_{ij}$

`C=A./B` $\Rightarrow$ $c_{ij} = a_{ij}/b_{ij}$

`C=A.^B` $\Rightarrow$ $c_{ij} = a_{ij}^{b_{ij}}$

5.3.1 Costanti predefinite

Come la maggior parte dei linguaggi di programmazione il Matlab ha alcune costanti predefinite cioè delle variabili che hanno un proprio valore senza che esso venga esplicitamente assegnato:

`eps`	precisione di macchina ($\simeq 2.2 \cdot 10^{-16}$)
`pi`	π (cioe' 3.14159265358979)
`i`	unita' immaginaria ($\sqrt{-1}$)
`j`	unita' immaginaria ($\sqrt{-1}$)
`realmax`	il piu' grande numero floating point ($1.7976e+308$)
`realmin`	il piu' piccolo numero floating point ($2.2251e-308$)
`inf`	infinito (∞)
`NaN`	Not a Number.

La costante `inf` è ottenuta come risultato di una divisione per zero oppure il calcolo del logaritmo di zero o se il risultato è un overflow (per esempio `2*realmax`). La costante `NaN` invece è ottenuta come risultato di operazioni matematicamente non definite come $0/0$ oppure $\infty - \infty$.
Come accade per la maggior parte dei linguaggi di programmazione anche in Matlab è possibile definire variabili il cui nome è una costante predefinita, quindi per esempio è possibile usare la variabile `i` come indice intero.

5.3.2 Operatori relazionali e logici

I seguenti sono gli operatori relazionali

$<$	minore
$>$	maggiore
$<=$	minore o uguale
$>=$	maggiore o uguale
$==$	uguale
$\sim=$	diverso

Una relazione di tipo logico assume valore 0 o 1 a seconda del fatto se essa sia rispettivamente falsa o vera. Per esempio scrivendo

```
>> 3<5
```

otterremo

```
>> 3<5
ans =
   1
```

oppure scrivendo

```
>> 1>5
```

la risposta è

```
>> 1>5
ans =
   0
```

Quando un operatore relazionale è applicato a matrici di dimensioni uguali si ottiene come risultato una matrice i cui elementi sono 1 oppure 0. Vediamo il seguente esempio:

```
>> A=[2 1 ; 0 3];
>> B=[2 -1 ; -2 3];
>> A==B
ans =
   1 0
   0 1

>> A>B
ans =
   0 1
   1 0

>> A>=B
ans =
   1 1
   1 1
```

Gli operatori logici che il Matlab consente di utilizzare sono i seguenti:

&	AND
|	OR
$\sim$	NOT

5.4 Funzioni predefinite

In MatLab è possibile utilizzare un gran numero di funzioni predefinite in grado di semplificare notevolmente la scrittura di programmi. In questo paragafo elenchiamo soltanto quelle più utilizzate nell'ambito di programmi di calcolo. Tali funzioni sono state suddivise in tre gruppi: le funzioni scalari, quelle vettoriali e quelle matriciali.

5.4.1 Funzioni scalari

Le funzioni Matlab riportate di seguito sono praticamente quelle di tipo matematico che accettano come argomento di input un numero reale (o complesso), che sintatticamente deve essere scritto tra parentesi tonde subito dopo il nome della funzione (esempio: `cos(2*pi)`, oppure `log(x+1)`).

`sin`	seno
`cos`	coseno
`tan`	tangente
`asin`	arcoseno
`acos`	arcocoseno
`atan`	arcotangente
`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolica
`asinh`	arcoseno iperbolico
`acosh`	arcocoseno iperbolico
`atanh`	arcotangente iperbolica
`exp`	esponenziale
`log`	logaritmo naturale
`log10`	logaritmo in base 10
`sqrt`	radice quadrata
`abs`	valore assoluto

`rem`	resto della divisione
`sign`	segno
`round`	arrotondamento
`floor`	parte intera inferiore
`ceil`	parte intera superiore

Tra queste funzioni appena nominate le ultime tre meritano un piccolo approfondimento; nella seguente tabella sono riportati i valori di tali funzioni per differenti numeri reali:

`x`	`round(x)`	`floor(x)`	`ceil(x)`
3.7	4	3	4
3.1	3	3	4
-4.7	-5	-5	-4
-4.3	-4	-5	-4

Osserviamo che `floor(x)` è sempre minore di `x` mentre `ceil(x)` è maggiore di `x`.
Le funzioni riportate in precedenza possono essere applicate anche a variabili di tipo vettore (o matrice), in questo si comportano come se fossero applicate a ciascun elemento del vettore (o della matrice). Per esempio se `x` è un vettore riga allora `abs(x)` è anch'esso un vettore riga le cui componenti sono i valori assoluti delle componenti del vettore `x`.

5.4.2 Funzioni vettoriali

Le seguenti funzioni Matlab operano principalmente su vettori (riga o colonna):

`max`	massimo elemento di un vettore
`min`	minimo elemento di un vettore
`sum`	somma degli elementi di un vettore
`prod`	prodotto degli elementi di un vettore
`sort`	ordinamento di un vettore
`length`	numero di elementi di un vettore
`fliplr`	inverte gli elementi di un vettore riga
`flipud`	inverte gli elementi di un vettore colonna

Le funzioni `max` e `min` possono fornire in uscita anche l'indice della componente massima (o minima) del vettore. La sintassi in questo caso è la seguente:

```
>> [massimo,k]=max(x);
>> [minimo,k]=min(x);
```

Le funzioni appena elencate possono essere applicate anche a variabili di tipo matrice, ma producono come risultato un vettore riga contenente i risultati della loro applicazione a ciascuna colonna. Per esempio scrivere

```
>> y=max(A);
```

significa assegnare al vettore (riga) `y` gli elementi massimi delle colonne della matrice `A`. Per ottenere il risultato della loro azione alle righe basta applicare la stessa funzione alla matrice trasposta `A'`. Volendo trovare il massimo elemento della matrice A si dovrebbe scrivere `max(max(A))`. La funzione `max` può essere applicata per trovare anche la posizione dell'elemento massimo (ovvero l'indice della riga e della colonna in cui si trova). Infatti, se `A` è una matrice allora

```
>> [mass,r]=max(A);
```

calcola il vettore dei massimi (ovvero `mass`) e l'indice di riga in cui si trova. Per esempio il massimo della prima colonna è `mass(1)` che si trova sulla riga `r(1)`, il massimo della seconda colonna è `mass(2)` che si trova sulla riga `r(2)`, e così via. Applicando la funzione al vettore `mass`, si trova:

```
>> [massimo,c]=max(mass);
```

in cui `massimo` è il valore cercato che si trova nella colonna `c`, la riga sarà invece `r(c)`.
Se la funzione viene applicata a due vettorix e `y`, entrambi di lunghezza n,

```
>> z=max(x,y);
```

allora le componenti del vettore risultato `z` sono pari al massimo tra le componenti dei due vettori, ovvero

$$z_i = \max(x_i, y_i), \qquad i = 1, 2, \ldots, n.$$

Analogamente se viene applicata ad un vettore `x` e ad uno scalare α

```
>> z=max(x,α);
```

allora le componenti del vettore risultato `z` sono pari a

$$z_i = \max(x_i, \alpha), \qquad i = 1, 2, \ldots, n.$$

Nella seguente tabella sono riportate altre funzioni di tipo vettoriale di uso meno comune rispetto a quelle descritte in precedenza:

`any`	Funzione logica vera se esiste un elemento del vettore diverso da zero
`all`	Funzione logica vera se tutti gli elementi del vettore sono diversi da zero
`mean`	Calcola la media del vettore
`median`	Calcola il valore mediano del vettore
`cumsum`	Calcola la somma cumulativa degli elementi del vettore

Va specificato che la media di un vettore `x` è equivalente alla seguente istruzione:

```
sum(x)/length(x)
```

Il mediano di un vettore è uguale all'elemento del vettore che si trova al centro dello stesso. Per essere chiari, se un vettore ha un numero dispari `n` di componenti, allora, una volta ordinato, il mediano coincide con l'elemento in posizione `(n+1)/2`. Se invece il numero `n` è pari allora il mediano coincide con il valor medio degli elementi in posizione `n/2` ed `n/2+1`.

5.4.3 Funzioni di matrici

Le più utili funzioni di matrici sono le seguenti:

`eig`	autovalori e autovettori
`inv`	inversa
`det`	determinante
`size`	dimensioni
`norm`	norma
`cond`	numero di condizione in norma 2
`rank`	rango
`tril`	parte triangolare inferiore
`triu`	parte triangolare superiore
`diag`	fornisce in output un vettore colonna dove è memorizzata la parte diagonale di una matrice. Se la funzione è applicata invece ad un vettore allora in uscita avremo una matrice diagonale i cui elementi principali sono quelli del vettore di input.

Le funzioni Matlab possono avere uno o più argomenti di output. Per esempio `y = eig(A)`, o semplicemente `eig(A)`, produce un vettore colonna contenente gli autovalori di A mentre

```
>> [U,D] = eig(A);
```

produce una matrice U le cui colonne sono gli autovettori di A e una matrice diagonale D con gli autovalori di A sulla sua diagonale. Anche la funzione `size` ha due parametri di output:

```
>> [m,n] = size(A);
```

assegna a `m` ed `n` rispettivamente il numero di righe e di colonne della matrice `A`.
La funzione `norm` se viene applicata ad una matrice calcola la norma 2 della stessa matrice.

```
>> norm(A);
```

È tuttavia possibile calcolare anche altre norme specificando un secondo parametro. Per esempio

```
>> norm(A,'inf');
```

calcola la norma infinito di `A`, mentre

```
>> norm(A,1);
```

calcola la norma 1 di `A` e

```
>> norm(A,'fro');
```

calcola la norma di Frobenius di `A`. Ponendo il econdo parametro uguale a 2 viene calcolata analogamente la norma 2 della matrice.
Un'altra funzione matriciale molto utile è quella che calcola la decomposizione ai valori singolari di una matrice:

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

dove `U` e `V` sono matrici ortogonali mentre `S` è una matrice diagonale, tali che

$$A = USV^T.$$

Applicando la funzione nel modo seguente

```
>> [U,S,V]=svd(A,0);
```

si ottiene la decomposizione ai valori singolari economica (ovvero la SVD compatta ma solo per la matrice `U`), mentre

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

fa lo stesso ma solo per la matrice `V`. La funzione

```
>> x=svds(A,k);
```

calcola ed assegna al vettore `x` solo i primi `k` valori singolari della matrice (ovvero i più grandi). Riassumiano nella successiva tabella altre funzioni di matrici:

`repmat`	Duplica un vattore in una determinata dimensione

5.5 Le istruzioni for, while, if e switch

Il Matlab è dotato delle principali istruzioni che servono a renderlo un linguaggio strutturato. La più importante istruzione per la ripetizione in sequenza delle istruzioni è il `for`, che ha la seguente sintassi:

```
for var=val_0:step:val_1
    lista istruzioni
end
```

La variabile *var* assume come valore iniziale *val_0*, viene eseguita la lista di istruzioni che segue, poi è incrementata del valore *step*, vengono rieseguite le istruzioni che seguono e così via, finchè il suo valore non supera *val_1*. Il valore dello *step* può essere negativo, nel qual caso il valore di *val_0* deve essere logicamente superiore a *val_1*.
Nel caso in cui il valore di *step* sia uguale a 1 questo può essere omesso, in tutti gli altri casi va necessariamente specificato. Questo vuol dire che se $step = 1$ la sintassi dell'istruzione `for` è la seguente

```
for var=val_0:val_1
    lista istruzioni
end
```

La sintassi per l'istruzione `while` è la seguente.

```
while espressione logica
    istruzioni
end
```

Le *istruzioni* vengono eseguite fintantochè l'*espressione logica* rimane vera. La sintassi completa dell'istruzione `if` è la seguente:

```
if espressione logica
    istruzioni
elseif espressione logica
    istruzioni
else
    istruzioni
end
```

I rami `elseif` possono essere più di uno come anche essere assenti. Anche il ramo `else` può mancare. Vediamo ora alcuni esempi di come le istruzioni appena descritte possono essere applicate.
Se all'interno delle istruzioni che seguono il `for` o il `while` si verifica la necessità di interrompere il ciclo delle istruzioni allora ciò può essere fatto utilizzando l'istruzione `break`.
Ultima istruzione di questo tipo (e presente solo nell'ultima versione del programma) è l'istruzione `switch` che ha lo stesso ruolo e quasi la stessa sintassi dell'omonima istruzione in linguaggio C:

```
switch variabile
   case valore_0
      istruzioni
   case valore_1
      istruzioni
   case valore_2
      istruzioni
   otherwise
      istruzioni
end
```

che, in funzione del valore assunto dalla variabile, esegue o meno una serie di istruzioni. In particolare se nessuno dei valori previsti è assunto dalla variabile allora viene previsto un caso alternativo (`otherwise`) che li contempla tutti. Vediamo il seguente esempio:

```
switch rem(n,2)
   case 0
      disp('n e'' un numero pari')
   case 1
      disp('n e'' un numero dispari')
   otherwise
      disp('Caso impossibile')
end
```

5.6 Istruzioni per gestire il Workspace

Il comando

```
>> who
```

elenca le variabili presenti nell'area di lavoro, mentre il comando

```
>> whos
```

elenca, oltre al nome delle variabili, anche il tipo e l'occupazione di memoria. Una variabile può essere cancellata da tale area con il comando

```
>> clear nome variabile
```

mentre il comando

```
>> clear
```

cancella tutte le variabili presenti nell'area di lavoro.
Premendo contemporaneamente i tasti *Ctrl* e *c* si interrompe l'esecuzione di un file Matlab. L'istruzione

```
>> save
```

salva il contenuto dell'area di lavoro (cioè le variabili e il loro valore) nel file binario `matlab.mat`. Se invece si scrive

```
>> save nomefile
```

allora tutta l'area di lavoro viene salvata nel file `nomefile.mat`. Se invece si vogliono salvare solo alcune variabili e non tutta l'area di lavoro allora è possibile farlo specificando, oltre al nome del file, anche l'elenco di tali variabili. Per esempio

```
>> save nomefile A B x
```

salva nel file `nomefile.mat` solo il contenuto delle variabili `A, B` e `x`. Scrivendo

```
>> save nomefile A B x -ascii
```

allora il file `nomefile.mat` non ha il formato binario ma ascii, e questo è utile se si vuole verificare il contenuto del file.
Per ripristinare il contenuto dell'area di lavoro dal file `matlab.mat` il comando è

```
>> load
```

mentre è possibile anche in questo caso specificare il file da caricare. Facendo riferimento all'esempio del comando `save` allora scrivendo

```
>> load nomefile
```

ripristina le variabili e il loro valore che erano stati memorizzati nel file `nomefile.mat`.

5.7 M-files

Il Matlab può eseguire una sequenza di istruzioni memorizzate in un file. Questi file prendono il nome di *M-files* perchè la loro estensione è `.m`. Ci sono due tipi di M-files: gli *script files* e i *function files*.

Script files

Uno script file consiste in una sequenza di normali istruzioni Matlab. Se il file ha come nome `prova.m` allora basterà eseguire il comando

```
>> prova
```

per far sì che le istruzioni vengano eseguite. Le variabili di uno script file sono di tipo globale, per questo sono spesso utilizzati anche per assegnare dati a matrici di grosse dimensioni, in modo tale da evitare errori di input. Per esempio se in file `assegna.m` vi è la seguente assegnazione:

```
A=[0 -2 13 4; -5 3 10 -8; 10 -12 14 17; -1 4 5 6];
```

allora l'istruzione `assegna` servirà per definire la matrice `A`.

Function files

Permettono all'utente di definire funzioni che non sono standard. Le variabili definite nelle funzioni sono locali, anche se esistono delle istruzioni che permettono di operare su variabili globali. Vediamo il seguente esempio.

```
function a = randint(m,n)
% randint(m,n) Fornisce in output una matrice
% di dimensioni m×n di numeri casuali
% interi compresi tra 0 e 9.
a = floor(10*rand(m,n));
return
```

Tale funzione va scritta in un file chiamato `randint.m` (corrispondente al nome della funzione). La prima linea definisce il nome della funzione e gli argomenti di input e di output. Questa linea serve a distinguere i function files dagli script files. Quindi l'istruzione Matlab

```
>> c=randint(5,4);
```

assegna a c una matrice di elementi interi casuali di 5 righe e 4 colonne. Le variabili m, n e a sono interne alla funzione quindi il loro valore non modifica il valore di eventuali variabili globali aventi lo stesso nome. Vediamo ora il seguente esempio di funzione che ammette un numero di argomenti di output superiore a 1,

```
function [somma, prodotto]= stat(x)
% stat(x) Fornisce in output due numeri
% contenenti la somma ed il prodotto degli
% elementi di un vettore di input x.
somma = sum(x);
prodotto = prod(x);
```

La funzione `stat` deve essere richiamata mediante l'istruzione:

```
>> [p q]=stat(y);
```

così alle variabili `p` e `q` vengono assegnate rispettivamente la somma e il prodotto degli elementi di `y`.
In definitiva se la funzione ammette più di un parametro di output allora la prima riga del function file deve essere modificata nel seguente modo:

```
function [var_0,var_1,var_2]= nomefunzione(inp_0,inp_1)
```

Come ulteriore esempio scriviamo una funzione Matlab per calcolare il valore assunto da un polinomio per un certo valore x. Ricordiamo che assegnato un polinomio di grado n

$$p(x) = a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1} + a_{n+1}x^n$$

la seguente *Regola di Horner* permette di valutare il polinomio minimizzando il numero di operazioni necessarie. Tale regola consiste nel riscrivere lo stesso polinomio in questo modo:

$$p(x) = a_1 + x(a_2 + x(a_3 + \cdots + x(a_n + a_{n+1}x)\ldots))).$$

In questo modo il numero di moltiplicazioni necessarie passa all'incirca da $O(n^2/2)$ a $O(n)$. Vediamo ora la funzione Matlab che implementa tale regola.

```
function y= horner(a,x,n)
% horner(a,x,n) Fornisce in output il valore
```

```
% di un polinomio di grado n nel punto x
% a vettore di n+1 elementi contenente i
%  coefficienti del polinomio
% x punto dove si vuol calcolare il polinomio
% n grado del polinomio
p=0;
for i=n+1:-1:1
   p=p*x+a(i);
end
y=p;
```

Per interrompere l'esecuzione di una funzione e tornare al programma chiamante si usa l'istruzione

```
return
```

Tale istruzione può essere anche quella che chiude una funzione (anche se la sua presenza non è obbligatoria).
All'interno di una funzione possono essere utilizzate due variabili, `nargin`, che è uguale al numero di parametri di input che sono passati alla funzione, e `nargout`, uguale al numero di parametri che si vogliono in output. Infatti una caratteristica sintattica è che una funzione può accettare un numero variabile di parametri di input o fornire un numero variabile di parametri di output.

5.8 Messaggi di errore, Istruzioni di Input

Stringhe di testo possono essere visualizzate sullo schermo mediante l'istruzione `disp`. Per esempio

```
disp('Messaggio sul video')
```

Se la stringa tra parentesi contiene un apice allora deve essere raddoppiato. La stessa istruzione può essere utilizzata anche per visualizzare il valore di una variabile: è sufficiente scrivere, al posto della stringa e senza apici, il nome della variabile.
I messaggi di errore possono essere visualizzati con l'istruzione `error`. Consideriamo il seguente esempio:

```
if a==0
   error('Divisione per zero')
else
   b=b/a;
end
```

l'istruzione `error` causa l'interruzione nell'esecuzione del file.
In un M-file è possibile introdurre dati di input in maniera interattiva mediante l'istruzione `input`. Per esempio quando l'istruzione

```
iter = input('Inserire il numero di iterate ')
```

viene eseguita allora il messaggio è visualizzato sullo schermo e il programma rimane in attesa che l'utente digiti un valore da tastiera e tale valore, di qualsiasi tipo esso sia, viene assegnato alla variabile `iter`.
Vediamo ora un modo per poter memorizzare in un file di ascii l'output di un M-file oppure di una sequenza di istruzioni Matlab. Infatti

```
>> diary nomefile
>> istruzioni
>> diary off
```

serve a memorizzare nel file *nomefile* tutte le istruzioni e l'output che è stato prodotto dopo la prima chiamata della funzione e prima della seconda.

5.8.1 Formato di output

In Matlab tutte le elaborazioni vengono effettuate in doppia precisione. Il formato con cui l'output compare sul video può però essere controllato mediante i seguenti comandi.

```
format short
```

È il formato utilizzato per default dal programma ed è di tipo fixed point con 4 cifre decimali;

```
format long
```

Tale formato è di tipo fixed point con 14 cifre decimali;

```
format short e
```

Tale formato è la notazione scientifica (esponenziale) con 4 cifre decimali;

```
format long e
```

Tale formato è la notazione scientifica (esponenziale) con 15 cifre decimali. Vediamo per esempio come i numeri $4/3$ e $1.2345e-6$ sono rappresentati nei formati che abbiamo appena descritto e negli altri disponibili:

```
format short            1.3333
format short e          1.3333e+000
format short g          1.3333
format long             1.33333333333333
format long e           1.333333333333333e+000
format long g           1.333333333333333
format rat              4/3


format short            0.0000
format short e          1.2345e-006
format short g          1.2345e-006
format long             0.00000123450000
format long e           1.234500000000000e-006
format long g           1.2345e-006
format rat              1/810045
```

Oltre ai formati appena visti il comando

```
format compact
```

serve a sopprimere le righe vuote e gli spazi dell'output scrivendo sullo schermo il maggior numero di informazioni possibile, in modo appunto compatto.

5.9 La grafica con il Matlab

Il Matlab dispone di numerose istruzioni per grafici bidimensionali e tridimensionali e anche di alcune funzioni per la creazione di animazioni. Il comando `plot` serve a disegnare curve nel piano xy. Infatti se `x` e `y` sono due vettori di uguale lunghezza allora il comando

```
>> plot(x,y)
```

traccia una curva spezzata che congiunge i punti `(x(i),y(i))`. Per esempio

```
>> x=-4:.01:4;
>> y=sin(x);
>> plot(x,y)
```

traccia il grafico della funzione seno nell'intervallo $[-4, 4]$.
L'istruzione `plot` ammette un parametro opzionale di tipo stringa (racchiuso tra apici) per definire il tipo e il colore del grafico. Infatti è possibile scegliere tra 4 tipi di linee, 5 di punti e 8 colori base. In particolare

`'-'`	linea continua
`'--'`	linea tratteggiata
`'-.'`	linea tratteggiata e a punti
`':'`	linea a punti
`'+'`	piu'
`'o'`	cerchio
`'x'`	croce
`'.'`	punto
`'*'`	asterisco
`'y'`	colore giallo
`'r'`	colore rosso
`'c'`	colore ciano
`'m'`	colore magenta
`'g'`	colore verde
`'w'`	colore bianco
`'b'`	colore blu
`'k'`	colore nero

Volendo tracciare per esempio un grafico con linea a puntini e di colore verde l'istruzione è:

```
>> plot(x,y,':g')
```

L'istruzione `plot` consente di tracciare più grafici contemporaneamente. Per esempio

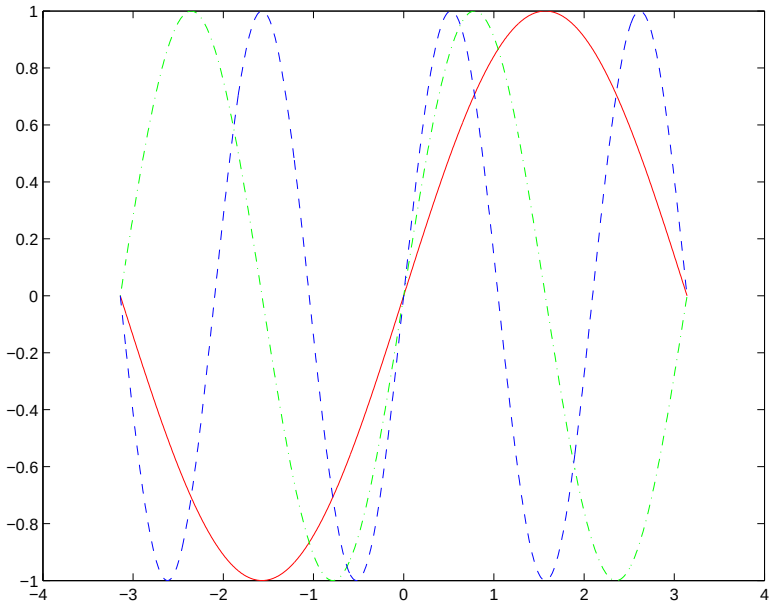


Figura 5.1:

```
>> x=-pi:pi/500:pi;
>> y=sin(x);
>> y1=sin(2*x);
>> y2=sin(3*x);
>> plot(x,y,'r',x,y1,'-.g',x,y2,'--b')
```

traccia tre grafici nella stessa figura, il primo a tratto continuo (tratto di default) rosso, il secondo verde tratteggiato e a punti, il terzo tratteggiato e di colore blu. Nella Figura 5.1 è riportato il risultato di tale istruzione.
È possibile anche decidere la larghezza del tratto, impostando il parametro `LineWidth` dell'istruzione `plot`:

```
>> plot(x,y,'LineWidth',2)
```

Il valore intero specificato indica lo spessore del tratto della curva tracciata
Vediamo ora le altre più importanti istruzioni grafiche:

```
>> title(stringa)
```

serve a dare un titolo al grafico che viene visualizzato al centro nella parte superiore della figura;

```
>> xlabel(stringa)
```

stampa una stringa al di sotto dell'asse delle ascisse;

```
>> ylabel(stringa)
```

stampa una stringa a destra dell'asse delle ordinate (orientata verso l'alto). Per inserire un testo in una qualsiasi parte del grafico esiste il comando

```
>> text(x,y,'testo')
```

che posiziona la stringa di caratteri *testo* nel punto di coordinate (x, y) (`x` e `y` non devono essere vettori). Di tale comando ne esiste anche una versione che utilizza il mouse:

```
>> gtext('testo')
```

posiziona il testo nel punto selezionato all'interno del grafico schiacciando il pulsante sinistro del mouse.
La finestra grafica può essere partizionata in un certo numero di finestre più piccole, per esempio il comando

```
>> subplot(m,n,p)
```

divide la finestra grafica in una matrice `m` $\times$ `n` di finestre più piccole e seleziona la $p-$esima. Le finestre sono numerate iniziando dalla prima in alto a sinistra e poi si procede verso destra per poi passare alla riga successiva, verso il basso.
Il grafico tracciato con il comando `plot` è scalato automaticamente, questo vuol dire che le coordinate della finestra grafica sono calcolate dal programma, tuttavia l'istruzione

```
>> axis([x_min x_max y_min y_max])
```

consente di ridefinire gli assi, e quindi le dimensioni della finestra del grafico corrente. A volte può essere utile, una volta tracciato un grafico, ingrandire alcune parti dello stesso. Questo può essere fatto utilizzando il comando `zoom`. Per attivare tale caratteristica è sufficiente il comando

```
>> zoom on
```

mentre per disattivarlo bisogna scrivere:

```
>> zoom off
```

Il funzionamento di tale istruzione è molto semplice. Una volta attivato lo zoom per ingrandire un'area del grafico è sufficiente portare il puntatore del mouse in tale area e cliccare con il tasto sinistro dello stesso. Tale operazione può essere ripetuta alcune volte (non si può ottenere l'ingrandimento un numero molto grande di volte). Per effettuare uno zoom a ritroso bisogna cliccare con il tasto destro del mouse.
La stessa funzione può essere attivata utilizzando una delle icone che si trovano nel menu della finestra grafica.
Le istruzioni grafiche del Matlab permettono di tracciare curve in tre dimensioni, superfici, di creare delle animazioni e così via. Per approfondire le istruzioni che consentono queste operazioni si può richiedere l'`help` per le istruzioni `plot3`, `mesh` e `movie`.
Nel caso di una superficie, per tracciare il grafico si devono definire due vettori, uno per le ascisse, cioè `x`, uno per le ordinate, `y`, e una matrice `A` per memorizzare le quote, cioè tale che

```
A(j,i)=f(x(j),y(i))
```

Nella Figura 5.2 è tracciato come esempio il grafico della funzione

$$f(x,y) = x(10-x) + y(10-y), \qquad 0 \le x, y \le 10.$$

Le istruzioni per tracciare tale grafico sono le seguenti:

```
x=[0:0.1:10];
y=[0:0.1:10];
n=length(x);
for i=1:n
    for j=1:n
        A(j,i)=x(j)*(10-x(j))+y(i)*(10-y(i));
    end
end
mesh(x,y,A);
xlabel('Asse x');
ylabel('Asse y');
title('f(x,y)=x(10-x)+y(10-y)');
```

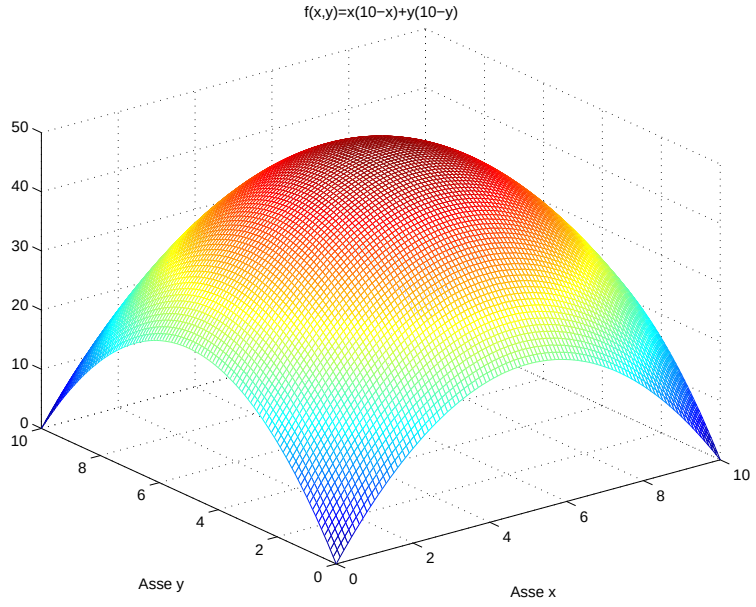


Figura 5.2:

5.10 Le immagini in MatLab

In MatLab le immagini sono memorizzate come array solitamente bidimensionali in cui ogni elemento della matrice corrisponde ad un singolo pixel. Pertanto le dimensioni di tale matrice coincidono con le dimensioni dell'immagini. Per le immagini a colori non è sufficiente utilizzare un vettore bidmensionale in quanto ad ogni pixel viene associato un colore che può essere rappresentato da un insieme di valori, in funzione della modalità con la quale sono scomposti i colori. Per esempio utilizzando lo spazo RGB in cui ad ogni colore viene associata una terna di valori, che indicano le quantità di rosso, di verde e di blu che consentono di ottenerlo, allora l'immagine viene rappresentata come una matrice tridimensionale (ovvero un tensore) composta da tre matrici bidimensionali (ognuna delle quali è associata ad un colore base). Le matrici che rappresentano le immagini sono considerate come tutte le altre matrici.
Come detto in precedenza i valori reali in MatLab sono rappresentati nel formato `double`, ovvero floating point a 64 bit in doppia precisione. Nel campo dell'Image Processing questo tipo di rappresentazione non è quello ideale, in quanto il numero di pixel di un'immagine può essere piuttosto elevato e la sua memorizzazione richiederebbe una quantità di memoria molto elevata. Per ridurre la memoria necessaria i dati relativi alle immagini sono memorizzati utilizzando i tipi `uint8` o `uint16`, ovvero come numeri interi a 8 o a 16 bit rispettivamente (quindi un ottavo o un quarto rispetto ad un numero reale in doppia precisione). Nell'Image Processing Toolbox sono supportati quattro tipi di immagini:

- Immagini indicizzate
- Immagini intensità
- Immagini binarie
- Immagini RGB.

Un'Immagine indicizzata consiste di una matrice `A` e di una matrice dei colori `colormap`, di dimensione $m \times 3$, di numeri reali in doppia precisione i cui valori appartengono all'intervallo $[0, 1]$. Ogni riga della matrice `colormap` specifica le componenti del rosso, del verde e del blu di un singolo colore. Il colore di ogni pixel viene determinato memorizzando, nella matrice `A`, l'indice della riga della variabile `colomap` in cui si trova. La matrice `colormap`

viene caricata automaticamente in memoria quando un'immagine viene letta utilizzando la funzione `imread`. I dati della matrice `A` sono valori interi. Se la matrice immagine appartiene al tipo `double` i valori che può assumere il pixel iniziano da 1 e rappresentano effettivamente la riga della matrice `colormap`, se sono interi iniziano da zero pertanto per individuare la riga bisogna aggiungere un offset pari a 1. Le istruzioni MatLab consentono di trattare maggiormente matrici immagine di tipo `double` o `uint8`, per convertire una matrice `uint16` in una matrice ad elementi reali si deve utilizzare la funzione `im2double`, per il formato `uint8` invece si deve usare la funzione `imapprox`.
Un'Immagine intensità è una matrice di dati i cui valori rappresentano le intensità, appartenenti ad un determinato range numerico. L'immagine viene memorizzata in una singola matrice in cui ad ogni elemento corrisponde un singolo pixel. I dati possono essere di tipo `double`, `uint8` oppure `uint16`. Gli elementi di una matrice intensità rappresentano i livelli di grigio, dove l'intensità 0 corrisponde al nero mentre il valore 1, 255 o 65535 al bianco (in funzione ovviamente del tipo di dato).
In un'Immagine binaria ogni pixel può assumere solo due valori discreti, 0 oppure 1. Può essere considerata come un caso particolare di matrice intensità con solo due possibili valori. Un'immagine binaria può essere memorizzata solo in array di tipo `double` o `uint8`, ma il secondo tipo risulta essere senz'altro preferibile in quanto viene utilizzata una quantità inferiore di memoria.
Un'Immagine RGB, spesso indicata come tipo `truecolor`, viene memorizzata in una matrice di dimensione $m \times n \times 3$ che serve per determinare i valori di rosso, verde e blu di ciascun pixel. Ognuna di tali componenti viene memorizzata utilizzando 8 bit (ovvero 256 valori), quindi sono necessari 24 bit per pixel. Il numero di potenziali colori rappresentabili è di circa 16 milioni. Una matrice immagine RGB può essere di tipo `double`, `uint8` oppure `uint16`. Nel primo caso i valori sono compresi tra 0 e 1. Il colore del pixel in posizione $(20, 35)$ si ottiene prendendo il valore $(20, 35, 1)$ per la componente rossa, $(20, 35, 2)$ per quella verde e $(20, 35, 3)$ per quella blu e combinandole rispettivamente.
In alcune applicazioni può essere necessario lavorare con collezioni di immagini, per esempio nel caso di video digitali che non sono altro se non un insieme di frame. Se un array è composto da un insieme di frame allora questi possono essere concatenati lungo la quarta dimensione. Per esempio un array composto da dieci immagini RGB di dimensione 640×512 ha dimensione $640 \times 512 \times 3 \times 10$. Per creare tale array si utilizza la funzione `cat`. Per esempio per creare un'immagine multiframe composta dalle immagini `I1`, `I2`,

`I3`, `I4` e `I5` si usa la funzione:

```
A=cat(5,I1,I2,I3,I4,I5);
```

Volendo estrarre un singolo frame da un'immagine multiframe si usa la funzione `multi`:

```
FRAME3=multi(:,:,:,3);
```

Appare superfluo specificare che, poter definire ed utilizzare immagini multiframe è necessario che tutti i frame siano immagini dello stesso tipo (ovvero stesse dimensioni e stessa tipologia di immagine).
MatLab consente di gestire i principali formati grafici:

- BMP (Bitmap)
- HDF (Hierarchical Data Format)
- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- XWD (X Window Dump).

La funzione `imread` legge un file immagine in uno dei formati riportati in precedenza. Solitamente l'immagine viene immagazzinata in una variabile (matrice) di valori interi di tipo `uint8`:

```
A=imread(nomefile);
```

in cui `nomefile` è una stringa (completa di estensione). Per esempio

```
A=imread('lena.jpg');
```

memorizza nella variabile `A` il file `lena.jpg` che deve ovviamente trovarsi nella directory di lavoro corrente. In modo analogo la funzione `imwrite` consente di memorizzare in un file una matrice che rappresenta un'immagine digitale, per esempio

```
imwrite(X,map,'lena.bmp');
```

memorizza nel file bitmap `lena.bmp` l'immagine digitale memorizzata nella matrice `X` indicizzato dalla mappa `map`. Il tipo di dato utilizzato è sempre quello intero ad 8 bit.
È sempre possibile convertire un'immagine in un altro tipo, per brevità riportiamo nella seguente tabella le più importanti:

`gray2ind`	Crea un'immagine indicizzata da una definita su livelli di grigio
`gray2slice`	Crea un'immagine indicizzata da una definita su livelli di grigio applicando una determinata soglia
`im2bw`	Crea un'immagine binaria da una definita su livelli di grigio oppure da una indicizzata o da una RGB
`ind2gray`	Crea un'immagine in toni di grigio da una indicizzata
`ind2rgb`	Crea un'immagine RGB da una indicizzata
`mat2gray`	Crea un'immagine in toni di grigio da una matrice generica
`rgb2gray`	Crea un'immagine in toni di grigio da una RGB
`rgb2ind`	Crea un'immagine indicizzata da una RGB

Per visualizzare le immagini sono presenti diverse funzioni, le due più importanti sono sicuramente:

`imshow`	Visualizza una matrice come immagine
`imagesc`	Visualizza una matrice come immagine

Le due funzioni hanno però una differenza sostanziale, in quanto `imshow` richiede che la matrice che viene passata come parametro sia bidimensionale e di un tipo compatibile per poter essere un'immagine, ovvero i suoi elementi devono essere numeri reali compresi tra 0 e 1 oppure interi nel range $[0, 255]$. Invece `imagesc` accetta come array in input una qualsiasi struttura con diversi tipi di dat (`uint8`, `uint16` oppure `double`) e appartenenti ad un qualsiasi range. La funzione procede a scalare opportunamente i dati.
Altre funzioni utili per l'elaborazione delle immagini sono:

`imresize`	Cambia le dimensioni (numero di pixel) di un'immagine
`imrotate`	Ruota l'immagine di un angolo specificato (in gradi)